The Chor programming language

Fabrizio Montesi <fmontesi@imada.sdu.dk>



Background and Motivations

• Distributed system:

a network of endpoints that communicate by exchanging messages.

• Widespread! Let's see some examples...

The Internet

• The Internet is a distributed system:



The Internet

• The Internet is a distributed system:



Your Computer

• The OS and apps in your computer (or phone):



Your browser

• Even applications can be distributed systems. Google Chrome:



Distributed systems are big!

System	Number of endpoints			
My computer	160			
A house	Hundreds			
A company	Thousands (or millions)			
The Internet	At least 20 billions			

Endpoint Programming

• How do we program all these endpoints?

• We write a program for each.

• Programs interact by sending and receiving messages.

Endpoint Programming: example

• Alice wants to know the price of a book from Amazon.

Alice				Amazon				
send	"rabbits"	to	Amazon	recv	book	from	Alice	
recv	price	from	Amazon	send	price(book)	to	Alice	







- Endpoint programming is error-prone.
- Mismatching of input/output actions leads to a deadlock.

Alice				Amazon				
recv	price	from	Amazon	re	CV	book	from	Alice
send	"rabbits"	to	Amazon	se	nd	price(book)	to	Alice

- Endpoint programming is error-prone.
- Mismatching of input/output actions leads to a deadlock.

Alice				Amazon				
recv	price	from	Amazon	recv	book	from	Alice	
send	"rabbits"	to	Amazon	send	price(book)	to	Alice	

• Creating deadlocks is easy.

• Detecting deadlocks is hard [Kobayashi, 06].

Famous bugs

- Therac-25: a machine for radiation therapy in the 80s.
- Unsafe communications caused excessive amounts of radiation (100x).
- At least 6 accidents, 3 deaths.



Famous bugs

- 2003: Blackout in Northeast America.
- Started from a communication bug in a monitoring station.
- Affected 55 million people.
- 11 deaths.
- At least 7 billion USD lost.



• In general, we would like systems to be **safe**.

• **Safe** = no bugs given by wrong sending/receiving actions.

How does it happen?

• Human error.



How does it happen?

• Human error.



• More quality control?











	Alice		
send	"rabbits"	to	Amazon
recv	price	from	Amazon
send	price	to	Bob
recv	address	from	Charlie
recv	text	from	Bob

Amazon

recv	book	from	Alice
send	price(book)	to	Alice
recv	price(book)	from	Charlie
send	<i>text</i> (book)	to	Charlie

		Bob		
recv	price		from	Alice
recv	text		from	Charlie
send	text		to	Alice

	Charlie					
recv	price	from	Alice			
send	<i>money</i> (price)	to	Amazon			
recv	text	from	Amazon			
send	address	to	Alice			



<pre>send "rabbits" recv price send price recv address recv text send "horses"</pre>		to from from from to	Amaz Amaz Bob Char Bob Amaz	recv send recv recv send recv	book price price price money text	(book) (book) (price)	from to from from to from	Alice Alice Charl Alice Amazo Amazo	ie n n	
<pre>recv price2 send "spiders" send price3 recv address1 recv price send money(pri recv text</pre>	recv recv recv send send	price text price text text text	9	send recv recv send recv send	addre price price money text addre	(price)	to from from to from	Alice Alice Alice Amazo Amazo Alice	n n	
<pre>send address recv price send money(pri recv text send address recv text2 recv price send price recv address recv text</pre>	recv send recv send recv send recv send recv recv	price money price mc se te re ac se te re ac re pi se pi re	<pre> (pri (pri nd "r cv pr nd pr cv ac cv te nd "f cv pr cv te nd "f cv pr cv pr cv pr nd "f cv pr nd "f cv pr nd "f cv pr nd "f cv pr nd</pre>	send recv send rabbit rice rice ddress ext norses rice2 spider	mone text ¹ addı ² s"	recv prisend mon to from to from from to from	ice ney(pric Amazon Amazon Bob Charlie Bob Amazon Amazon Amazon	f e) t f f f f f) t f) t t	from co from from from co from co	Alice Amazon Alice Alice Amazon Alice Amazon Amazon Alice
	send recv send	te se te re te re se re se re se	nd pr cv ac cv pr nd ma cv te nd ac cv pr nd ma cv te nd ac cv te nd ac	rice3 ddress rice oney(p ext ddress rice oney(p ext ddress	1 rice) rice)	to from from to from to from to from	Bob Charlie Alice Amazon Alice Alice Amazon Amazon Alice	f t f f t	from co from co co	Amazon Alice Amazon Amazon Alice Alice



		S	end "ral 1	cecv bo	ok	fr	om Alice	
.		r	ecv pric s	send pr	<i>cice</i> (book) to	Alice	
send "rai recv boo	ok	fr s	enc recv 1	cecv pr	<i>ice</i> (book) fr	om Charli	ie
recy pric send pr.	<i>ice</i> (book)	to r	ecv recv 1	cecv pr	ice	fr	om Alice	
senc recv recv pr.	<i>ice</i> (book)	fror	ecv rear	d Wrah	\cdot	+-		.ce
recv recv pr	ice.	fr s	enc re sen	a rap.	DILS	to	Amazon	ızon
recy recy and "rabh	$a^{+}a^{-}$	to	ecv se rec	v pric	e	I FOM	Amazon	ızon
senc re send labt)S	from	enc se sen	a pric	e	to	Charlia	.ce
recv se recv price	5	S	enc re rec		ess	from	Charite	.ce
send "ral recv boo	ok	fr ^r	ecv se rec	v lext	~~~″	ITOM	BOD	ızon
recv pric send pra	<i>ice</i> (book)	to r	ecv re sen		ses	to	Amazon	.ce
send recv recv pri	<i>ice</i> (book)	fra ^{Se}	enc se rec	v pric	ez dene"	I FOM	Amazon	ızon
recv recv recv pr	· · · · ·	r	eav re sen	a spi	ders		Amazon	ızon
recy recy recy	send "rai	recv bo	ok	fr	om Alice	from	Charlie	.ce
senc re send "rabk	recv pri	send pr	<i>ice</i> (book)	to	Alice	from		ızon
recv se recv price	senc recv	recv pr	<i>ice</i> (book)	fr	om Charl:		ALICE	.ce
senc se send price	recv recv	recy pr	ice.	fr	om Alice		Amazon	ızon
senc re recv addre	recv reat	nd "rah	nits"	to	Amazon	ice om	Aliao	ızon
recv se recv text	senc re se	cy price	2	from	Amazon	ızon	Alice	.ce
recv re send "hors	recv se re	nd price	~	to	Bob	izon om	ATICE	.ce
senc se recv price	senc se se	cv addre	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	from	Charlie	.ce	Amazon	
recv r. send "spic	senc re re	cv text		from	Bob	.ce Off	Alice	
send se send price	recv se - se	nd "hors	ses"	to	Amazon	lzon	ATICE	
recv re recv addre	recv re be	cv price	-2	from	Amazon	.ce		
send se recv price	senc se re	end "spic	ders"	to	Amazon	lzon		
recv resend money	recv re se	nd price	-3	to	Bob	izon		
send recv text	senc st se	cv addre	ess1	from	Charlie	.ce		
recv re send addre	recv re	cv price	2	from	Alice	Izon		
recv se recv price	senc se re	and money	v(price)	to	Amazon	.ce		
send re send money	recv re se	cv text	(P1100)	from	Amazon	Izon		
recv se recv text	senc re re	addre	233	to	Alice	izon		
recv t send addre	recv re se	cv price		from	Alice	.ce		
	recv se re	nd money	v(price)	to	Amazon	.ce		
	senc re se	cv text		from	Amazon			
	recv se re	addre	255	to	Alice			
	recv t se				MITCE			

Distributed bugs are hard to spot!

• Avoiding distributed bugs, like deadlocks, is hard [Kobayashi, 06].

• We need **tools** to deal with this.

Problem

Developing safe distributed systems by programming endpoints separately is error-prone.

Choreographic Programming: the idea

- A single program for defining the behaviour of many endpoints [W3C, 05]
- Our previous example as a choreography:

alice."rabbits" \rightarrow amazon.book ; amazon.price(book) \rightarrow alice.price

- A single program for defining the behaviour of many endpoints [W3C, 05]
- Our previous example as a choreography:

alice."rabbits" → amazon.book ;
amazon.price(book) → alice.price

- A single program for defining the behaviour of many endpoints [W3C, 05]
- Our previous example as a choreography:

- A single program for defining the behaviour of many endpoints [W3C, 05]
- Our previous example as a choreography:

alice."rabbits" \rightarrow amazon.book ; amazon.price(book) \rightarrow alice.price

• Defines **what** communications we want to happen, rather than **how** to implement them.

Choreographic Programming

• [Mendling and Hafner, 05] [Qiu et al., 07] [Carbone et al., 07] [Lanese et al., 08] ...

Choreographic Programming

• Write a choreography.

Choreography

Choreographic Programming

• Write a choreography.

• Compile it to an executable distributed implementation.


alice."rabbits" \rightarrow amazon.book ; amazon.price(book) \rightarrow alice.price









Correct by construction!

- Correct pairing of I/O actions prevents deadlocks!
- Promising approach.
- Instead of detecting deadlocks after programming, prevent deadlocks from being written.

The momentum of choreographies

• Great momentum: there are lots of choreography models out there.

[Busi et al., 05] [Busi et al., 06] [Qiu et al., 07] [Bravetti and Zavattaro, 07] [Carbone et al., 07] [Lanese et al., 08] [Basu et al., 11] [Dalla Preda et al., 14] ...

• Bisimulation, session types, web services, adaptability, ...

The momentum of choreographies

• Dawn of a new paradigm?

	Sequential programming	Distributed programming
High level	C, Java, ML, Compiler	
Low level	Assembly	Endpoint Programming

The momentum of choreographies

• Dawn of a new paradigm?



Towards a new paradigm

• Lots of promising formal models.

• We need tools to evaluate the paradigm.

Methodology

In this work

• We present the Chor language: **Chor**

• A programming language based on choreographies.

• Eclipse-based IDE with on-the-fly deadlock verification.

• A compiler for generating executable Jolie code.

• An initial evaluation of the choreographic programming paradigm.

Development methodology



- We describe the behaviour of **processes** in a system.
- Each process has a local state.
- Processes take part to multiparty conversations, tracked as **sessions**.
- Sessions are **started** through **public channels** (e.g., URLs).
- Both sessions and processes can be dynamically created.

An example

• Alice and Bob buy together a book on Amazon.

An example



```
Buyer \rightarrow Seller : string; // Ask the price
Seller \rightarrow Buyer : int; // Get the price
Buyer \rightarrow Helper : int; // Contribution
Helper \rightarrow Seller : { // Choice
ok: Seller \rightarrow Helper: string;
end,
ko: end
}
```

Buyer	\rightarrow	Seller	•	strin	ng; // Ask the price
Seller	\rightarrow	Buyer	:	<pre>int;</pre>	// Get the price
Buyer	\rightarrow	Helper	:	<pre>int;</pre>	// Contribution
Helper	\rightarrow	Seller	:	{	// Choice
				ok:	Seller \rightarrow Helper: string;
					end,
				ko:	end
				}	

Buyer	\rightarrow	Seller	:	strin	g; //	Ask the p	price
Seller	\rightarrow	Buyer	•	<pre>int;</pre>	//	Get the p	price
Buyer	\rightarrow	Helper	:	<pre>int;</pre>	//	Contribut	tion
Helper	\rightarrow	Seller	•	{	//	Choice	
				ok:	Seller	\rightarrow Helper:	<pre>string;</pre>
					end,		
				ko:	end		
				}			

Buyer	\rightarrow	Seller	:	strin	g; // Ask the price
Seller	\rightarrow	Buyer	:	<pre>int;</pre>	// Get the price
Buyer	\rightarrow	Helper	•	<pre>int;</pre>	// Contribution
Helper	\rightarrow	Seller	•	{	// Choice
				ok:	Seller \rightarrow Helper: string;
					end,
				ko:	end

Buyer	\rightarrow	Seller	:	strin	ng; // Ask the price
Seller	\rightarrow	Buyer	:	<pre>int;</pre>	// Get the price
Buyer	\rightarrow	Helper	•	<pre>int;</pre>	// Contribution
Helper	\rightarrow	Seller	•	{	// Choice
_				ok:	Seller - Helper: string ;
					end,
				ko:	end
				}	

Buyer	\rightarrow	Seller	:	strin	ig;	/ /	Ask the price
Seller	\rightarrow	Buyer	•	<pre>int;</pre>	,		Get the price
Buyer	\rightarrow	Helper	•	<pre>int;</pre>	,	/ /	Contribution
Helper	\rightarrow	Seller	•	{	,	/ /	Choice
				ok:	Seller	-	→ Helper: string ;
					end,		
				ko:	end		
				}			

Buyer	\rightarrow	Seller	•	strin	g; // Ask the price
Seller	\rightarrow	Buyer	•	<pre>int;</pre>	// Get the price
Buyer	\rightarrow	Helper	•	<pre>int;</pre>	// Contribution
Helper	\rightarrow	Seller	•	{	// Choice
				ok:	Seller \rightarrow Helper: string;
					end,
				ko:	end
				}	



Choreography

```
Buyer \rightarrow Seller : string; // Ask the price

Seller \rightarrow Buyer : int; // Get the price

Buyer \rightarrow Helper : int; // Contribution

Helper \rightarrow Seller : { // Choice

ok: Seller \rightarrow Helper: string;

end,

ko: end

}
```



• Let a be a public URL (e.g., www.amazon.com) with the protocol above.





Receiver

Sender

Session





Buyer Seller Buyer Helper	$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array}$	Seller Buyer Helper Seller	•	<pre>string; // Ask the pric int; // Get the pric int; // Contribution { // Choice ok: Seller → Helper: st: end, ko: end }</pre>	e rir	ng;	
alice[alice. amazon alice. if (co	Buye "rab . <i>pri</i> (pri ntri	er], bob[bits" . <i>ce</i> (book) .ce/2) .b < 100\$	He	<pre>per] start amazon[Seller] → amazon.book → alice.price → bob.contrib</pre>	•	a(k) k k k	;;;;

Condition Evaluator









Buyer	\rightarrow	Seller	:	strin	ig; /	/	Ask the price
Seller	\rightarrow	Buyer	:	<pre>int;</pre>	/	/	Get the price
Buyer	\rightarrow	Helper	:	<pre>int;</pre>	/	/	Contribution
Helper	\rightarrow	Seller	:	{	/	/	Choice
				ok:	Seller		→ Helper: string;
					end,		
				ko:	end		
				}			

alice[Buyer], bob[Helpe	er] start amazon[S	eller]: a(k)	;
alice."rabbits"	→ amazon.book	: k	;
amazon.price(book)	alice.price	: k	;
alice.(price/2)	bob.contrib	: k	;
if (contrib < 100\$)@bob	C		
bob	→ amazon	: k[ok]	;
amazon. <i>text</i> (book) -	<pre>> bob.text</pre>	: k	
else			

Buyer	\rightarrow	Seller	•	strin	g; // Ask the price
Seller	\rightarrow	Buyer	:	<pre>int;</pre>	// Get the price
Buyer	\rightarrow	Helper	•	<pre>int;</pre>	// Contribution
Helper	\rightarrow	Seller	•	{	// Choice
				ok:	Seller -> Helper: string;
					end,
				ko:	end
				}	

alice[Buyer], bob[Hel	per] start amazon[Se	ell	er]: a(k)	;
alice."rabbits"	\rightarrow	amazon.book	•	k	;
amazon. <i>price</i> (book)	\rightarrow	alice.price	•	k	;
alice.(price/2)	\rightarrow	bob.contrib	•	k	;
if (contrib < 100\$)@b	ob				
bob	\rightarrow	amazon	•	k[ok]	;
amazon. <i>text</i> (book)	\rightarrow	bob.text	•	k	
else					
bob	\rightarrow	amazon	•	k[ko]	
Sender		Receiver	Se	ession	

Compiler



• We provide a compiler to executable code in Jolie.

• The code is guaranteed to be deadlock-free by construction.

• Jolie allows us to reuse executables in different deployments (HTTP, etc.).

• Demo.

Evaluation

- We used Chor for evaluating our approach with representative use cases:
 - authentication protocols (OpenID);
 - E-Commerce;
 - data streaming;
 - service discovery.
- Industrial collaborators:






Conclusions

Conclusions

- A new language for a new paradigm.
- Guarantees deadlock-freedom: suitable for critical systems.
- Fast prototyping of systems.
- A lot of future work to do!

Limitations

- No support for external services yet (e.g., cannot invoke Google search).
- No support for round-trip programming.

Future Directions

Future directions

• How far can we go? Need for more systematic studies.



Future directions

- Exploiting the global view of choreographies in:
 - Fault handling.

• Reversible computing.

• Security.

• Model checking.

Fabrizio \rightarrow audience : Thank you!

- More information at:
 - Chor Website: http://www.chor-lang.org/
 - Jolie Website: http://www.jolie-lang.org/
 - My web page: http://www.fabriziomontesi.com/



