Ensuring Faultless Communication Behaviour in a Commercial Cloud Application

Ross Horne, Rustem A. Kamun and Timur Umarov

Faculty of Information Technology, Kazakh-British Technical University, Almaty, Kazakhstan
ross.horne@gmail.com r.kamun@gmail.com t.umarov@kbtu.kz

Abstract. The goal of this work is to ensure that processes that integrate several services in a Cloud correctly interact according to a specification of their communication behaviour. To accomplish this goal, we employ session types to analyse the global and local communication patterns. A session type represents a "formal blueprint" of how users and services should interact with the Cloud at an appropriate level of abstraction for specifying message flows.
This work confirms the feasibility of applying session types to business protocols used by an e-commerce Cloud provider. The protocols are developed in SessionJ, an extension of Java implementing session-based programming. Furthermore, we highlight how our approach can be used to intergrate services across multiple Cloud providers, each of whom must correctly cooperate.

1 Introduction

Cloud providers typically offer a portfolio of services, where access and billing for all services are integrated in a single distributed system. Services are then made available on demand to anyone with a credit card, eliminating the up front commitment of users [1]. Furthermore, there is a drive for services to be integrated, not only within a Cloud, but also between multiple Cloud providers [14].

Protocols that integrate heterogeneous services with a single point of access and billing strategy can become complex. Thus we require an appropriate level of abstraction to specify and implement such protocols. Further to the complexity, the protocols are a critical component of the business strategy of a Cloud provider. Failure of the protocols could result in divergent behaviour that jeopardises services, leading to loss of customers or even legal disputes. These risks can be limited by using techniques that statically prove that protocols are correct and dynamically check that protocols are not violated at runtime.

It is challenging to manage service interactions that go beyond simple sequences of requests and responses or involve large numbers of participants. One technique for managing protocols between multiple services is to specify the protocol using a choreography. A choreography specifies a global view of the interactions between participating services. However, by itsself, a choreography does not determine how the global view can be executed.

The challenge of controlling interactions of participants motivated the design of Web Services Choreography Description Language (WS-CDL) [8]. The WS-CDL working group identified critical issues [3] including:

1. the need for tools to validate conformance to choreography specifications to ensure correct cooperation between Web Services;
2. design time verification of choreographies to guarantee correctness of properties such as deadlock and livelock, as well as the conformance of the behaviour of participants.

The aforementioned challenges can be tackled by adopting a solid foundational model, such as session types [8,7]. Successful approaches related to session types include: SessionJ [12], Scribble [11] and Session C [13] due to Honda and Yoshida; Sing# [4] that extends Spec# with choreographies; and UBF(B) [2] for Erlang.

In this paper, we present a case study where the interaction of process that integrate services in a commercial Cloud provider[1] are controlled using session types. Session types ensure communication safety by verifying that session implementations of each participant (the users, the services and the Cloud), conform to the specified protocols. In our case study, we use SessionJ, an extension of Java supporting sessions, to specify protocols used by the Cloud provider that involve iteration and higher order communication.

In Section 2 we provide an overview of SessionJ. In Section 3, we explain and refine a protocol used by a Cloud provider and implemented using SessionJ. Finally, in Section 4, we suggest that session types can be used in the design of reliable inter-Cloud protocols, following the techniques employed in this work.

2 Methodology for Verifying Protocols in SessionJ

We chose SessionJ for the core of our application, since Java was already used for several services. Furthermore, the language has a concise syntax and an active community.

We briefly outline how SessionJ is employed to correctly implement protocols. Firstly, the global protocol is specified using a global calculus similar to sequence diagrams. Secondly, the global calculus is projected to sessions types, which specify the protocol for each participant. Thirdly, the session is implemented using operations on session sockets. The correctness of the global protocol can be verified by proving that the implementation of each session conforms to the corresponding session type.

Protocol Specification. The body of a protocol defines a *session type*, according to the grammar in Figure 1. The session type specifies the actions that the participant in a session should perform. In SessionJ, the behaviour of an implementation of a session is monitored by the associated protocol, as enforced by the SessionJ compiler and runtime. The constructs in Figure 1 can describe a diverse range of complex interactions, including message passing, branching and iteration. Each session type construct has its dual construct, because a typical requirement is that two parties implement compatible protocols such that the specification of one party is dual to another party.

Higher Order Communication. SessionJ allows message types to themselves be session types. This is called higher-order communication and is supported by using subtyping [15]. Consider the dual constructs $!\langle?(\text{int})\rangle$ and $?(?(\text{int}))$. These types specify

[1] V3na Cloud Platform. AlmaCloud Ltd., Kazakhstan. http://v3na.com

L_1, L_2 label

p protocol name

$M ::= \mathit{Datatype} \mid T$ message

$S ::= p\,\{T\}$ protocol

T	$::= T\,.\,T$	Sequencing
	$\mid \texttt{begin}$	Session initiation
	$\mid !\langle M \rangle$	Message send
	$\mid ?(M)$	Message receive
	$\mid \{L_1 : T_1, \ldots, L_n : T_n\}$	Session branching
	$\mid [T]*$	Session iteration
	$\mid \texttt{rec}\,L\,[T]$	Session recursion scope
	$\mid \#L$	Recursive jump
	$\mid @p$	Protocol reference

Fig. 1: SessionJ protocol specification using session types (T).

sessions that expect to respectively send and receive a session of type ?(int). Higher order communication is often referred to a session delegation. Figure 2 shows a basic delegation scenario.

In Figure 2, the left diagram represents the session configuration before the delegation is performed: the user is engaged in a session s of type $!\langle \text{int} \rangle$ with the Cloud, while the Cloud is also involved in a session s' with a service of type $!\langle ?(\text{int}) \rangle$. So, instead of accepting the integer from the user, Cloud delegates his role in s to the service. The diagram on the right of Figure 2 represents the session configuration after the delegation has been performed: the user now directly interacts with the service for the session s. The delegation action corresponds to a higher-order send type for the session s' between the Cloud and the service.

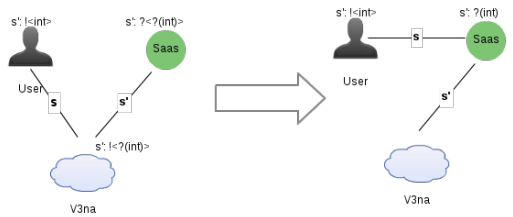

Fig. 2: Session delegation

Protocol Implementation using SessionJ. Session sockets represent the participants of a session connection. Each socket implements the session code according to the specified session type. In SessionJ session sockets extend the abstract *SJSocket* class. The session is implemented within a session-try scope using a vocabulary of session operations.

3 Case Study: Protocols for a Cloud

Our case study is an e-commerce Cloud application, V3na, that provides integrated Software as a Service solutions for business. V3na provides a central access point to several services, including document storage, document flow, and customer relations management. The central component of the e-commerce Cloud is responsible for seamless integration and maintenance of the services that a user subscribes to, while managing user accounts and billing.

A typical scenario is when a user requires the document storage service. The user will first subscribe for the service either by registering to be billed or by entering a trial period. When the user has subscribed for the document service and they attempt to access their documents, requests to the API of the document store are delegated, by V3na, to the relevant document server for a lease period. After delegation, the client interacts directly with the API of the document store until the lease expires.

A major challenge was to automate the process of service integration as a reliable service. In particular, V3na implements the following problems that can be addressed using sessions types:

- A customer can connect to a service for a trial period;
- V3na provides one entry point to all services a user subscribes to;
- A subscription may be extended or frozen;
- Billing and payment for use of a service can be managed.

In this section, we illustrate two refinements of the first scenario above.

3.1 Scenario 1: Forwarding and Branching

To begin, we specify a simple business protocol for connecting to a service. The protocol is informally specified as follows:

Protocol 1.1: User

```
protocol p_uv {
  begin.
  !<JSONMsg>.
  ?{
    OK: ?(JSONMsg),
    FAIL:
  }
}
```

Protocol 1.2: Cloud

```
protocol p_vu {
  begin.?(JSONMsg).!{
    OK: !<JSONMsg>,
    FAIL:
  }
}
protocol p_vs {
  begin.!<JSONMsg>.
        ?(JSONMsg)
}
```

Protocol 1.3: Service

```
protocol p_sv {
  begin.
  ?(JSONMsg).!<JSONMsg>
}
```

Fig. 3: Protocol specifications for Scenario 1

1. The user begins a request session with Cloud service (V3na) and sends the request "connect to service" as JSON-encoded message.
2. V3na selects either:
   (a) FAIL, if the user has no active session (not signed in).

(b) OK, if the user has logged in and the request data has passed validation steps.

3. If OK is selected, then, instead of responding immediately to the user, the Cloud initiates a new session with the service. In the new session the Cloud forwards the JSON message from the client to the service and receives a response from the server. The session between the Cloud and the service terminates. Finally, the original session resumes and the Cloud forwards the response from the service to the Client. From the perspective of the user it appears that the Cloud responded directly.

In Figure 3 we provide the protocol specifications for each participant (User, Cloud or Service). The protocols between the user and the Cloud and between the Cloud and the service are dual, i.e. the specification of interaction from one perspective is opposite to the other perspective. SessionJ employs the `outbranch` and `inbranch` operations to implement the branching behaviour.

3.2 Scenario 2: Session Delegation and Iteration

We present a refined example that demonstrates iteration and session delegation. To avoid becoming a bottleneck, the core processes of the Cloud should delegate the session to a service as soon as the user is authenticated for the service.

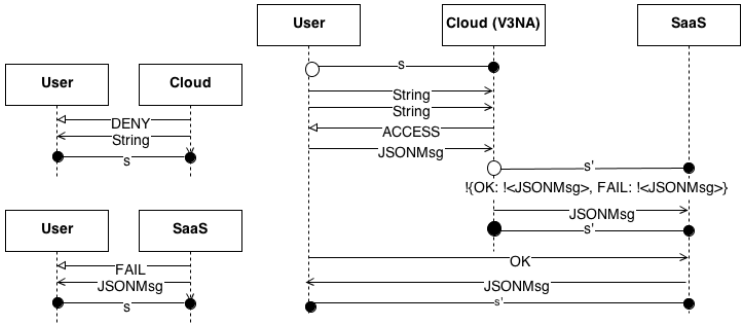

Fig. 4: Sequence diagram of interactions for Scenario 2

Figure 4 depicts two related sessions s and s'. Session s begins with interactions between the user and the Cloud; but, after authentication, s' delegates the rest of session s from the Cloud to the service. Session s is completed by an exchange between the user and the service directly. We informally describe the global protocol in more detail:

1. The user begins a request session (session s in Fig. 4) with the Cloud (V3na).
2. The user logs in by providing the Cloud with a user name and password.
3. V3na receives the user credentials and verifies them: If the user is not authenticated and still has tries go back to step 2, otherwise continue.
4. If the user is not allowed to access the Cloud, the interaction between the user and the Cloud continues on the DENY-branch, otherwise on the ACCESS-branch.

5. In the case of the ACCESS branch, the user sends his connection request in a JSON message to the Cloud. The Cloud creates a new session with the service (session s' in Fig. 4). The new session delegates the remaining session with the user to the service, and also forwards relevant user request details to the service. Session s' is then terminated.
6. The service continues session s, but now interactions are between the user and the service. The service either responds to the user with OK or FAIL. In either case, the user receives the reason and status of his request directly from the service in a JSON message. Finally, session s is terminated.

Protocol 2.1: User

```
protocol p_uv {
  begin.?[!<String>.!<String> ]*.
  ?{
   ACCESS: !<JSONMsg>.
      ?{
        OK: ?(JSONMsg),
        FAIL: ?(JSONMsg)
      },
   DENY: ?(String)
  }
}
```

Protocol 2.2: Cloud

```
protocol p_vu {
  begin.
   ![ ?(String).?(String) ]*. // login
   !{
     ACCESS: ?(JSONMsg).
        !{
          OK: !<JSONMsg>,
          FAIL: !<JSONMsg>
        },
     DENY: !<String>
   }
}
```

Fig. 5: User-Cloud interaction protocol specifications for Scenario 2

In Figure 5, the protocol between the user and the cloud provider appear to interact perfectly. The iterative login step works as expected, so either the ACCESS or DENY branch will be selected. If the ACCESS branch is selected, then, as expected, a JSON message is sent from the user to the Cloud. However, instead of the Cloud choosing OK or FAIL directly, the session (s') in Figure 6 is triggered.

Protocol 2.3: Cloud

```
protocol p_vs {
  begin.
  !<!{
     OK: !<JSONMsg>,
     FAIL: !<JSONMsg>
  }>.
  !<JSONMsg>
}
```

Protocol 2.4: Service

```
protocol p_sv {
  begin.
  ?(!{
     OK: !<JSONMsg>,
     FAIL: !<JSONMsg>
  }).
  ?(JSONMsg)
}
```

Fig. 6: Cloud-Service interaction protocol specifications for Scenario 2

The session in Figure 6 delegates the part of the session where either OK or FAIL is selected to the service. The delegation is enabled by a higher order session type, where

a socket of session type $!\{\texttt{OK}: !\langle \mathit{JSONMessage}\rangle, \texttt{FAIL}: !\langle \mathit{JSONMessage}\rangle\}$ is sent from the Cloud in protocol p_vs and received by the service in protocol p_sv. Following, the delegation, a JSON message is sent from the Cloud to the service containing the relevant user request details.

Once the delegation has taken place, the service is able to complete the session that was begun by the Cloud. The service can negotiate directly with the client and either choose the OK branch or the FAIL branch, followed by sending the appropriate JSON message. The simple choice between an OK and a FAIL message could be replaced by a more complex iterated session between the user and the service.

4 Future Work: Inter-Cloud Protocols and Session Types

For customers of Cloud providers, there are considerable benefits when service can be hosted on more than one Cloud provider [6,5,1]. If data is replicated across multiple Cloud providers, customers can avoid becoming locked in to one provider. Thus customers are less exposed to risks such as fluctuations in prices and quality of service at a single provider. If a Cloud provider goes out of business, then customers entirely dependent on one Cloud provider are critically exposed.

Several visions have been proposed for inter-Cloud protocols [10,9]. In [14], they identify the main components of general inter-Cloud architecture: a *Cloud coordinator*, for exposing Cloud services; a *Cloud broker*, for mediating between service consumers and Cloud coordinators; and a *Cloud exchange*, for collecting consumers' demands and locating Cloud providers. Based on our experience in this work, we suggest that multi-party session types [13] are appropriate for specifying and correctly implementing protocols between Cloud providers and Cloud integrators.

5 Conclusion

This case study demonstrates the ability of session types to control interaction patterns between communicating processes in a Cloud. Participants are statically type-checked at compile time and dynamically monitored at run-time to ensure that components critical to a Cloud provider communicate correctly. The high level of abstraction of session types, implemented in the SessionJ language, enabled effortless translation of business scenarios into protocols. We were able to refine our protocol from Scenario 1 to Scenario 2, due to support of higher-order message passing (session delegation). Further to scenarios presented here, our case study covered payment and wallet recharging transactions, where we discovered the benefits of combining session delegation and threading provided by SessionJ. Our experience shows that the session-programming approach is suited to correctly implementing inter-Cloud protocols, which is our future objective.

Acknowlegements. We thank the anonymous reviewers for their clear and constructive comments. We are grateful to Ramesh Kini for his support for this project.

References

1. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.
2. Joe Armstrong. Getting Erlang to talk to the outside world. In *Proceedings of the 2002 ACM SIGPLAN workshop on Erlang*, pages 64–72. ACM, 2002.
3. Alistair Barros, Marlon Dumas, and Phillipa Oaks. A critical overview of the web services choreography description language. *BPTrends Newsletter*, 3:1–24, 2005.
4. Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. *ACM SIGPLAN Notices*, 47(1):191–202, 2012.
5. David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. Blueprint for the intercloud-protocols and formats for cloud computing interoperability. In *ICIW'09.*, pages 328–336. IEEE, 2009.
6. Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.
7. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for Web services. In *Programming Languages and Systems*, pages 2–17. Springer, 2007.
8. Marco Carbone, Kohei Honda, Nobuko Yoshida, Robin Milner, Gary Brown, and Steve Ross-Talbot. A theoretical basis of communication-centred concurrent programming. WS-CDL working report, W3C, 2006.
9. E. Cavalcante, F. Lopes, T. Batista, N. Cacho, F.C. Delicato, and P.F. Pires. Cloud integrator: Building value-added services on the Cloud. In *Network Cloud Computing and Applications (NCCA'11)*, pages 135–142, 2011.
10. Vij D. Bernstein. Using xmpp as a transport in intercloud protocols. In *In Proceedings of CloudComp 2010, the 2nd International Conference on Cloud Computing*, pages 973–982. CiteSeer.
11. Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *Distributed Computing and Internet Technology*, pages 55–75. Springer, 2011.
12. Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *ECOOP 2008*, pages 516–541. Springer, 2008.
13. Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty Session C: Safe parallel programming with message optimisation. In *Objects, Models, Components, Patterns*, pages 202–218. Springer, 2012.
14. Rajiv Ranjan Rajkumar Buyya and Rodrigo N. Calheiros. InterCloud: Utility-oriented federation of Cloud Computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.
15. Malcolm Hole Simon Gay. Subtyping for session types in the pi calculus. *Journal Acta Informatica*, 42(2-3):191–225, 2005.