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>
```

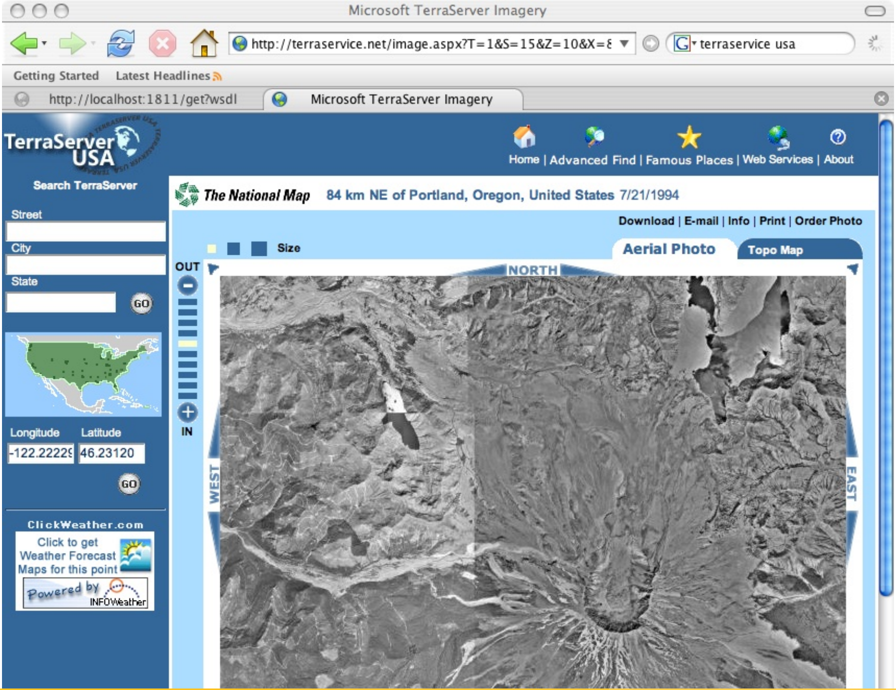
- there exist several schema languages for defining a notion of "document valid with respect to a grammar"
  - DTDs (Document Type Definitions) based on CFG
  - XML-Schema, based on CFG with extensions/restrictions
  - Relax-NG based on regular expressions

```
<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


Microsoft TerraServer Imagery
http://terraservice.net/image.aspx?T=1&S=15&Z=10&X=ε
terraservice usa
Getting Started
Latest Headlines
http://localhost:1811/get?wsdl
Microsoft TerraServer Imagery
TerraServer USA
Home | Advanced Find | Famous Places | Web Services | About
Search TerraServer
Street
City
State
GO
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
NORTH
WEST
EAST
Longitude
Latitude
-122.2222
46.23120
GO
ClickWeather.com
Click to get Weather Forecast Maps for this point
Powered by INFOWeather

http://terraservice.net/TerraService2.asmx?WSDL

```
<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>
```

http://terraservice.net/TerraService2.asmx?WSDL

```
<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

$$\text{service} = \pi\text{-calculus channel}$$

  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?

PiDuce architecture

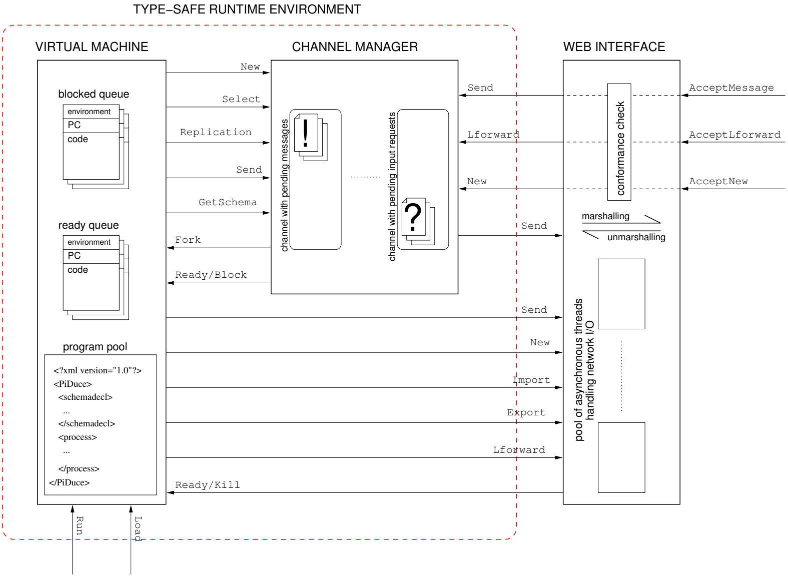
In PiDuce

- 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)

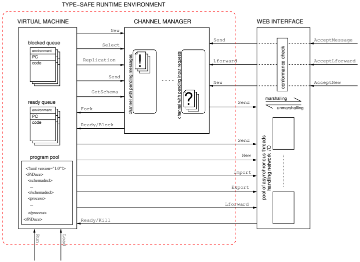
Publishing: each channel is published in its own WSDL, PiDuce schemas are translated into XML schema

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

Modularity for flexibility



- the channel manager and the Web interface can be used as libraries from native programs
- as the Web interface becomes obsolete (technology evolves) it can be easily replaced
- the virtual machine and the channel manager are type-safe. Nothing wrong can happen once in the red zone

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

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

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

$$A = \dots a!()$$
$$B = \dots b!()$$
$$C = \dots c!()$$

$$a?().b?().P \qquad b?().c?().Q$$

- 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:

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

(see Petri nets)

Example: lock

Lock definition:

$$
\begin{array}{l}
\texttt{new}\ \texttt{mutex}, \texttt{lock}, \texttt{unlock}\ \texttt{in} \\
\quad \texttt{spawn}\{\ \texttt{mutex}!()\ \} \\
\quad \texttt{join}{*}\{ \\
\qquad\quad \texttt{mutex}?()\ \&\ \texttt{lock}?(r) \triangleright r!() \\
\qquad + \texttt{unlock}?() \triangleright \texttt{mutex}!() \\
\quad \}
\end{array}
$$

Lock usage:

$$
\begin{array}{l}
\texttt{new}\ r\ \texttt{in} \\
\quad \texttt{spawn}\{\ \texttt{lock}!(r)\ \} \\
\quad r?().P
\end{array}
$$

where P does

$$\texttt{spawn}\{\ \texttt{unlock}!()\ \}$$

when it's done using the critical section

Example: one-place buffer

Buffer definition:

$$
\begin{array}{l}
\texttt{new}\ \texttt{empty}, \texttt{full}, \texttt{put}, \texttt{get}\ \texttt{in} \\
\quad \texttt{spawn}\{\ \texttt{empty}!()\ \} \\
\quad \texttt{join}{*}\{ \\
\qquad\quad \texttt{empty}?()\ \&\ \texttt{put}?(v) \triangleright \texttt{full}!(v) \\
\qquad + \texttt{full}?(v)\ \&\ \texttt{get}?(r) \triangleright \texttt{spawn}\{\ \texttt{empty}!()\ \}\ r!(v) \\
\quad \}
\end{array}
$$

(see Objective Join Calculus)

Implementing joined channels

Same problems as for simple input operations, same solution?
What if

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

is encoded into

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

?

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

Smooth orchestration

We generalize linear forwarders into smooth orchestrators

The process

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

is encoded into

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

where $\texttt{join}\{\ x?(u)\,\&\,y?(v) \triangleright z!(u,v)\ \}$ 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:

$$
\begin{array}{l}
\texttt{buy?}(\texttt{item}, x). \\
\quad \texttt{new voucher@item in} \\
\quad\quad \texttt{spawn}\{\ x!(\texttt{voucher}, \texttt{amount})\ \} \\
\quad\quad \texttt{join}{*}\{ \\
\quad\quad\quad \texttt{voucher?}(u)\,\&\,\texttt{item?}(v) \triangleright \\
\quad\quad\quad\quad \texttt{spawn}\{\ \texttt{deliver!}(u, v)\ \} \\
\quad\quad\quad\quad \texttt{record!}(u, v) \\
\quad\quad \}
\end{array}
$$

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

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

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

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

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

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

`<:` 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

$$x : \langle\langle T\rangle^{\mathtt{I}}\rangle \qquad u : \langle S\rangle^{\mathtt{I}}$$

When is $x!(u)$ well-typed?
Whoever receives u will think that it has type $\langle T\rangle^{\mathtt{I}}$, so is prepared to received data of type T from u

Co-variance:

$$\langle S\rangle^{\mathtt{I}} <: \langle T\rangle^{\mathtt{I}} \iff S <: T$$

Assume

$$x : \langle\langle T\rangle^{\mathtt{O}}\rangle \qquad u : \langle S\rangle^{\mathtt{O}}$$

When is $x!(u)$ well-typed?
Whoever receives u will think that it has type $\langle T\rangle^{\mathtt{I}}$, so is authorized to send data of type T on u

Contra-variance:

$$\langle S\rangle^{\mathtt{O}} <: \langle T\rangle^{\mathtt{O}} \iff T <: S$$

Complexity matters

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

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

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**

Bibliography

- `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