Modal Types for Mobile Code

thesis defense

Tom Murphy VII

Robert Harper (co-chair)
Karl Crary (co-chair)
Frank Pfenning
Peter Sewell (Cambridge)
My thesis project is to design and implement a programming language for distributed computing based on logic.
Tell you what I did

Argue for the thesis statement

Present some of the best ideas from the work
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.
Modal type systems provide an elegant and practical means for controlling local resources in **spatially distributed computer programs**.

A spatially distributed program is one that spans multiple computers in different places.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs. They usually do so because of specific local resources that are only available in those places.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

The technology I use is a modal type system, derived from modal logic. A modal logic is one that can reason about truth from multiple simultaneous perspectives, called worlds.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs. I interpret these worlds as the places in a distributed program, which leads to a methodology I call located programming.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

Each part of the program is associated with the place in which it makes sense. The language is simultaneously aware of each place's differing perspective on the code and data.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

To show it is elegant, I present a modal logic formulated for this purpose, show how a language can be derived from it, and prove properties of these in Twelf.
Modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.

To show it is practical, I extend the language to a full-fledged programming language based on ML, specialized to web programming. I then build realistic applications in the language.
This work has a nice **end-to-end** character. The talk is arranged according to the same trajectory as the research, dissertation.
The single-vision problem
The single-vision problem

Most languages: values and code classified from a single universal viewpoint.

"integer," "file handle," etc.
The single-vision problem

Most languages: values and code classified from a single universal viewpoint.

"integer," "file handle," etc.

This monococularism leads to failures that are too early or too late.
The single-vision problem

Consider the remote procedure call.

Kurt

let
  val e = 5
  val y = h(e)
in
  print y
end

fun h(e : int) =
  e + 1

Bert
The single-vision problem

Consider the remote procedure call.

Kurt

let
  val e = 5
  val y = h(e)
in
  print y
end

Bert

fun h(e : int) =
  e + 1
The single-vision problem

Consider the remote procedure call.

Kurt

let
val e = 5
val y = h(e)
in
print y
end

Bert

fun h(e : int) =
e + 1

also, marshaling
The single-vision problem

What about local resources?

Kurt

```ml
let
  val e : file = open "thesis.tex"
  val y = g(e)
in
(* ... *)
end
```

Bert

```ml
fun g(e : file) =
  (* ... *)
```
The single-vision problem

What about local resources?

Kurt

let val e : file = open "thesis.tex"
val y = g(e)
in (* ... *)
end

Bert

fun g(e : file) = (* ... *)
The single-vision problem

What happens depends on the language.
The single-vision problem

What happens depends on the language.

**POD.** Program is rejected statically.
"You may only send plain old data."
—[DCOM/ CORBA/ XMLRPC, etc.]

**RPC.** Program fails at RPC time.
"Can't serialize local resources."
—[Java/ Acute/ Alice, etc.]
The single-vision problem

**DYN.** Program continues, might fail in function $g$.
"Decide at the last second."
—[Dynamically typed languages/ Grid/ ML, etc.]

**MOB.** Transparent mobility.
[D'caml, etc.]
Diagnosis

(POD) is overconservative.

fun g(f : file) = f

occurs in practice!

(Callbacks)
(POD) is overconservative.

fun g(f : file) = f

occurs in practice!

(RPC) admits runtime failures.

even on safe programs such as above
Diagnosis

(POD) is overconservative.

```plaintext
fun g(f : file) = f
```
occurs in practice!

(RPC) admits runtime failures.

even on safe programs such as above

(DYN) admits runtime failures.

allows

```plaintext
fun g(f : file) = f
```

fails on

```plaintext
fun g(f : file) = write(f, "hello")
```
What's going on?

Even though a file handle is a local resource, we have a single global notion (type) of file.
What's going on?

Even though a file handle is a local resource, we have a **single global notion** (type) of file.

If Bert has a file, he (reasonably) expects to be able to write to it.

*(POD) and (RPC) prevent* Bert from ever getting the file.

*(DYN) checks* that every file access is local.

*(MOB) makes every file global.*
Located programming

Instead: treat all code and data as relative to a world.

- e.g. Kurt, Burt
- allows language notion of "Kurt's file"
let
  val e : kurt's file = open "thesis.tex"
  val y = g(e)
in
  write(y, "hello")
end

fun g(e : kurt's file) =
e
Located programming

This excludes unsafe uses statically.

Kurt's code

```ml
let
  val e : kurt's file =
  open "thesis.tex"
  val y = g(e)
in
(* ... *)
end
```

Bert's code

```ml
fun g(e : kurt's file) =
  write(e, "oops")
```

type error
Kurt

let
  val e : kurt's int
  = 5
  val y = h(e)
in
  print y
end

Bert

fun h(e : kurt's int) =
  e + 1
Located programming

Kurt

let
  val e : kurt's int = 5
  val y = h(e)
in
print y
end

Bert

fun h(e : kurt's int) =
e + 1

Modal types for mobile code                  Tom Murphy VII
let
  val e : kurt's int = 5
  val y = h(e)
in
  print y
end
Located programming

Semantic question: When can we convert Kurt's t to Bert's t?

file: no, int: yes

This is not the same as marshaling
problem

solution: located programming

logic

abstract compilation

language and implementation

applications

end
Modal logic

A logic is concerned with the truth of propositions.

"A true"
Modal logic is concerned with the truth of propositions, relative to a set of worlds.

"A true @w₁"
Modal logic is concerned with the truth of propositions, relative to a set of worlds.

"A true $\Diamond w_1$"

(A proposition might only be true in some worlds because of different contingent facts at those worlds.)
Modal logic

Contingent facts are represented by hypotheses, themselves relative to a set of worlds.

\[ A \text{ true } @w_1, B \text{ true } @w_2 \vdash A \text{ true } @w_1 \]

\[ A \text{ true } @w_1, B \text{ true } @w_2 \nvdash A \text{ true } @w_2 \]
(Again, we'll think of **worlds** as **hosts** on the network.)
A proof in modal logic reasons from these distributed facts to produce a conclusion.
Modal logic

These proofs interpreted as programs appear to require non-local computation, or "action at a distance."
A novel formulation of modal logic: **Lambda 5**

- reasoning (computation) is always local
- a single rule allows us to move facts (data) between worlds

"get"
This formulation of modal logic is:

Logically faithful
(Proved sound, complete, equivalent to known logics.)
This formulation of modal logic is:

- **Logically faithful**
  (Proved sound, complete, equivalent to known logics.)

- **Computationally realistic**
  (Straightforward type-safe dynamic semantics.)
This formulation of modal logic is:

- Logically faithful
  (Proved sound, complete, equivalent to known logics.)

- Computationally realistic
  (Straightforward type-safe dynamic semantics.)

- Not enough
  (I study two extensions in detail: classical reasoning and global reasoning.)
This formulation of modal logic is:

- **Logically faithful**
  (Proved sound, complete, equivalent to known logics.)

- **Computationally realistic**
  (Straightforward type-safe dynamic semantics.)

- **Not enough**
  (I study two extensions in detail: classical reasoning and global reasoning.)

[All proofs in Twelf]
Abstract compilation applications

problem

solution: located programming

logic

abstract compilation

language and implementation

Modal types for mobile code  Tom Murphy VII
Abstract compilation

Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

Mini version of ML5
(Leaves out the complications of a full-fledged language.)
Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- **Mini version of ML5**
  (Leaves out the complications of a full-fledged language.)

- **Formalize several phases:**
  - Elimination of syntactic sugar
  - Continuation passing style transformation
  - Closure conversion
Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- Mini version of ML5
- Formalize several phases
- Feedback of ideas into logic/language

Typed compilation is a good exercise of a language's expressiveness!
Next, I take the extended modal lambda calculus and carefully show how it can be compiled.

- Mini version of ML5
- Formalize several phases
- Feedback of ideas into logic/language
- Typed compilation is a good exercise of a language's expressiveness!
- Prove static correctness for each phase

[All proofs in Twelf]
problem

solution: located programming

logic

abstract compilation

language and implementation

applications

end
ML5 is an ML-like programming language with a modal type system.
ML5 is an ML-like programming language with a modal type system.

Its implementation is specialized to web programming.

- Exactly two worlds: the browser ("home") and "server"
- AJAX-style applications (single page)
ML5 is an ML-like programming language with a modal type system.

Its implementation is specialized to web programming.

- Exactly two worlds: the browser ("home") and "server"
- AJAX-style applications (single page)
- A compiler (ML5/pgh)
- A runtime system including a web server
Modal type systems

A type system assigns a type to an expression, to classify the values it may produce.
A type system assigns a type to an expression, to classify the values it may produce.

ML5's modal type system assigns a type and world to an expression, to classify the values it may produce and the location in which it may be evaluated.
Modal type systems

M : A
shape of value that results

M : A @w
where exp can be evaluated

v : A
shape of value

v : A @w
where value can be used
js.prompt "What is your name?" : string @home

Returns a string and can only be evaluated on the web browser.
js.prompt "What is your name?" : string @home

Returns a string and can only be evaluated on the web browser.

db.lookup "name" : string @server

Returns a string and can only be evaluated on the web server.
Variables like `js.prompt` are the contingent (local) resources that form the context for type checking.
Variables like `js.prompt` are the contingent (local) resources that form the context for type checking.

\[
\text{js.prompt} : \text{string} \rightarrow \text{string} \hspace{1em} @\text{client}, \ldots
\]

\[
\vdash \text{js.prompt} : \text{string} \rightarrow \text{string} \hspace{1em} @\text{client}
\]
Local resources

The programmer can declare a local resource by importing it at a name, type and world.

```
extern val js.prompt @1: string -> string @ home
extern val js.alert @1: string -> unit @ home

extern val db.lookup @1: string -> string @ server
extern val version @1: unit -> string @ server
```
ML5 source code includes parts for both the browser and server.
ML5 source code includes parts for both the browser and server.
Execution begins in the web browser.
Control may flow to the server and back during execution.
This is done with the language construct `from ... get ....`
This is done with the language construct `from ... get ....`

```javascript
js.alert (from server get version());
```

Transfers control to server to evaluate expression.
This is done with the language construct `from ... get ....`

```javascript
js.alert (from server get version());
```

Transfers control to server to evaluate expression.

"2.0"
js.alert (from server get version());

This is done with the language construct `from ... get ....`

Transfers control to server to evaluate expression.
The `get` construct is (exclusively) how control and data flow between worlds.
The **get** construct is (exclusively) how control and data flow between worlds.

\[ \Gamma \vdash M : \text{w' addr} \ @ w \]
\[ \Gamma \vdash N : A @ w' \]
\[ \Gamma \vdash \text{from M get } N : A @ w \]

Address of remote world

(IP/ port, etc.)

Expression to evaluate
When we \texttt{get}, a value \( v : A @w' \)
becomes a value \( v : A @w \)

This only makes sense for certain types of values...

\[
\Gamma \vdash M : w' \text{ addr } @w \\
\Gamma \vdash N : A @w' \\
\hline
\Gamma \vdash \text{ from } M \text{ get } N : A @w
\]
When we get, a value $v : A \@w'$
becomes a value $v : A \@w$

This only makes sense for certain types of values...

$$
\Gamma \vdash M : w' \text{ addr} \@w \\
\Gamma \vdash N : A \@w'
$$

$$
\Gamma \vdash \text{from } M \text{ get } N : A \@w
$$
A type is mobile if every value that inhabits it is portable.

- int mobile
- w addr mobile
- A mobile
- B mobile
- (A × B) mobile

(ps: mobility has a logical justification)
A type is mobile if every value that inhabits it is portable.

\[ \text{int} \quad \text{mobile} \]
\[ \text{w addr} \quad \text{mobile} \]
\[ (A \times B) \quad \text{mobile} \]
\[ (A \rightarrow B) \quad \text{not mobile} \]
Mobile types

Would try to access a local database when called on the client!

(* string -> string @ client *)
from server get db.lookup

(A → B) mobile
Would try to access a local database when called on the client!

(ML5 statically excludes such wrong-world accesses.)
Not every function value is portable, so function types are not mobile.
Mobility vs. validity

Not every function value is portable, so function types are not mobile.

However, some particular functions are portable. We have a way to demonstrate this in the type system: validity.

(ps: validity has a logical justification)
Valid hypotheses are bindings that can be used anywhere.

\[ x \sim A \vdash x : A @w \]
Validity

Just as ML type inference automatically makes definitions maximally polymorphic, ML5 type inference makes definitions maximally valid:

```ml
(* map ~ ('a -> 'b) -> 'a list -> 'b list *)
fun \@1map f nil = nil
|\>1map f (h :: t) = (f h) :: map f t
```
To validate a binding, hypothesize the existence of a world $\omega'$. If the value is well-typed there, then it would be well-typed anywhere, since we know nothing about $\omega'$. 

\[
\begin{align*}
\Gamma, \omega' \text{ world} &\vdash v : A @ \omega' \\
\Gamma, x \sim A &\vdash N : C @ w \\
\hline
\Gamma &\vdash \text{let val } x = v \text{ in } N : C @ w
\end{align*}
\]
\(\Gamma, \omega' \text{ world}, x : \text{int} @\omega' \vdash x : \text{int} @\omega'\)

\(\Gamma, \omega' \text{ world} \vdash \text{fn } x \Rightarrow x : \text{int } \rightarrow \text{int} @\omega'\) ...

\(\Gamma \vdash \text{let val } x = (\text{fn } x \Rightarrow x) \text{ in } ... : C @w\)
Validity

\[ \Gamma, \omega' \text{ world}, x : \text{int} \@\omega' \vdash x : \text{int} \@\omega' \]
\[ \Gamma, \omega' \text{ world} \vdash \text{fn } x \Rightarrow x : \text{int} \rightarrow \text{int} \@\omega' \]
\[ \Gamma \vdash \text{let val } x = (\text{fn } x \Rightarrow x) \text{ in ... } : C \@\omega \]

Note: values only! (cf. ML value restriction)

(* r : int ref @ client *)
val r = ref 0
The judgments $x \sim A$ and $x : A \@ w$ allow us to define new types that encapsulate the notions of validity and locality.
The judgments $x \sim A$ and $x : A @w$ allow us to define new types that encapsulate the notions of validity and locality.

- A valid value of type $A$.

- An encapsulated value of type $A$ that can be used only at $w$.

(Can also have as derived forms: $\square A$, $\Diamond A$)
These are all mobile no matter what $A$ is.

- $A$ A valid value of type $A$.
- $A$ at $w$ An encapsulated value of type $A$ that can be used only at $w$.

(Can also have as derived forms: $\Box A$, $\Diamond A$)
ML5 has most of the features of core SML.

- algebraic datatypes, extensible types
- pattern matching
- mutable references
- exceptions
- mutual recursion
ML5 has most of the features of core SML.

- algebraic datatypes, extensible types
- pattern matching
- mutable references
- exceptions
- mutual recursion

... and some extensions:

- first-class continuations, threads
- quote/antiquote
Most features behave as they do in SML. We usually just need to consider whether a given type should be mobile.

```plaintext
datatype (a, b) t =
   First of a * int
| Second of (b at home) * t
```

The type \((t_1, t_2) t\) is mobile if both arms (with \(t_1, t_2\) filled in) carry mobile types.
Most features behave as they do in SML. We usually just need to consider whether a given type should be mobile.

```
datatype (a, b) t' =
  First of a * int
|  Second of (b at home) * t'
|  Third of a → b
```

The type \((t_1, t_2) \ t\) is mobile if both arms (with \(t_1, t_2\) filled in) carry mobile types.
The `exn` type and other extensible types are always mobile.

```ml
exception TagA of int
exception TagB of unit -> unit

(* ! *)
do case (from server get e) : exn of
  \@1TagA _ => ()
  | \>1TagB f => f ()
```
The `exn` type and other extensible types are always mobile.

```haskell
exception TagA of int
exception TagB of unit -> unit

(* ! *)
do case (from server get e) : exn of
  \@1TagA _ => ()
  | \>1TagB f => f ()
```

The extensible type tags give permission to retrieve the stored value.
The `exn` type and other extensible types are always mobile.

```ml
vexception TagA of int          \@3(* valid *)
exception TagB of unit -> unit \>3(* can't be valid *)

(* ! *)
do case (from server get e) : exn of
  \@1TagA _ => ()
  | \>1TagB f => f ()
```

The extensible type tags give permission to retrieve the stored value.
Another construct `put` can evaluate an expression and validate the resulting binding, but only if its type is mobile.

\[
\Gamma \vdash M : A \downarrow w \quad \text{A mobile}
\]

\[
\Gamma, x \sim A \vdash N : C \downarrow w
\]

\[
\Gamma \vdash \text{let put } x = M \text{ in } N : C \downarrow w
\]

(no communication)
Example: proxy

let
  \@1extern val db.lookup : string -> string @ server

  \>1(* plookup ~ string -> string *)
  \>1fun plookup s =
  \>1  \@2let \@3put s' = s
       \>2in \>3from server get (db.lookup s')
  \>2end

in
  \>1(* ... *)
end
Ok.
The ML5 implementation consists of a compiler, and a web server that hosts and runs the server part of programs.
The **ML5/pgh compiler** transforms the source program into client-side **JavaScript** and server-side **bytecode**.

- Elaboration and type inference
- CPS conversion
- Type and world representation
- Closure conversion
- Code generation
CPS conversion allows us to support first-class continuations and threads.

from ... get ... replaced with to ... go ...

k (from server get e)
CPS conversion allows us to support first-class continuations and threads.

from ... get ... replaced with to ... go ...:

\[ k \text{ (from server get e)} \]

becomes

\[ \text{put back} = \text{localhost} () \]
\[ \text{(to server go put ret = e)} \]
\[ \text{(to back go k(ret))} \]
Type and world representation

Marshaling uses type and world information at run-time, so we must represent these as data.

\( \alpha \text{ type}, \omega \text{ world}, \ldots \vdash A \rightarrow w \)
Marshaling uses type and world information at run-time, so we must represent these as data.

\[ \alpha \text{ type, } \omega \text{ world, ... } \vdash A @ w \]

\[ \alpha \text{ type, } u_\alpha \sim \alpha \text{ rep, } \omega \text{ world, } u_\omega \sim \omega \text{ rep, ... } \vdash A @ w \]
Closure conversion explicitly constructs closures so that we can label each piece of code.

This means abstracting over any free variables:

\[ x : A @w_1, u \sim B \vdash C \rightarrow D @w_2 \]
Closure conversion explicitly constructs closures so that we can label each piece of code.

This means abstracting over any free variables:

\[
\begin{align*}
  & x : A @w_1, u \leadsto B \\ 
  & \vdash C \rightarrow D @w_2 \\
  & \cdot \vdash (C \times A @at w_1 \times \exists B) \rightarrow D @w_2
\end{align*}
\]

modalities internalize judgments
For each piece of closed code, we use its world to decide what code we must generate for it.

@server - generate bytecode
@client - generate javascript
@w - generate both (polymorphic)

Typing guarantees that code @server will only use server resources.
The **runtime system**:  

- Web server delivers code, starts session  
- Runs server code, database, etc.  
- Marshaling and maintaining communication  
- Thread scheduling, event handling  

I'll mention these in the demo.
Modal types for mobile code
Built realistic applications with ML5.

- Evaluate its practicality, expressiveness
- Discover performance bottlenecks
- Missing features
- Feedback of ideas into language, compiler
Demo

Time!

Modal types for mobile code

Tom Murphy VII
Modal types for mobile code

Tom Murphy VII

999/999
In conclusion,
In conclusion, modal type systems provide an elegant and practical means for controlling local resources in spatially distributed computer programs.
In conclusion,

New programming language for spatially distributed computing.
In conclusion,

- New programming language for spatially distributed computing.
  - Express locality of resources
  - Statically-typed, higher order programming
In conclusion,

- **New programming language for spatially distributed computing.**
  - Express **locality** of resources
  - Statically-typed, higher order programming

- **Based on novel formulation of modal logic.**
In conclusion,

- New programming language for spatially distributed computing.
  - Express locality of resources
  - Statically-typed, higher order programming
- Based on novel formulation of modal logic.
- Mechanized theory and usable implementation.
Thanks! Questions?

Bonus topics: security tierless
Thanks! Questions?

Bonus topics: security tierless
Thanks! Questions?

Bonus topics: security  tierless
Thanks! Questions?

Bonus topics:  security  tierless
Thanks! Questions?

Bonus topics: security  tierless
Thanks! Questions?

Bonus topics: security  tierless
Thanks! Questions?

Bonus topics: security  tierless
Thanks! Questions?

Bonus topics: security tierless
Security is a difficult problem in the presence of uncooperative participants: We have no real control over what the client does with his Javascript.
Compilation obscures some security issues.

```ocaml
let
  extern format : unit -> unit @ server
val password = "my_cool_password"
put input = js.prompt ("password?")
in
  from server get
    if input = password
    then (\@1from client get js.alert ("Formatting...");
         \>1format ())
    else ()
  end
```

Modal types for mobile code  
Tom Murphy VII  
999/999  
99:99
Compilation obscures some security issues.

```ocaml
let
  extern format : unit -> unit @ server
val password = "my_cool_password"
put input = js.prompt ("password?")
in
from server get
  if input = password
  then (@1from client get js.alert ("Formatting...");
    >1format ())
  else ()
end
```

Does client source contain "my_cool_password"?
Compilation obscures some security issues.

```ocaml
let
    extern format : unit -> unit @ server
val password = "my_cool_password"
put input = js.prompt("password?")
in
from server get
    if input = password
    then (@1from client get js.alert("Formatting...");
             \>1format ())
    else ()
end
```

Server entry point 1

Server entry point 2
Types can help...

```ocaml
let extern format : unit -> unit @ server
val password : string @ server = "my_cool_password"
put input = js.prompt ("password?")
in
from server get
  if input = password
  then (@1 from client get js.alert ("Formatting..."));
    \>1 format ()
  else ()
end
```
Modal types for mobile code

Tom Murphy VII
Modal types for mobile code

Tom Murphy VII
Tierless programming

**Links** programming language (Wadler et al.)
- built-in notion of "client" and "server" (only)
- tied to function calls
- marshaling can fail at runtime

**Hop** (Serrano et al.)
- based on scheme (just one type)
- no static checks
- two `gets`, specialized to client/server
ML5 or bust

Twelf code, implementation, dissertation at

http://tom7.org/ml5/
ML5 or bust

Twelf code, implementation, dissertation at

http://tom7.org/ml5/
Twelf code, implementation, dissertation at

http://tom7.org/ml5/
A host can compute its address with `localhost`. 
A host can compute its address with `localhost`.

\[ \Gamma \vdash \text{localhost}() : w \text{ addr } @w \]
Addresses

A host can compute its address with `localhost`.

Γ ⊢ localhost() : w addr @w

For now assume we have two worlds `client` and `server` and variables in context:

client : client addr @server
server : server addr @client
client: client addr @server
server: server addr @client

from server get
(@1db.update ("greeting", "hello");
\>1 from client get
\>1 js.alert "greeting updated!")