Session Types for Software Engineering

Dimitrios Kouzapas[1], Raymond Hu[2], Rumyana Neykova[2], Simon J. Gay[1], Tihana Galinac Grbac[4], Ornela Dardha[1], Simon Fowler[3], Nicholas Ng[2], Sam Lindley[3], J. Garrett Morris[3], Philip Wadler[3], and Nobuko Yoshida[2]

[1] University of Glasgow, UK
[2] Imperial College London, UK
[3] University of Edinburgh, UK
[4] University of Rijeka, Croatia

Abstract. Session types are formal descriptions of communication protocols, which can be incorporated into programming languages and tools so that the technology of typechecking can be used to verify the dynamic structure of communication as well as the static structure of data. During recent years, research on session types has moved from theoretical studies towards applications in practical programming; the field is maturing to the point where session types can be integrated into software engineering methodologies. However, some of the relevant literature is not sufficiently accessible to a wider academic and industrial audience; more systematic technology transfer is required. The present paper provides an entry point to a substantial repository of practical use-cases, drawn from a range of application domains, for programming languages and tools based on session types. It aims to serve as an accessible introduction to session types as a practical foundation for the development of communication-oriented software, and to encourage the adoption of session-type-based tools.

1 Introduction

The session types discipline emerged during the last two decades, as a framework for enforcing a number of *good* and desirable properties in concurrent programs. Since their original inception in the the seminal paper by Honda et. al. [3], there has been a broad research effort to study the properties of session types and to apply the principles of session types to different computational models and programming paradigms. At the moment, there is a research effort converging towards the the development of tools and technologies that use session types as part of the software development process. It is therefore important to initiate a process that will integrate the principles of session types in the broader context of software and system development.

The research process on session types is mainly driven be the need to demonstrate that the principles of session types can capture arbitrary patterns of concurrency, i.e. concurrency patterns that are not artificial and restricted to a narrow set of cases. A typical research paper on session types identifies a problem in terms of a real world usecase and proposes a session typed framework,

expressed in a strict mathematical form, that applies the principles of session types for the solution of the problem. The development of session typed tools and technologies is a consequence of the above research process; tools are developed following following the theory and are used to solve real world usecases.

The above standard of presenting results on session types, although successful within the session types community, it presents difficulties in the knowledge exchange process between session types experts and someone, not necessarily academic, interested in session types. The difficulties are summarised in the points below:

1. There is a huge volume of research information on session types. Interested individuals will need to spend effort to find and study a diverse universe of literature.
2. There is no uniform approach and/or terminology in the presentation of research results. Due to the diversity of session types, there are may terms that derive from different disciplines and are extended to refer to the same session type notions. This may lead to confusion between less expert readers. An example is the use of the term typestate, the term session type and the term protocol in the context of of the OO programming.
3. Session type results are often presented in isolation with respect to other results. A researcher will have to perform research following a series of publications and technical reports in order to have a broader understanding of the subject.
4. The majority of the results are tightly coupled with a high level of formal technicality making the comprehension of session types by people without a formal background even more difficult.

A conclusion on the above discussion is that a wider audience would need to study a large series of partial results and at the same time filter out a huge amount of unnecessary technical details in order to achieve a satisfying level of understanding session types. The problems identified above, are not just difficulties to a newcomer, they might also discourage his or her efforts in understanding session types.

Towards the direction of integrating session types with the software development process, this paper identifies the problem of accessibility of session types to a wider academic and industrial audience. Thus, the motivation is driven by: i) the need to demonstrate session types in a comprehensive way and in terms that are easier to understand by less experts in the domain of session types; and ii) the need enable a process that will integrate session types in a broader context in computing.

A way to meet the above objectives is to identify a common ground that can be used for knowledge exchange between researchers. This paper identifies as common means of communication the demonstration of practical scenarios from a broad area of application in computer science. Usecases can bridge the communication gap between researcher and can help in a wider promotion of the principles of session types.

Furthermore, the current research directions on session types are pushing towards a dialectic between session types and disciplines like software engineering, and system design and implementation. This is justified by the research experience gained by working on the project "From Data Types to Session Types: A Basis for Concurrency and Distribution" (ABCD for short) [2]. Part of the requirements of the ABCD project is to compile a set of usecases for session types that can be used for current and future research on the design and development of frameworks and technologies.

More concretely, the aim of the paper is to:

1. Describe in common language and develop common terminology for mathematical terms, currently used in the theory of session types. The terms are then used to describe the practical usecases that are exhibited in this paper.
2. Demonstrate the methods that are currently used to present, analyse, and express an application in session types terms. The method includes the Scribble [4] protocol description language and tool chain and different diagrammatic languages.
3. Demonstrate the robustness, functionality, and overall applicability of session types through a diverse overview of usecases that cover:
   - Different domains that exhibit different computational needs.
   - Different interpretations of session types depending on the underlying programming paradigm used to implement a usecase. This would help realise the perspective of applying session types to further programming paradigms.
     Communication based programming, events, typestate-OO, functional programming
   - Current tools and technologies that use session types in the software development process.
4. Demonstrate software design and development techniques that are derived from session types principles. Engineers and developers should be able to appreciate the usage of session types in the software development process.
5. Share the experience in using session types for the implementation of different usecase scenarios. This is the first paper to exhibit diverse practical results and can be used as a reference point for future work.

Overall this paper can be used by experts and non experts to: i) exchange knowledge on the overall discipline of session types, and ii) to enable future work on the application of session types.

1.1 Overview of the paper

The paper begins in Section 2 with the definition of the basic terms and notions that are used throughout the paper for the exhibition of the usecase results.

The usecases that are presented in the paper are compiled by the researchers working on the ABCD project and can be found in the ABCD online repository [1]. In Section **??** there is a summary presentation of the usecases in the

online repository that gives an overview for each usecase and discusses its design and implementation specifics as well as the features that are being demonstrated.

In Section **??** there is the selection of two usecases from the online repository in order to demonstrate some of the basic notions of the application of session types. The first example is an example of an online book-store, where two buyers want to share the expenses to buy a book. The online book-store is consider in the bibliography as the standard example for the presentation session types. It first demonstrates the basic send receive operations between multiple entities in a concurrent protocol. A second feature is the fact that it demonstrate in a comprehensive way the interaction logic between the entities. The implementation of the book-store usecase using different technologies gives a further insight to the applicability of session types. The second usecase presented in full is the design and implementation of the Simple Mail Transfer Protocol [**?**] (SMTP), as a way to demonstrate how session types cope with real network protocols. SMTP is a protocol based on states. The increased complexity of states makes the design and implementation of the SMTP a demanding procedure. The paper claims that the use of session types can reduce the demands of the implementation of SMTP.

Section **??** makes a summary of the software engineering practices that are identified... put more

2 Preliminaries

In this section the basic notions and terms that are used throughout the paper, are defined and discussed.

2.1 Session Types: Notions and Terminology

Typically, a type is broadly understood as a meta-information concept, used describe a class of a data. Structures of types are used to describe data structures. When types are structured as input/output requirements they are used to describe functionality. The combination of functionality and data structures give rise to the concepts of classes and interfaces.A session type extends the notion of a type to capture the communication behaviour in concurrent systems. *A session type is a type structure that describes a communication behaviour in terms of a series of send and receive interactions between a set of concurrent entities.*

A session type may also be seen as a formal *specification* of a communication *protocol* among concurrent *roles*. The syntax for session types is defined on *send* and *receive*, and *select* and *branch* operators. Sending and receiving data gives rise to the notion of *message passing*. In a session type messages are described via their types. The case where a session typed value is passed as a message is called *session delegation*. The interaction of select and branch defines the *choice* interaction. Choice requires for a role to use a value called *label* to select a communication behaviour out of a set of behaviours, called *branch*, offered by another role.

Session types assume a *global* protocol that describes the communication interaction of all the roles inside a concurrent system. The perspective of a global protocol through a single role is called *local* protocol. The local protocol describes the communication interaction of a single role with all other roles in the system.

The relation between a global protocol and the local protocols of its roles, is expressed through the *projection* procedure; given the global protocol and a role, projection returns as a local protocol only the interactions of the global protocol that are concerned with the role.

The reverse procedure from projection is called *synthesis*, where a set of local protocols are composed together in a global protocol.

Below the term system is used to describe the whole concurrent system and the term program is used for communication module inside the system.

Typically, a protocol is first expressed globally and then projected locally on the protocol roles. Local specifications are then implemented by the corresponding programs. The term *session fidelity* is used to express the conformance of a program, to the local protocol of the corresponding role. Session fidelity can be checked mechanically, through techniques like type-checking and monitoring.

If all the roles of a protocol are implemented by a system and session fidelity is ensured, then a number of communication properties can be guaranteed for the communication behaviour of the system:

- Every execution state of the system has *safe* communication behaviour:
  - *Communication operator matching*: Every send (resp., select) operation has a corresponding receive (resp., branch) operation.
  - *Message type matching*: The type of every message being send matches the type of the message expected to be receive.
  - *Deadlock-freedom*: Consequently, every message send will be eventually received.
- Every state of the protocol will *progress* to a safe state.

2.2 Session Types as Protocols: The Scribble Tool-chain

This section presents the Scribble [4] tool-chain as the core tool for the application of session types. This section also uses Scribble to clarify the terminology developed in the previous section.

The basic module of the Scribble tool-chain is the Scribble protocol description language, which is a syntax that expresses session type specifications. The design of the Scribble language draws directly from the principles of session types.

In Figure 1, uses the Scribble protocol for the Bookstore usecase in the online repository [1], to present the Scribble syntax. Line 1 defines the of global protocol `Bookstore` between roles `Buyer1`, `Buyer2` and `Seller`. Line 2 gives an example of the simplest endpoint-to-endpoint message passing interaction, where a message with label `book` and type `title` (notation `book(title)`) is send from (keyword `from`)

```
1  global protocol Bookstore(role Buyer1, role Buyer2, role Seller) {
2    book(title) from Buyer1 to Seller;
3    book(price) from Seller to Buyer1;
4    quote(contribution) from Buyer1 to Buyer2;
5    choice at Buyer2 {
6      agree() fom Buyer2 to Buyer1, Seller;
7      transfer(money) from Buyer1 to Seller;
8      transfer(money) from Buyer2 to Seller;
9    } or {
10     quit() from Buyer2 to Buyer1, Seller;
11   }
12 }
```

Fig. 1. Scribble: Global protocol for the Bookstore usecase

```
1 local protocol Bookstore at Buyer1(role Buyer1, role Buyer2, role Seller) {
2   book(title) to Seller;
3   book(price) from Seller;
4   quote(contribution) to Buyer2;
5   choice at Buyer2 { agree() fom Buyer2; transfer(money) to Seller; }
6   or               { quit() from Buyer2; }
7 }
```

Fig. 2. Local Protocol for Role Buyer1

role `Buyer1` to (keyword `to`) role `Seller`. In Scribble messages are described as a type structure consisted of a label and a list of message types.

Line 5 of the protocol clarifies the choice interaction, where role `Buyer2` selects a choice from two possible outcomes, expressed with the labels `agree` or `quit`. In both cases roles `Buyer1` and `Seller` offer alternative branches.

describe the recursion construct

2.3 Domain Classification of Usecases

The paper presents a diversity of usecase scenarios of session types from different application domains, in order to demonstrate the fact that session types can capture a broad area of communication specifications. A taxonomy of domains are presented below with the main characteristic that each domain is using technologies from the previous domains in the taxonomy. This taxonomy also implies a stratification of the application of session types in different computation layers.

1. *Network Application/Business Logic.* Session types can be used to develop protocols for applications that run inside a network. A protocol given in a session type structure, apart from the specification of the communication of the application, will reveal a kind of business logic for the application.
2. *Network Protocols.* Session types can be used to describe standard and non-standard network protocols. Typically a standard network protocol should

conform to an informal RFC (request for comments specification. Session types can present a network protocol formally its manipulation easier by both engineers and machines. Non-standard network protocols can also be developed.

3. *Systems/Applications*. A session type may be used to describe the communication specifics of an application that uses multiple resources inside a computing machine.
4. *Operating System*. Another domain where session types can be applied to is the description of the communication specifics of operating system algorithms and routines, that co-ordinate the usage of hardware resources.
5. *Data Structures and Algorithms*. The above layers are using data structures and algorithms. Session types can express the communication concurrent algorithms are using. Furthermore, session types can express the interaction with data structures.
6. *Hardware*. Hardware mechanisms complete the stratification of domains. The communication of hardware modules may also be expressed using session types.
7. *Security*. Session types can also find applications in the security domain, which is a domain that supports all other domains in the above list.

2.4 Usecase Classification on Technologies used

Just presentation and references

1. Session Java - standard binary session types library with compiler support
2. Eventful Session Java - event driven parafigm
3. Multiparty Session C: a multiparty session type library for MPI with compiler support
4. Monitoring: Session fidelity at runtime. Implemented as a python library - no need for support
5. The actors paradigm
   - Implemented in python using the monitor module
   - Erlang
6. Pabble - MPI: parametric Scribble used to express parametric algorithms and topologies.
7. Mungo and StMungo: based on the OO paradigm a tool that integrates session types and the typestate theory in Java. StMungo transforms Scribble into Mungo specifications.
8. GV: A functional programming implementation of binary session types.
9. LINKS: Functional programming based on the linear logic interpretation of session types.
10. SILL: Functional programming based on the linear dual intuitionistic interpretation of session types.
11. A state pattern implementation as a library. Uses Scribble to automatically create an API for state pattern programming and uses runtime checks to cope with linearity requirements.

2.5 Fine/coarse grain representation

this should go as future work

Concurrency patterns e.g. network topology, client/server, race condition, etc. (maybe I am writing nonsense here)

3 Usecases

4 Session types and Software Engineering

- Session types protocols. Structures that describe a behaviour on the communication interface.
- Top down design: The global type to local types via projection procedure as
  - a way for engineers to design
  - a way to co-ordinate software developers.
- Bottom up design: The inference/synthesis procedure as a way for more agile independent teams to co-ordinate and exchange the behaviour of interfaces as session type specifications.
- Testing: Session typed modules can be used to test a software in the development phase.
- Reverse engineering: inference/synthesis
- The relation of session types with diagrammatic languages
  - BPMN [**?**]
  - global graphs[**?**]
  - petri nets[**?**]
  - state machines [**?**]

References

1. ABCD: Session Types Usecase Repository. https://github.com/epsrc-abcd/session-types-use-cases.
2. From Data Types to Session Types: A Basis for Concurrency and Distribution. http://groups.inf.ed.ac.uk/abcd/.
3. Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
4. Scribble Project homepage. www.scribble.org.

	Usecase	Domain	Technologies/Tools	Description
1.	Book Store	Business Application	Mungo, Session Java	Interaction Logic
2.	Hyper-Text Transfer Protocol	Network Protocol		Request/Response protocol
3.	Simple Mail Transfer Protocol	Network Protocol	Mungo and StMungo, Session Java, LINKS	Stateful protocol
4.	Domain Name System	Network Protocol / Service	Erlang	
5.	Concurrency Problems • Dinning Philosophers • Sleeping Barber • Cigarette Smokers	Operating Systems	Python/Actors	Race conditions
6.	Lock	Operating Systems	Eventful Session Java	Race conditions
7.	Collection	Data Structures	Mungo	A stack client
8.	File Access	Data Structures and Algorithms	Mungo	File Access client
9.	Concurrent Fibonacci	Concurrent Algorithms	Mungo	
10.	Network Topologies • Ring • Butterfly • All to All • Stencil • Farm (Master-Worker) • Map Reduce	Parallel Algorithms	Pabble and MPI	Parametric algorithms
11.	Concurrent Algorithms • Peano Numbers • Add Server	Concurrent Algorithms	Links, GV	
12.	Memory Coherence	Systems, Hardware	Mungo	

Fig. 3. Usecases in the ABCD online repository