Programming Distributed Adaptive Applications
A Choreographic Approach

Ivan Lanese
Computer Science Department
University of Bologna/INRIA
Italy

Joint work with Mila Dalla Preda, Maurizio Gabbrielli, Saverio Giallorenzo and Jacopo Mauro
Map of the talk

- Choreographic programming
- Dynamic updates
- Conclusion
Map of the talk

- Choreographic programming
- Dynamic updates
- Conclusion
Programming distributed applications

- Distributed applications are composed by many nodes
- Each node executes a program, possibly different from the ones executing on the other nodes
- The nodes interact by exchanging messages
  - Main primitives: send and receive of a message
- Distributed applications are everywhere
  - Facebook, google, expedia, …
  - Banking applications, online shops, …
Distributed programming: pitfalls

- Programming distributed applications is difficult and error-prone
- **Deadlocks**: a process waits for a message that will never arrive
- **Orphan messages**: a message is sent to a target not expecting it
- **Races**: two messages are expected in a given order, but may arrive in a different one
- (Same problems also in message-based concurrent programming)
Choreographic programming: aim

- Choreographic programming aims at avoiding these pitfalls
- Pitfalls due to a mismatch between send and receive
- Solution: send and receive cannot be written in isolation, but only composed inside an interaction
- An interaction is a communication between two participants:

  \[ \text{op: alice}(z+5) \rightarrow \text{bob}(x) \]
Choreographic programming: basics

- A single interaction needs to be executed by two participants
  - Not just one as for standard programming language constructs
- A single choreographic program describes the behavior of a whole distributed application
- It is built by composing interactions using standard constructs: sequences, conditionals, cycles,...
- Each construct may describe the behavior of one, two, or more participants
Choreographic syntax

- $C ::= o: s(e) \rightarrow r(x)$
  - $x@r = e$
  - skip
  - $C ; C'$
  - $C | C'$
  - if $b@r \{C\}$ else $\{C'\}$
  - while $b@r \{C\}$

- For multiparty session types addicts
  choreographic programs $\approx$ global types + data + conditions
A sample choreographic program

product_name@client = getInput( "Insert product name" );
quote: client( product_name ) → seller( sel_product );
price@seller = getInput( "Quote " + sel_product )
Choreographic programming: advantages

- Unique description of the whole distributed system ⇒ Clear view of the global behavior
- Sends and receives are paired into interactions ⇒ No deadlocks, orphan messages nor races by construction
How to execute choreographic programs?

- A choreographic program describes the behavior of many participants
- We want to compile the choreographic program generating a local code for each participant
- When executed, the derived local codes should interact as specified in the choreographic program
  - Correctness of the compilation
  - No deadlocks, orphan messages nor races
- We specify compilation as a projection function
The target language

- $P ::= o: \text{send e to r}$
  - $o: \text{receive x from r}$
  - $x = e$
  - $\text{skip}$
  - $P ; P'$
  - $P | P'$
  - if $b \{P\} \text{ else } \{P'\}$
  - while $b \{P\}$

- A distributed program is composed by multiple named participants each executing its own $P$
Projection: basic idea

- An interaction $o: s(e) \rightarrow r(x)$ is projected as
  - $o$: send $e$ to $r$ on $s$
  - $o$: receive $x$ from $s$ on $r$
  - skip on all the other participants

- Assignments $x@r = e$ are executed by the role $r$

- Other constructs are projected homomorphically

- Very simple…

- …but it does not work
Projection: problems

- Actions of different participants are independent
  \[ o_1: s_1(5) \rightarrow r_1(x) ; o_2: s_2(7) \rightarrow r_2(y) \]

- Interaction on \( o_2 \) should happen after interaction on \( o_1 \)
  - No participant can force this

- The behavior of a participant may depend on the decision of another participant
  if \( b@s_1 \) \{ \( o: s_2(5) \rightarrow r(x) \) \} else \{ \( o: s_2(7) \rightarrow r(x) \) \}

- Participant \( s_2 \) should send 5 or 7 according to a local decision of \( s_1 \)
Projection: solutions

- These problems are solved by introducing auxiliary communications
- Automatically generated by the projection
- In a sequence C;C’, the participants in C notify the participants in C’ when C is finished and C’ can start
- In a conditional if b@r {C} else {C’} participant r notifies participants in C and C’ of the outcome of the choice
From abstract theory to the real world

- We implemented AIOCS as an instance of this theory
- Choreographic programs can be written in Eclipse
- Projection generates code in the Jolie language
  - Jolie is a language for programming distributed systems
  - Follows the microservice paradigm
  - Microservices are independent entities communicating via message passing over well-defined interfaces
- The microservices can be executed, obtaining the desired behavior
Integration with Legacy Software

- In the real world applications are not built from scratch
- One uses external libraries, services (e.g., google maps), existing applications…
- We want to integrate them into our framework
- We allow invocations to external software inside expressions
- We allow the use of various technologies (sockets, bluetooth,...) and protocols (SOAP, HTML,...)
- We declare at the beginning which functions we need, where they are, and how to interact with them
Map of the talk

- Choreographic programming
- Dynamic updates
- Conclusion
Dynamic updates

- We want to change the code of running applications, by integrating new pieces of code coming from outside.
- Many possible uses:
  - Deal with emergency behavior
  - Deal with changing business rules or environment conditions
  - Specialise the application to user preferences
- The external pieces of code are called updates.
- External updates may arrive at any time, and disappear at any time.
- Only currently available updates may be applied.
Our approach, syntactically

- Pair a running application with a set of updates
  - Each update is a choreographic program
  - The set of updates may change at any time
- At the choreographic level, the update may replace a part of the application
  - Which part?
- Extend choreographic programs with scopes
  - scope @r {C}
  - Normally executes C, but executes an update instead if a suitable one is available
  - Participant r is in charge of managing the update procedure
Our approach, graphically
Our approach, graphically
Our approach, graphically
Dynamic updates: challenges

• All the participants should agree on
  – whether to update a scope or not
  – in case, which update to apply

• All the participants need to retrieve (their part of) the update
  – Not easy, since updates may disappear

• No participant should start executing the code inside a scope that needs to be updated
Dynamic updates: our approach

- For each scope a single participant coordinates its execution
  - Decides whether to update it or not, and which update to apply
  - Gets the update, and sends to the other participants their part
- The other participants wait for the decision before executing the scope
- This behavior can be formalized by extending our target language, and actually programmed in practice
Results

- A choreographic program and its projection behave the same
  - They have the same set of traces (up to auxiliary actions)
  - Under all possible, dynamically changing, sets of updates

- The projected application is deadlock free, orphan message free and race free by construction

- These results are strong given that we are considering an application which is
  - distributed
  - updatable
From updates to adaptation rules

- Nondeterministic application of updates is not practical
- We use adaptation rules composed by an update and its applicability condition
  - May refer to environment variables, scope properties, and local variables of the coordinator of the update
- A rule can be applied only if its applicability condition is satisfied
- We add properties to scopes
Back to the real world

- Adaptation managed by a dedicated middleware composed of microservices
- Rules are managed by dedicated microservices called adaptation servers
- A microservice called adaptation manager coordinates the adaptation procedure
- A microservice stores environment information
Map of the talk

- Choreographic programming
- Dynamic updates
- Conclusion
Conclusion

- A choreographic approach to program distributed applications
- The derived distributed application follows the behavior defined by the choreographic program
- We ensure deadlock freedom, orphan message freedom and race freedom in a challenging setting
- We allow integration of external functionalities
- We allow dynamic update, preserving the correctness guarantees
- We instantiated the theoretical framework into a real programming language
Future work

- Is choreographic programming practical?
  - We need to allow asynchronous communication
  - We need to improve performance
    - Drop redundant communications
  - We need error handling

- Only a serious case study can tell
  - Not so easy to find a suitable case study for adaptation
For a general introduction to choreographies and related topics:

AIOCJ:

Theory:

Comprehensive journal paper:

Jolie website: http://www.jolie-lang.org/
End of talk

Thanks!

Questions?