A Programming Language for Developing Interactive Web Services

Claus Brabrand

BRICS, University of Aarhus, Denmark
Plan

- Introduction
- Runtime Model
- Dynamic Documents
- PowerForms
- Conclusion
Plan

• Introduction
• Runtime Model
• Dynamic Documents
• PowerForms
• Conclusion
What is <bigwig>?

- A domain-specific high-level programming language for developing interactive Web services.
A collection of DSLs

- C-like skeleton language with
  - Runtime system
  - Concurrency control
  - Database
  - Dynamic documents: *DynDoc*
  - Input validation language: *PowerForms*
  - Security
  - Cryptographic security
  - Syntactic-level macros
A collection of DSLs

- C-like skeleton language *with*
  - *Runtime system* (briefly)
  - *Concurrency control* (briefly)
  - Database
  - *Dynamic documents: DynDoc*
  - *Input validation language: PowerForms*
  - Security
  - Cryptographic security
  - Syntactic-level macros
DSL vs. GPL

- DSL ::= Domain Specific Language
- GPL ::= General Purpose Language

- DSL?
  - Targeted for specific problem domain
  - Abstraction level match problem domain

- Examples: Lex/Yacc, LaTeX
DSL vs. GPL

• DSL ::= Domain Specific Language
• GPL ::= General Purpose Language

• DSL?
  – Targeted for specific problem domain
  – Abstraction level match problem domain

• Examples: Lex/Yacc, LaTeX, <bigwig>
DSL Advantages
(vs. GPL + library)

- Syntax
  - Design (and restrict) expressibility
  - Syntax conveys nature of constructs

- Analysis
  - Analyze domain specific aspects

- Implementation
  - Efficient implementation
    - Rely on syntax restrictions and analysis information
Goals

- Lower development time (= cost):
  - Targeted at Web services
  - Low-level → high-level

- Increase functionality:
  - Compiler does “the dirty work”

- Increase reliability:
  - Catch errors during development
    - Runtime errors → Compile-time errors
Assumptions

• “Rules of engagement”:
  – Lowest common denominator
    • Any browser/Web server combination
  – Only include basic primitives
    • Syntactic macro language does the rest
Target Audience

• Programmers!
  – No expert Web knowledge required
  – No multiple choice questionnaires

“Reduce Web service development to a *standard* programming task.”
Core Language Features

- C-like to minimize syntactic burdens
- Not C-like features:
  - Garbage collection
  - Relations, vectors, tuples
  - Strong type checking
People

- 1x Michael I. Schwartzbach
- 1x Post Doc.
- 5x Ph.D. students
- 2x Programmers
- 2x Testers
- 1x External contributor
Plan

- Introduction
- Runtime Model
- Dynamic Documents
- PowerForms
- Conclusion
Plan

- Introduction
- Runtime Model
- Dynamic Documents
- PowerForms
- Conclusion
3 Approaches

Perl/CGI

Script-centered
3 Approaches

Page-centered

Perl/CGI

Script-centered

ASP, PHP

Microsoft Research

March 20, 2000
3 Approaches

- Session-centered
- Page-centered
- Script-centered

Mawl, <bigwig>
Perl/CGI
ASP, PHP

Microsoft Research
March 20, 2000
Script-Centered

A script

A page

A script

A page

A script
Page-Centered

A page

A page

A page
Page-Centered

“Service code embedded in tags and interpreted by specialized Web server”

• Increased level of abstraction
• Easy to add dynamics to static pages
• Scalability
Script- vs. Page-Centered

• As the service complexity increases:
  – Page-centered $\rightarrow$ Script-centered

• Script-centered:
  – default programming, escape printing.

• Page-centered:
  – default printing, escape programming.
(Fundamental) Problems

• Interpretation-based:
  – Typically no static checks
  – (Efficiency)

• Not domain specific:
  – Cannot check Web related issues

• Implicit control-flow:
  – A service = A collection of scripts/pages!
Language Requirements

• Compilation-based:
  – Static checks
  – (Efficiency)
• Domain specific:
  – Check Web related issues
• Explicit control-flow:
  – A clear service specification
Language Requirements

• Compilation-based:
  – Static checks
  – (Efficiency)

• Domain specific:
  – Check Web related issues

• Explicit control-flow:
  – A clear service specification
Session-Centered

client:

server:

Internet

one service program

Microsoft Research

<bigwig>

March 20, 2000
Hello World

service {
    session Hello() {
        html D = "Hello World!";
        show D;
    }
}

Microsoft Research                                  March 20, 2000
Hello World

service {
    session Hello() {
        show <html>Hello World!</html>;
    }
}
A Page Counter

```c
service {
    session Access() {
        shared int counter;
        string name;
        show EnterName receive [name=name];
        counter = counter + 1;
        show AccessDoc <[counter = counter];
    }
}
```
A Page Counter

:  
  if (counter == 100) {
    counter = 0;
    show Congratulations <![name = name];
  } else {
    counter = counter + 1;
    show AccessDoc <![counter = counter];
  }
  :

Microsoft Research <bigwig> March 20, 2000
CGI Shortcomings

- Stateless protocol
  - Session model requires state
- No bookmarking
  - CGI URL bookmarked, not HTML URL
- Back-button problem
  - “Step-back-in-time” does not make sense
- Long response times
  - Clients get impatient
Our Solution

• Runtime System (based on CGI)
  • “Any browser/Web server combination”

• Problems: Solutions:
  – Stateless protocol
  – No bookmarking
  – Back-button problem
  – Long response times
Our Solution

- Runtime System (based on CGI)
  - “Any browser/Web server combination”

- Problems:  
  - Stateless protocol  
  - No bookmarking  
  - Back-button problem  
  - Long response times

- Solutions:  
  - connector process
Our Solution

- Runtime System (based on CGI)
  - “Any browser/Web server combination”

- Problems:
  - Stateless protocol
  - No bookmarking
  - Back-button problem
  - Long response times

- Solutions:
  - *connector process*
  - *use html reply file*
Our Solution

• Runtime System (based on CGI)
  • “Any browser/Web server combination”

• Problems: Solutions:
  – Stateless protocol  connector process
  – No bookmarking use html reply file
  – Back-button problem use html reply file
  – Long response times

Microsoft Research  March 20, 2000
Our Solution

• Runtime System (based on CGI)
  • “Any browser/Web server combination”

• Problems: Solutions:
  – Stateless protocol connector process
  – No bookmarking use html reply file
  – Back-button problem use html reply file
  – Long response times + refresh + timeout
Runtime System

• Availability:
  – in <bigwig> compiler
  – as stand-alone package

• Underway...
  – Specialized runtime system:
    • CGI → Specialized Web server
      – efficiency, scalability
Concurrency Control

• Problem:
  – Parallel service processes.
    • Access shared resources.
    • Require synchronization.

• Example:

  \[\text{counter} = \text{counter} + 1;\]
Counter Example

counter = counter + 1;

Microsoft Research <bigwig> March 20, 2000
Counter Example

...add checkpoints

```
:  
  wait A;
  counter = counter + 1;
  wait B;
  :
```
Counter Example

...and constraints ("M2L-Str" logic)

\[ \forall t, t'': A(t) \land A(t'') \land t < t'' \Rightarrow \exists t': t < t' < t'' \land B(t') \]

: wait A; counter = counter + 1; wait B; :
Counter Example

“Ensure that service obeys constraints”

∀t,t’’: A(t) ∧ A(t’’) ∧ t<t’’ ⇒ ∃t’: t<t’<t’’ ∧ B(t’)

: wait A;
counter = counter + 1;
wait B;
:

Exclusive access
Compilation Process

\( \forall t, t'' : A(t) \land A(t'') \land t < t'' \Rightarrow \exists t': t < t' < t'' \land B(t') \)
In practice...

```c
service {
  session Access() {
    shared int counter;
    :
    counter = counter + 1;
    :
  }
}
```
Syntax Macros

```
service {
    session Access() {
        region shared int counter;
        :
        exclusive (counter) {
            counter = counter + 1;
        }
    }
}
```
Demo Example: **Mutex**

\[\text{label } A, B; \]
\[\forall t, t'': A(t) \land A(t'') \land t < t'' \Rightarrow \exists t': t < t' < t'' \land B(t')\]
Demo Example: **Mutex**

label A, B;
\( \forall t, t'': A(t) \land A(t'') \land t < t'' \Rightarrow \exists t': t < t' < t'' \land B(t') \)

```plaintext
session A() {
    while (!quit) {
        wait A;
        show Passed_A;
    }
}
```

Microsoft Research  <bigwig>  March 20, 2000
Demo Example: Mutex

\[ \text{label } A, B; \]
\[ \forall t,t'': A(t) \land A(t'') \land t < t'' \Rightarrow \exists t': t < t' < t'' \land B(t') \]

```
session A() { 
    \underline{while} (!quit) { 
        \underline{wait} A; 
        show Passed_A; 
    } 
} 
```

```
session B() { 
    \underline{while} (!quit) { 
        \underline{wait} B; 
        show Passed_B; 
    } 
} 
```
Plan

• Introduction
• Runtime Model
• Dynamic Documents
• PowerForms
• Conclusion
Plan

• Introduction
• Runtime Model
• Dynamic Documents
• PowerForms
• Conclusion
Documents

- Traditionally: `printf(…)` / `<% print(…) %>`

- Problems:
  - Only linear construction
  - Programmer/Designer tasks not separated
  - Show/Receive correspondence?
  - Legal/sensible HTML generated?
Documents

- Traditionally: `printf(…)` / `<% print(…)%>`

- Problems:
  - Only linear construction
  - Programmer/Designer tasks not separated
  - Show/Receive correspondence?
  - Legal/sensible HTML generated?
Our Solution: Document Templates

- HTML → HTML with *named gaps*

```html
<html>
  <body bgcolor=[color]>
    <h1>Hello <![what]>!</h1>
    <input type="text" name="name">
  </body>
</html>
```

...*gaps are plugged at runtime*
Dynamic Documents

- Domain specific type: \texttt{html} (with \texttt{gaps})
  - \textit{type} ::= \texttt{int} | \texttt{float} | \texttt{string} | ... | \texttt{html}

- Domain specific (sub)language: \texttt{DynDoc}
  - \textit{exp} ::= ... | \texttt{c} | \texttt{id} | \texttt{id} = \textit{exp} | \textit{exp} <[\texttt{id} = \textit{exp}]
  - \textit{stm} ::= ... | \texttt{show} \textit{exp}; | \texttt{show} \textit{exp} \texttt{receive} [ \texttt{id} = \texttt{id} ];
Plug

• Syntax:
  \[ \text{exp ::= exp \langle id = \text{exp} \rangle} \]

• Semantics: (no side-effects!)
Hello World (revisited)

```plaintext
session Hello() {
    html H = <html>Hello <[what]>!</html>;  
    html W = <html><b>World</b></html>;    
    html D;

    D = H <[what = W];
    show D;
}
```
Rec. Example: Genealogy

```html
html GenDoc = <html><ul><li>...</li></ul></html>;

html genTree(int n, string s) {
    if (n == 0) return <html></html>;
    else return GenDoc <[mother = s + “mother”,
                        mothers_tree = genTree(n-1, “mother’s”),
                        father = s + “father”,
                        father_tree = genTree(n-1, “father’s”)];
}
```
Highly Efficient Runtime Representation

• Time:
  – Plug: $O(1)$ (constant time).
  – Show: $O(|D|)$ (linear time).

• Space (maximum sharing):
  – Doc.: $O(#plugs)$ (not $O(|D|)$ !)
Show / Show-Receive

• Syntax:
  – \( stm ::= \text{show} \ exp; | \text{show} \ exp \ \text{receive} [ \ id = id ]; \)

• Semantics:
Example: EnterData

```plaintext
string name, email;
html Input = <html>
    name: <input name="name">
    email: <input name="email">
</html>;

show Input receive [name = name, email = email];
```
Documents

• Problems:
  – Only linear construction
  – Programmer/Designer tasks not separated
  – Show/Receive correspondence?
  – Legal/sensible HTML generated?
Documents

• Problems:
  – Only linear construction √
  – Programmer/Designer tasks not separated
  – Show/Receive correspondence?
  – Legal/sensible HTML generated?
Documents

• Problems:
  – Only linear construction √
  – Programmer/Designer tasks not separated √
  – Show/Receive correspondence?
  – Legal/sensible HTML generated?
Documents

• Problems:
  – Only linear construction √
  – Programmer/Designer tasks not separated √
  – Show/Receive correspondence?
  – Legal/sensible HTML generated?
Static Guarantees?

- Documents well-formed?
  - Field consistency?

- Plug operation:
  - Gap present?
  - Consistent field union?

- Show/Receive correspondence:
  - All fields received?
  - Field receive types match?
Answer: Domain Specific Analysis

• Interprocedural data-flow analysis:
  – Infer exact types of all documents in program: (gaps, fields).
  – Check:
    • documents well-formed
    • plug operations
    • show/receive correspondence
Highly Domain Specific

(Simplified) Field lattice:

- ordinary
- radio
- checkbox
- checkbox1
- nofield
- tup(F₁) ... tup(Fₙ)
- rel(F₁) ... rel(Fₙ)
- error

Microsoft Research
March 20, 2000
Documents

• Problems:
  – Only linear construction ✓
  – Programmer/Designer tasks not separated ✓
  – Show/Receive correspondence?
  – Legal/sensible HTML generated?
Documents

• Problems:
  – Only linear construction √
  – Programmer/Designer tasks not separated √
  – Show/Receive correspondence? √
  – Legal/sensible HTML generated?
Documents

• Problems:
  – Only linear construction √
  – Programmer/Designer tasks not separated √
  – Show/Receive correspondence? √
  – Legal/sensible HTML generated? (√ )
Future Plan

- Analyze generated HTML documents
  - with respect to:
    - HTML 3.2 / 4.01 / ...
    - DTD / DSD / ...

- Ensure that only “legal” documents are generated
Plan

- Introduction
- Runtime Model
- Dynamic Documents
- PowerForms
- Conclusion
Plan

• Introduction
• Runtime Model
• Dynamic Documents
• PowerForms
• Conclusion
Server-side Input Validation

send
submit
re-send
re-submit
validate 😞
re-validate 😊; compute...

Microsoft Research <bigwig> March 20, 2000
Drawbacks

• It takes time
• Excess network traffic
• Requires explicit programming

– Affects all parties involved:
  • client
  • server
  • programmer
Client-side Input Validation

(validate 😊); compute...
Drawbacks

• It takes time
• Excess network traffic
• Requires explicit programming
Drawbacks

• It takes time √
• Excess network traffic
• Requires explicit programming
Drawbacks

• It takes time √
• Excess network traffic √
• Requires explicit programming
Drawbacks

• It takes time √
• Excess network traffic √
• Requires explicit programming:
  • re-showing of pages
  • actual validation
Drawbacks

• It takes time √
• Excess network traffic √
• Requires explicit programming:
  • re-showing of pages √
  • actual validation
Drawbacks

- It takes time ✓
- Excess network traffic ✓
- Requires explicit programming:
  - re-showing of pages ✓
  - actual validation 😞
Drawbacks

- It takes time √
- Excess network traffic √
- Requires explicit programming:
  - re-showing of pages √
  - actual validation 😞

- Client, server: 😊
- Programmer: 😞
Obvious Language: JavaScript

• Why avoid JavaScript?:
  – GPL for very specific task
  – Operational form
  – Diverging browser implementations:
    • Explorer vs. Netscape
Our Solution: *PowerForms*

- Domain specific language:
  - targeted uniquely for input validation
- Declarative nature (regexps):
  - abstracts away operational details
Syntax

- \textit{decl ::= format} \textit{id = regexp ;}
- \textit{regexp ::= id | stringconst}
  \hspace{0.5cm} | union ( regexp* )
  \hspace{0.5cm} | concat ( regexp* )
  \hspace{0.5cm} | star ( regexp ) | ...
Example: Email Format

```plaintext
format Alpha = union(range('a','z'),range('A','Z'));
format Word = ...;
format Email = concat(Word,"@",Word,
                      star(concat(".",Word)))
```

Microsoft Research <bigwig> March 20, 2000
Example: EnterData (revisited)

\[
\text{html Input} = <html>
   \text{name: } <input \text{name=“name”}>
   \text{email: } <input \text{name=“email”}>
\]

</html>;

\[
\text{show Input receive } [\text{name=name, email=email}];
\]
Example: EnterData (revisited)

```plaintext
format Email = ...;

html Input = <html>
    name: <input name="name">
    email: <input name="email">
</html>;

show Input receive [name=name,email=email];
```
Example: EnterData (revisited)

```html
format Email = ...;

html Input = <html>
    name: <input name="name">
    email: <input name="email" format="Email">
</html>;

show Input receive [name=name, email=email];
```
Example: **EnterData** (revisited)

```
format Email = ...;

html Input = <html>
    name: <input name="name"> 
    email: <input name="email" format="Email">
</html>;

show Input receive [name=name,email=email];
```
Field Interdependency

Have you attended past WWW conferences? ✧ Yes ✧ No
If Yes, how did WWW8 compare? ✧ Better ✧ Same ✧ Worse

...usually only handled on server-side
PowerForms (also)

• Extend (declarative specification):
  – formats depend on values of other fields

  – Update accordingly
    • text / password: status icons updated
    • radio / checkbox: illegal options deselected
    • select: illegal options filtered (and deselected)

  – Note: Fixed-point process ($\leq$ #fields)
Example Demos

...speak for themselves...

• “Spouse”
  • Basic interdependency
• “Vowels and Consonants”
  • Select filtering
• “NYC Office”
  • Complex interdependency
PowerForms

…also as Stand-alone Tool:

- HTML
- XML regexp formats
- PowerForms
- HTML
- JavaScript (subset)
Plan

- Introduction
- Runtime Model
- Dynamic Documents
- PowerForms
- Conclusion
Plan

• Introduction
• Runtime Model
• Dynamic Documents
• PowerForms
• Conclusion
Availability?

- `<bigwig>`: 1.5 MB C source
  - for UNIX/Linux
  - Also as stand-alone packages:
    - Runtime System
    - PowerForms

- License?
  - Gnu Public License
<bigwig> Publications

- <bigwig> ...submitted
  - Runtime System WWW 8, Toronto
  - Concurrency Control FASE’98, Lisbon
  - Database (IPL’92)
  - Dynamic Documents POPL’00, Boston
  - PowerForms ...submitted
  - Macros ...underway

- Planned:
  - Security / Cryptographic security / Workflow
Current Activities

– Scalable specialized runtime system.
– External database integration.
– Better dynamic documents.
– ”Next generation” syntax macros.
– Security (information flow).
– Cryptographic protocol integration.
– Service management.
– and more...
What is <bigwig>?

– Runtime System
– Concurrency Control
– Database
– Dynamic Documents
– PowerForms
– Security / Cryptographic security
– Syntactic-level Macros

http://www.brics.dk/bigwig/
Adding a Safety Controller

Internet

client

HTTP server

connector

program

database

controller

CGI

jump

CGI

jump

CGI

jump

CGI

done

Timeout

HTML reply file

update

pass

wait

Microsoft Research

March 20, 2000

<bigwig>