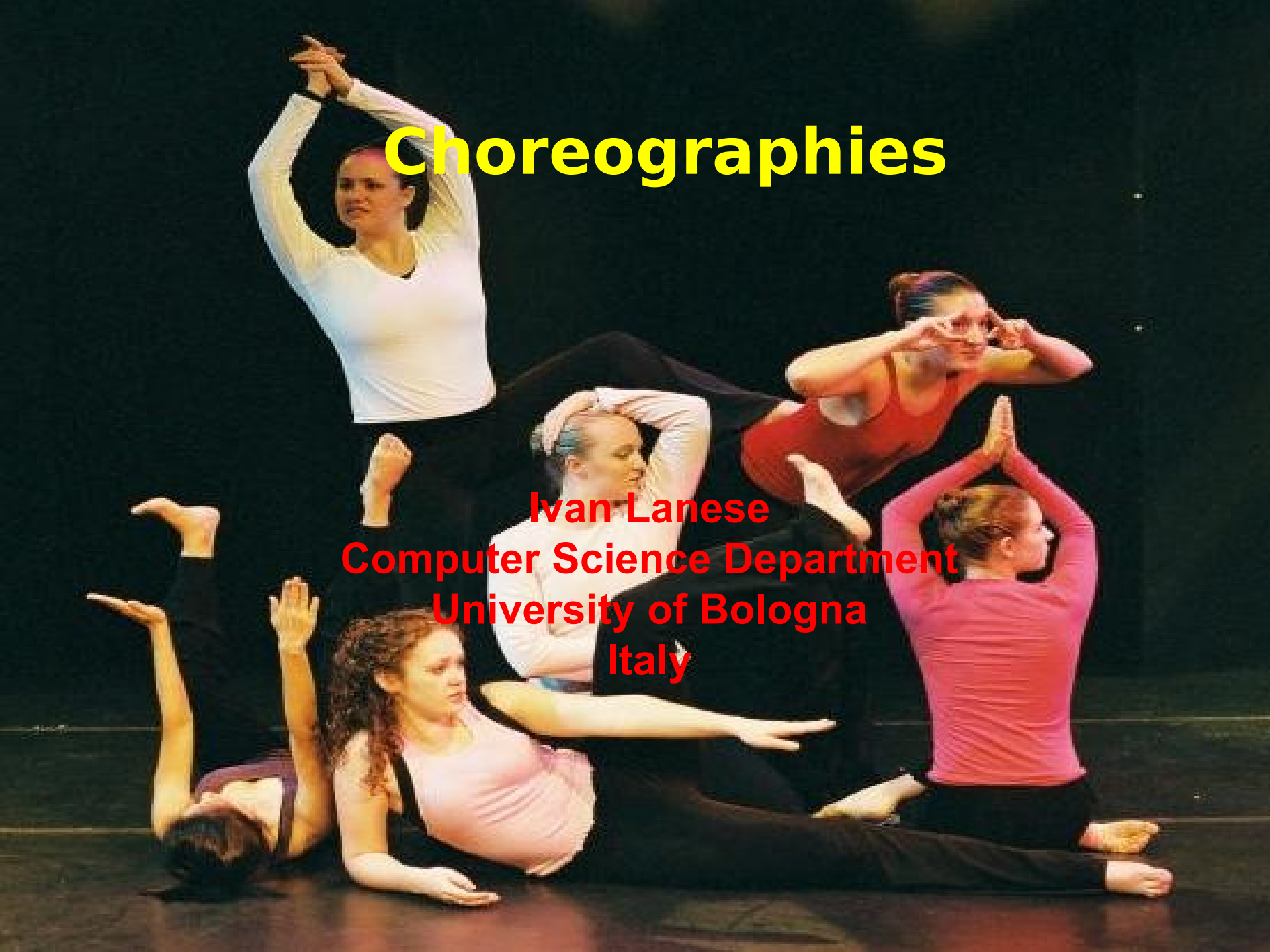

Choreographies

Ivan Lanese
Computer Science Department
University of Bologna
Italy

Service choreography

- A SOC application is usually composed by many services
- At runtime services interact via complex multiparty conversations
  - Client-server is just a (frequent) special case
- A service choreography is the description of the possible conversation patterns from a global point of view
  - Fundamental to describe the behavior of SOC applications
  - Even more if there is no central coordinator
- You have already seen/will see BPMN choreographies
- We will
  - consider a simpler setting
  - be more formal

Choreographies as abstractions

- Our choreographies fully define
  - Which kinds of messages can be exchanged
    - participant A sends a message to participant B on operation o
    - an interaction involves exactly two participants
  - Possible control flow around them
- Our choreographies do not define
  - Local computations
  - Which values are sent
  - Which conditions actually determine choices
- Choreographies can be used to avoid errors related to message passing
  - Communication errors
  - Communication deadlocks
  - Communication races

Choreography vs orchestration

- An orchestration describes the conversation from a centralized point of view
  - Either there is a participant acting as coordinator
  - Or the behavior emerges as the result of the interaction among the different participants
    - Difficult and error-prone
- A choreography describes the conversation from a global point of view
  - No need for a central point of control
  - Difficult to implement

Choreography idioms

- BPMN includes BPMN choreographies
- WS-CDL also supports choreographies as specification
- Choreographies can also be used for
  - Static analysis: choreographies as behavioral types
    - Multiparty session types
  - Dynamic analysis: choreographies as monitors
  - Generating application skeletons to be refined
  - Programming: choreographies as programs
    - Requires more information
    - See the AIOCJ language
      - http://www.cs.unibo.it/projects/jolie/aiocj.html

WS-CDL

- Web Services Choreography Description Language
- Candidate recommendation from W3C in 2005
- XML based
- A declarative language, not executable
- Very verbose
- Participants should implement their services conforming to the choreography
- WS-CDL does not clarify what conforming means
- No more work on it from W3C
- Even one of its developers thinks the standard is dead

Choreographies in practice

- Industrial practice only considers choreographies as specification (or no choreographies at all)
  - UML sequence diagrams
  - BPMN choreographies
- Academic research mainly considers choreographies for static analysis
  - Started with
    Kohei Honda, Nobuko Yoshida, Marco Carbone: Multiparty asynchronous session types. POPL 2008: 273-284
- Quite active research topic

Global vs local view

- Choreographies present a global view of a distributed system
  - Highlights the global behavior
  - Good tool for design
- Distributed systems are normally programmed by writing code for each endpoint
- Thus, we want to translate automatically a choreography into a system implementing it
- We derive the (desired or actual) behavior of each role in the system via a projection operation
- The roles by interacting should give rise to a behavior that is conformant to the original choreography

Choreographies, formally

- We want to describe choreographies and their projections formally
- To know exactly what is happening
- To be able to prove theorems about what is happening
- We will select a specific approach among the ones that exist (ideas are similar, technical details differ)
- We will use two process calculi, one for choreographies and one for projected systems
  - We will define their syntax and semantics
  - We will define projections
  - We will discuss some results
- The process calculus for choreographies is simpler and more precise than BPMN choreographies or WS-CDL
- I will not give you full details, just the main intuitions

Syntax of choreographies

C ::=	o: a → b	Interaction
	C;C	Sequential composition
	C|C	Parallel composition
	C+C	Nondeterministic choice
	C*	Iteration (0 or more times)
	1	Skip

This is an abstract language, it could be made more concrete by adding further information (data, actual conditions)

Semantics of choreographies

- Interactions are executed in the order specified by the choreography
- (ask: cl → se ; (answer: se → cl | log: se → logger))* ; end: cl → se ; endl: se → logger
- A client asks something to a server, the server answers and in parallel sends some information to a logger
- This behavior can be repeated 0 or more times
- At the end, the client tells the seller to stop, and the seller tells the logger to stop
- The semantics is formally defined by inference rules
- The semantics produces sequences of interactions, called traces

Syntax of systems

S ::=	$(P)_a$	role
	S|S	parallel composition
P ::=	$\bar{o}@a$	output
	o@a	input
	P;P	sequential composition
	P|P	parallel composition
	P+P	nondeterministic choice
	P*	iteration (0 or more times)
	1	skip

Semantics of systems

- Each role executes on its own
- Outputs must be matched by a corresponding input
  - An output $\overline{o}$@b in role a interacts with an input o@a in role b producing an interaction o:a→b
- Two possible semantics
  - Synchronous semantics: Outputs interact directly with inputs
    - The semantics produces sequences of interactions
  - Asynchronous semantics: Outputs go in the net, and then possibly interact with an input
    - The semantics produces sequences that include message sends and interactions

Back to our aim

- We want to project a given choreography onto roles to get a system exhibiting the corresponding behavior
- We consider the simplest projection possible
- The projection is as follows:

  $proj(o: a \rightarrow b, a) = \overline{o}@b$

  $proj(o: a \rightarrow b, b) = o@a$

  $proj(o: a \rightarrow b, c) = 1$
- An homomorphism on the other operators

  $proj(C;C', a) = proj(C, a);proj(C', a)$
- We want the projection to be conformant to the choreography

What does be conformant means?

What ; means?

Consider the simple choreography

o:a → b; o': c → d

- In the synchronous case the (atomic) interaction between a and b should occur before the (atomic) interaction between c and d
- In the asynchronous case there are different alternatives:
  - Sender: the sending at a should occur before the sending at c
  - Receive: the receive at b should occur before the receive at d
  - Sender-receive: both of the above
  - Disjoint: both the sending at a and the receive at b should occur before both the sending at c and the receive at d

Conditions for a correct projection

- We want conditions ensuring that the projection behaves well
  - They will depend on the chosen semantics
- We have three kinds of “connectedness” conditions
  - For sequential composition
  - For choice
  - For multiple uses of the same operation
  - (No condition for parallel composition)
- These conditions are sufficient for ensuring correctness, but not necessary
  - If the conditions hold we are sure the projection will work as expected
  - If the conditions do not hold, the projection may work as expected but we are not guaranteed

Connectedness for sequence

Ensures the correctness of sequential composition

- o: a → b; o': c → d
- Synchronous: a=c or a=d or b=c or b=d
- Sender: a=c or b=c
- Receiver: b=c or b=d
- Disjoint: b=c

- The conditions can be generalized to ensure the connectedness of arbitrary choreographies
- For each sequential composition and each pair of interactions on opposite sides of the sequential composition, the conditions should hold

Points of choice

Ensures the correctness of the projection of choice

- Synchronous:
  - The same role should occur in each initial transition
  - The roles in the two branches should be the same (if the projection on one branch is 1 then the projection on the other branch needs to be 1 too)
- Asynchronous:
  - The sender should be the same
  - The roles in the two branches should be the same (if the projection on one branch is 1 then the projection on the other branch needs to be 1 too)

Points of choice: example

- Let us consider the following example, which does not satisfy points of choice
  o:a → b + (o':c → d; o'': d → b)
  Either a and b or c and d should interact, but since there is no central control no one can enforce consistent choices
- Let us consider another example that does not satisfy points of choice
  o:a → b + o': a → c
  Both b and c occur in one branch only
  If a sends a message to b, c remains waiting forever
  However, if c is a service that was not involved elsewhere in the choreography, then he will just not be involved, hence this is not a real problem

A note on iteration

Iteration C* can be seen as a combination of sequential composition and choice:

- C can be executed multiple times: C;C should be connected
- At the end of C, there is a choice between C and what follows C*: this choice should be connected

Causality-safety

- Multiple communications on the same operation between the same participants enabled together may interfere
- Output of an interaction may interact with input of another interaction
- We need to ensure that they can never interact
  - either since they are enabled one after the other
  - or since they are in alternative executions

Correctness of the projection

- One can characterize the different forms of relation between choreography and projected systems in a formal way
  - Trace equivalence or bisimilarity
  - E.g., under the synchronous semantics a connected choreography and its projection produces the same set of traces
- The correctness of the projection can be proved

- Formal details for a slightly different calculus in Ivan Lanese, Claudio Guidi, Fabrizio Montesi, Gianluigi Zavattaro: Bridging the Gap between Interaction- and Process-Oriented Choreographies. SEFM 2008: 323-332

Choreographies for system development

- Choreographies are an abstraction of system behavior focusing on communications
- Projection on a role abstracts the behavior of an endpoint
- Projections can be used in multiple ways for system development
- Static checking: checking whether the given code of an endpoint conforms to its specification
- Dynamic checking: checking whether a running endpoint satisfies its specification via monitoring
- Base for refinement
  - Seen as a skeleton for actual code
  - Enriched with missing information to obtain the full code of the endpoints

Refining choreographies

- We can use choreographies as skeletons to write Jolie programs
  - Choices become if-then-else on the sender and input-choices on the receiver
  - Iterations become loops (or multiple executions)
  - Parallel becomes parallel
  - 1 can be expanded to any local computation
    - 1 is a neutral element of sequential and parallel composition (but not of choice), hence we may add 1s
  - We may group two interactions in a request-response

Exercise

- A buyer wants to buy some items from a seller. For each of them, if the price is low he sends the money to the seller and receives the item back. If the price is too high he can decide to buy the item together with an helper. Both the buyer and the helper send half of the needed money to the seller. When the seller receives the needed money he sends back the item to the buyer.
- Write a choreography modeling the interaction above
- Check whether the choreography is connected or not
- If not, try to understand which problems this may cause (if any), and try to change the choreography to fix it (hint: you should add additional communications)
- Project it

Exercise (cont.)

- Take the projection of the choreography from the previous slide and refine it to build a Jolie SOA