

PiDuce

<http://www.cs.unibo.it/PiDuce/>

Samuele Carpineti, Cosimo Laneve, Leonardo Mezzina, Luca Padovani

20 december 2005

Summary

- Web Services
- Web Services in PiDuce
- Implementing PiDuce
- Orchestrators
- Pitfalls of PiDuce's type system

What is a Web service?

“A Web Service is any resource that can be found at a URL (Uniform Resource Locator)”

- idea of passive resource
- the resource is readable by the user by means of a User Agent (*Web à la CERN*)

This definition has been extended in many ways. . .

- active/dynamic documents
- query/response Web services (Google, Amazon, . . .)
- sessions

. . . still what if we were to build a Web service using another one?

Screen scraping is unreliable, not scalable, fragile, . . .

Technologies are needed for making Web services understandable by machines as well as humans

Making machines talk to each other

- Data must be dealt with in a platform-neutral way
 - ▶ data representation
 - ▶ data validation
- Services must be advertised in a machine-understandable way
- Services and clients must be described in a language that fits with the context
 - ▶ communication
 - ▶ concurrency
 - ▶ synchronization
 - ▶ data construction/deconstruction

Describing data and grammars

- XML (eXtensible Markup Language) is the *lingua franca* for inter-platform communication of semi-structured data

```
<a>
  <b>123</b>
  <c/>
</a>
```

- there exist several schema languages for defining a notion of “document valid with respect to a grammar”
 - ▶ DTDs (Document Type Definitions) based on CFG
 - ▶ XML-Schema, based on CFG with extensions/restrictions
 - ▶ Relax-NG based on regular expressions

```
<element name="a">
  <element name="b" type="integer"/>
  <element name="c" minOccurs="0" maxOccurs="1"/>
</element>
```

Describing programs

- the π -calculus is a simple, platform-independent formalism for modeling distributed systems
- it has primitives for asynchronous communication over named channels
- no commitment is made to any specific programming language, the formalism can be seen as a target language into which interesting and relevant constructs are compiled
- it permits formal investigation and analysis, it is reasonably implementable

PiDuce = XML + π -calculus



Search TerraServer

Street

City

State

GO



Longitude Latitude

-122.22229 46.23120

GO

ClickWeather.com

 Click to get
 Weather Forecast
 Maps for this point

 Powered by
 INFOWeather

Home | Advanced Find | Famous Places | Web Services | About

The National Map 84 km NE of Portland, Oregon, United States 7/21/1994

[Download](#) | [E-mail](#) | [Info](#) | [Print](#) | [Order Photo](#)

Size

[Aerial Photo](#)
[Topo Map](#)

OUT

-

-

-

-

-

-

-

-

-

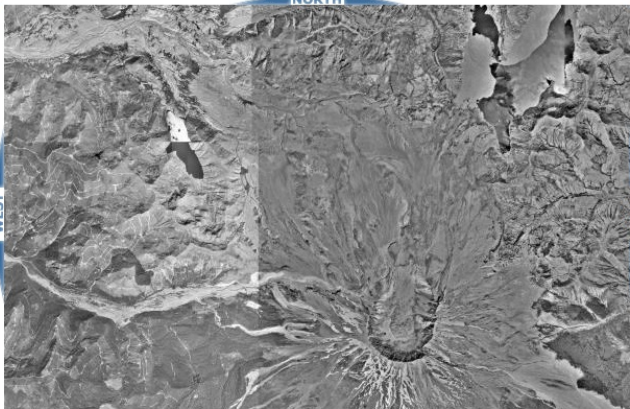
+

IN

WEST

NORTH

EAST



```
<wsdl:definitions>
  <wsdl:types> ... </wsdl:types>

  <wsdl:message name="GetTileSoapIn">
    <wsdl:part name="parameters" element="tns:GetTile" />
  </wsdl:message>
  <wsdl:message name="GetTileSoapOut">
    <wsdl:part name="parameters" element="tns:GetTileResponse" />
  </wsdl:message>

  <wsdl:portType name="TerraServiceSoap">
    ...
    <wsdl:operation name="GetTile">
      <wsdl:input message="tns:GetTileSoapIn" />
      <wsdl:output message="tns:GetTileSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

```
<wsdl:definitions>
  ...
  <wsdl:binding name="TerraServiceSoap" type="tns:TerraServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <wsdl:operation name="GetTile">
      <soap:operation soapAction="http://terraservice-usa.com/GetTile"
        style="document" />
      <wsdl:input> <soap:body use="literal" /> </wsdl:input>
      <wsdl:output> <soap:body use="literal" /> </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="TerraService">
    <wsdl:port name="TerraServiceSoap" binding="tns:TerraServiceSoap">
      <soap:address location="http://terraservice.net/TerraService2.asmx"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

A simple PiDuce Web service

With no schema annotations:

```
new add location="add" in
  add?*(a, b, res).
  res!(a + b)
```

The same PiDuce program annotated with schema information:

```
new add : <x[int], y[int], <int>> location="add" in
  add?*(x[a : int], y[b : int], res : <int>).
  res!(a + b)
```

<...> denotes a service type

A simple PiDuce client

```
new stdout : <any> location="stdout" in
import add : <x[int], y[int], <int>>
  wsdl="http://localhost:1811/add?wsdl" in
new res : <int> in
  spawn { add!(x[5], y[4], res) }
  res?(n : int).
  stdout!(n)
```

Note the difference between $x?(u).P$ and $x?*(u).P$

There is a mismatch between the published WSDL (synchronous service) and the process (asynchronous service)

First-class Web services

- WSDL 1.0 and schema languages don't deal with first-class Web services, whereas π -calculus is based on name-passing, so if

service = π -calculus channel

we can model first-class Web services naturally!

- Does it make any sense to talk about first-class Web services?
 - ▶ service replication
 - ▶ load balancing
 - ▶ fault tolerance
 - ▶ dynamic service composition
 - ▶ ...
- So what does it mean to communicate a Web service? Is it like sending a URL? The URL of what?

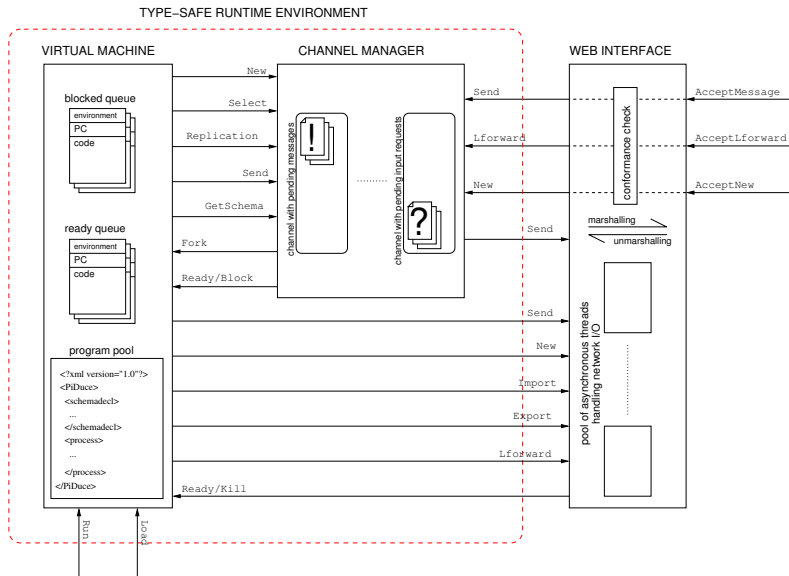
In PiDuce

- processes and channels are **static**, they stay where they have been created
- messages **travel** across the network

It seems pretty obvious but. . .

- it is not the only possibility (mobile agents, mobile code)
- it leaves “what does it mean to communicate a Web service?” unanswered
- it poses nontrivial issues in the implementation of the π -calculus (input capability)

PiDuce architecture



Virtual machine

- the virtual machine is intrinsically concurrent, *threads* in the virtual machine implement PiDuce processes
- its main data structures are
 - ▶ program pool
 - ▶ ready queue
 - ▶ blocked queue
- I/O operations are redirected to the channel manager (if the operation involves a local channel) or to the Web interface (if the operation involves a remote channel)
- the Load operation adds a program to the program pool and schedules its main thread for execution

Channel Manager

- the channel manager handles **local channels**
- each channel consists of
 - ▶ a queue of messages
 - ▶ a queue of input requests
- operations are provided for creating new channels, sending and receiving messages

Web Interface

The Web interface advertises any locally defined service defined to the world using standard technologies (interoperability)

Publishing: each channel is published in its own WSDL, PiDuce schemas are translated into XML schema

Translation: outgoing PiDuce messages are marshalled into XML documents, incoming XML documents are unmarshalled into PiDuce messages

Immigration: any incoming message/request is checked to make sure it conforms with the local schemas

Communicating a Web service means making its description (WSDL) public and sending a reference (URL) to it

Implementing output

Consider

$$x!(u)$$

and assume that u is the name of a local channel (service)

- If x is local it is sufficient to contact the local channel manager
- If x is remote
 - 1 the local Web interface publishes u making its WSDL available at a given URL (note that the WSDL includes the schema of u)
 - 2 what is sent to x is the URL of the WSDL associated with u . If the receiver needs the schema of u , that can be retrieved from u 's WSDL
 - 3 the remote Web interface downloads the type of u from its WSDL and checks that it is “compatible” with x 's type
 - 4 u is locally delivered in x 's message queue

Implementing input

Consider

$$x?(u).P$$

- easy if x is local! An input request is enqueued in x 's request queue, P is blocked until a message arrives on x
- what if x is a remote channel?

$$\boxed{x!(u) \mid x?(v).P} \rightarrow \boxed{P\{v/u\}}$$

Note that remote input cannot be detected statically:

$$x?(u).u?(w).P$$

Even if x is local, who knows where u is coming from. . .

Linear forwarding

We rewrite

$$x?(u).P$$

into

$$\begin{array}{l} \text{new } y \text{ in} \\ \text{spawn}\{ x?(v).y!(v) \} \\ y?(u).P \end{array}$$

Now $y(u).P$ is a local input operation. What is the upshot?

$x?(v).y!(v)$ is a *linear forwarder* $x \multimap y$

$x \multimap y$ is a small process with finite behavior which can migrate to x 's location and execute remotely

Synchronization

Assume we have three parallel activities A , B , and C and we want to execute P or Q depending on whoever finishes first between both A and B and both B and C

$$A = \dots a!()$$
$$B = \dots b!()$$
$$C = \dots c!()$$
$$a?().b?().P \quad b?().c?().Q$$

- this encoding is not correct: if B completes then A completes and C never completes we have a **deadlock!**
- rewriting doesn't always help, competing processes are not always known at compile time
- we need a way of expressing an atomic input from multiple channels:

$$\text{join}\{ a?() \& b?() \triangleright P + b?() \& c?() \triangleright Q \}$$

(see Petri nets)

Example: lock

Lock definition:

```
new mutex, lock, unlock in
  spawn{ mutex!() }
  join*{
    mutex?() & lock?(r) ▷ r!()
    + unlock?() ▷ mutex!()
  }
```

Lock usage:

```
new r in
  spawn{ lock!(r) }
  r?().P
```

where P does

```
spawn{ unlock!() }
```

when it's done using the critical section

Example: one-place buffer

Buffer definition:

```
new empty, full, put, get in
  spawn{ empty!() }
  join*{
    empty?() & put?(v) ▷ full!(v)
    + full?(v) & get?(r) ▷ spawn{ empty!() } r!(v)
  }
```

(see Objective Join Calculus)

Implementing joined channels

Same problems as for simple input operations, same solution?

What if

$$\text{join}\{ x?(u) \& y?(v) \triangleright P \}$$

is encoded into

$$\begin{aligned} &\text{new } x', y' \text{ in} \\ &\quad \text{spawn}\{ x \multimap x' \} \\ &\quad \text{spawn}\{ y \multimap y' \} \\ &\quad \text{join}\{ x'?(u) \& y'?(v) \triangleright P \} \end{aligned}$$

?

It doesn't work and that's no surprise (distributed consensus). Bummer!

Smooth orchestration

We generalize linear forwarders into **smooth orchestrators**

The process

$$\text{join}\{ x?(u) \& y?(v) \triangleright P \}$$

is encoded into

$$\begin{aligned} &\text{new } z \text{ in} \\ &\quad \text{spawn}\{ \text{join}\{ x?(u) \& y?(v) \triangleright z!(u, v) \} \} \\ &\quad z?(u, v).P \end{aligned}$$

where $\text{join}\{ x?(u) \& y?(v) \triangleright z!(u, v) \}$ is a smooth orchestrator that migrates to x 's and y 's location

Beware: x and y must be **co-located!**

Example: supplier/manufacturer/bank interaction

Supplier definition:

```
buy?(item, x).
  new voucher@item in
    spawn{ x!(voucher, amount) }
    join*{
      voucher?(u) & item?(v) ▷
        spawn{ deliver!(u, v) }
        record!(u, v)
    }
}
```

PiDuce schemas and type-checking

Assume we have a Web service x converting inches, picas and points into centimeters. It would accept messages belonging to the schema

$$\text{in}[\text{int}] + \text{pc}[\text{int}] + \text{pt}[\text{int}]$$

Assume we have a message m that we know being either an `in` or a `pt` element. It would belong to the schema

$$\text{in}[\text{int}] + \text{pt}[\text{int}]$$

What about $x!(m)$? It is well-typed, because

$$\text{in}[\text{int}] + \text{pt}[\text{int}] <: \text{in}[\text{int}] + \text{pc}[\text{int}] + \text{pt}[\text{int}]$$

`<:` is the *subschema relation* (similar to OO *subtyping*)

Channel schemas

Since channels (services) are first-class objects, they must have a schema too!

$$\langle S \rangle^\kappa$$

is the schema of channels carrying data of type S and κ is the channel capability:

- I input capability
- O output capability
- IO input/output capability

What about the subschema relation with channel types? When is it safe to use a channel of type $\langle S \rangle^\kappa$ when one of type $\langle T \rangle^{\kappa'}$ is expected?

Channel schemas and subschema relation

Assume

$$x : \langle\langle T \rangle^I \rangle \quad u : \langle S \rangle^I$$

When is $x!(u)$ well-typed?

Whoever receives u will think that it has type $\langle T \rangle^I$, so is prepared to received data of type T from u

Co-variance:

$$\langle S \rangle^I <: \langle T \rangle^I \iff S <: T$$

Assume

$$x : \langle\langle T \rangle^0 \rangle \quad u : \langle S \rangle^0$$

When is $x!(u)$ well-typed?

Whoever receives u will think that it has type $\langle T \rangle^I$, so is authorized to send data of type T on u

Contra-variance:

$$\langle S \rangle^0 <: \langle T \rangle^0 \iff T <: S$$

Why all this fuss about schemas?

During immigration the Web interface has to check whether incoming messages conforms with the local schemas

- checking that a plain XML document (without channel values) x belongs to a schema S can be done in linear time (w.r.t. x 's size)
- checking that a channel u belongs to a schema $\langle T \rangle$ entails computing the subschema relation

How **hard** is it to compute the subschema relation?

The subschema relation is **exponential**

The hard case is the sequence

$$L[S], L'[S'] <: \sum_{i \in I} L_i[T_i], L'_i[T'_i]$$

One can prove that

$$A \times B \subseteq \bigcup_{i \in I} C_i \times D_i \iff \forall J \subseteq I : A \subseteq \bigcup_{j \in J} C_j \vee B \subseteq \bigcup_{j \in I \setminus J} D_j$$

The **label-determinedness** condition enforces that

$$i \neq j \Rightarrow L_i \cap L_j = \emptyset \quad (C_i \cap C_j = \emptyset)$$

Under this condition, the subschema relation is **polynomial**

- <http://www.cs.unibo.it/PiDuce/>
- A. Brown, C. Laneve, G. Meredith, “PiDuce: a process calculus with native XML datatypes”, in Proceedings of WS-FM'05
- C. Laneve, L. Padovani, “Smooth Orchestrators”, in Proceedings of FOSSACS'06
- C. Laneve, S. Carpineti, “A basic contract language for Web services”, in Proceedings of ESOP'06