University of Glasgow | School of Computing Science

Session types in the wild

A. Laura Voinea

Primary Supervisor: Dr. Simon Gay
Secondary Supervisor: Dr. Wim Wanderbaude

1st Year PhD Report

June 7, 2016

Contents

1 Introduction

Communication is usually standardised via protocols that specify the possible interactions between the communicating parties in a specific order. Implementations of communication behaviours are based on informal protocol specification and thus informal verification. As a result they are prone to errors such as communication mismatch, when the message sent by one party is not expected by the other party, or deadlock, when two parties are waiting for a message from each other causing the system to block[1]. To allow formal protocol specification within the programming language session types have been devised. Session types describe communication by specifying the type and direction of messages exchanged between two parties[6]. Programmers can express a protocol specification as a session type, which can guarantee, within the scope where the session type applies, that communications will always match and the system will never deadlock. The main goal of the ABCD project[1] is to improve the practice of software development for concurrent and distributed systems through the use of session types. This is to be accomplished through built-in language support for protocol codification in existing languages such as Java or Python, in new languages such as Links[3], inter-language interoperability via session types, and through adapting interactive development environments and modelling techniques to support session types. Logical and automata foundations of session types will be further developed to express a wider class of behaviour and, as need arises, to support the former. Empirical studies to assess methodologies and tools will be carried out, with results being used to improve language and tool design and implementation. To help improve the practice of software development and improve developer adoption there is a need for extending integrated development environments(IDEs) and for carrying out empirical studies to investigate the impact of session types.

Nowadays, distribuded systems are ubiquitous and communication is an important feature and reson for its success. Communication-centred programming has proven to be one of the most successful attempts to replace shared memory for building concurrent, distributed systems. Communication is easier to reason about and scales well as opposed to shared memory, making it a more suitable approach for systems where scalability is a must, as in the case of multi-core programming, service-oriented application or, or web services[1].

Mainstream programming languages and methodologies fail to adequately support communication-centred programming. Support is required to exclude insidious programming errors at the development stage this is key to avoiding the deployment of programs which may be faulty at the execution stage.

2 Thesis

2.1 Thesis Statement

Session types are a useful addition to the syntax and semantics of modern languages. Programming with session types and the constraints that this comes with i.e. linearity can be understood and used by real world programmers. Moreover session types can help programmers understand the problem they are tying to solve with more ease and structure their code better. Session typed languages provides useful safeguards and diagnostic information that lead to a correct porogram with less effort.

communication-based software

2.2 Thesis Outline

communication-based software everywhere!! aaaa!

is the additional complexity justifiable? Can typestate be reasoned about effectively by real programmers?

3 Research Activities

3.1 Mungo

Mungo[1] is a Java front-end tool used to statically check the order of method calls. It is implemented using the JastAdd framework[2]. A protocol or session type is represented as a separate typestate file, associated with a Java class. The protocol definition is described as a sequence of method calls, the order of which determines the validity of the protocol.

The Mungo checks that the object instantiating the class performs method calls as defined by its typestate. If the protocol is not violated, standard .java files are produced for every .mungo file in the package. The resulting code can then be ran as any standard Java code.

Typestate is a refinement of the concept of a type in programming [7, 8]. Typestate is the constraining of a sequence of calls. In programming certain methods can sometimes only be called in certain states. Typestate checks what method is available to be called in which state and what the subsequence state of each method is. Furthermore Typestate can check at compile time that specifications are followed. Through Typestate analysis, it is possible to track the degree of initialisation of variables, guaranteeing that operations would never be applied on improperly initialised data.

The following work was undertaken on the Mungo tool:

- the tool has been moved from the Java 1.4 compiler to the Java 1.8 compiler and adapted to work with the new framework.(joint work with Dr. Dimitrios Kouzapas)
- moving the tool to a new compiler framework was a good opportunity for some refactoring.
- the tool has been extended to support Java enumerations.(joint work with Dr. Dimitrios Kouzapas)
- updated repositories
- work has been undertaken to support generic types
- work has been undertaken to typecheck exceptions, in a simple form at the moment

[1]http://www.dcs.gla.ac.uk/research/mungo/
[2]http://jastadd.org/web/extendj/

Areas of future work:

- aliasing
- typecheck generic types
- support annotations

3.2 StMungo

StMungo (Scribble to Mungo)[3] is a Java-based tool for translating Scribble[4] local protocol specifications into typestate specifications.

Scribble is a language that can describe application-level protocols among communicating systems. A protocol represents an agreement on how participating entities interact with each other.

After the Scribble protocol is translated to a Mungo specification, Mungo**??** is used to generate a Java implementation for the protocol. This tool allows an easy transition from a Scribble global protocol definition to working Java implementation. We start by specifying distributed multiparty protocol in Scribble. We can then use the Scribble tool chain to validate and project the global protocol into a local one describing the interactions from the point of view of a specific participant. For every Scribble local protocol, StMungo will produce .mungo files containing: a typestate specification describing the local protocol as a sequence of method calls, an API for the participant implementing the typestate methods and a main class skeleton calling the methods in the typestate.

To improve this tool various extensions, refactoring has been undertaken:

- extended the tool to translate messages with no payload i.e.

  ```
  message_operator ()
  ```

- extended the tool to translate messages with multiple payload i.e.

  ```
  message_operator ( payload_type1 , ... , payload_typen )
  ```

- extended the tool to translate messages without a message signature i.e.

  ```
  ( payload_type1 , ... , payload_typen )
  ```

- extended the tool to translate messages with annotated payloads i.e.

  ```
  message_operator ( annotation:payload_type)
  ```

- various small improvements to allow most translations to run without having to be edited by a human
- various improvements allowing the tool to crash gracefully
- adapted the tool to work with multiple versions of scribble specification

[3]http://www.dcs.gla.ac.uk/research/mungo/

[4]http://www.scribble.org

- improved the tool by implementing support for special cases of recursions nested in choice structures A simple example of a problematic scribble specification is:

  ```
  global protocol Example(role S, role C) {
  choice at C{
          rcpt(String) from C to S;
      } or {
          msg(String) from C to S;
          rec loop {
                  subject(String) from C to S;
                  continue loop;
              }
          }
      }
  }
  ```

- improved the tool by implementing support for special cases of nested choice inside a recursion

  A simple example of a problematic scribble specification is:

  ```
  global protocol ProtocolName(role S, role C) {
    command(String) from C to S;
    rec overall {
    choice at S
        {
        ok(String) from S to C;
                        choice at S {end(String) from S to C;}
                        or {
                        sum(String) from S to C;
                        }
                        message(String) from S to C;
        } or {   error(String) from S to C;      }
        continue overall;
    }
    }
  ```

- extending the tool to translate to support inlined protocols and sub-protocols(ongoing)
- kept it up to date with changes in Mungo
- refactoring to keep it simple
- regression testing to find any new bugs introduced

Areas of future work:

- test harness
- extension to translate more complex constructs such as interruptible or parallel
- keeping it up to date with changes in Scribble and Mungo

3.3 Usecases

To better understand the expressive power of current session type technology together with any limitations that may need to be addressed, the current use case repository[5] was surveyed as a first step. As a second step, new real-world examples were sought. From the various protocols looked after, representations were attempted for two, Paxos and the File Transfer Protocol(FTP).

The first protocol chosen as a usecase was Paxos. Paxos is a protocol for solving consensus in a network of unreliable processes. It ensures that a single value among the proposed values can be chosen. It assumes an asynchronous, non-Byzantine model.[**?**]

Two major advantages of Paxos are that it is provably correct in asynchronous networks that eventually become synchronous and it does not block if a majority of participants are available.Furthermore it has provably minimal message delays in the best case. Despite it's reputation of being difficult to understand & implement it is widely, a couple of examples would be Google in Chubby, Yahoo use something based on it in ZooKeeper. The protocol comes with three roles and a two-phase approach. A proposer responsible for initiating the protocol, that handles client requests and proposes values to be chosen. An acceptor that responds to messages from proposers by either rejecting them or agreeing in principle and making a promise about the proposals it will accept in the future. An a listener or learner, who wants to know which value was chosen. Each Paxos server can act as any or all 3 roles.

Some representations of the algorithm have been attempted using Scribble, combined with StMungo and Mungo to give a working Java code. However in trying to represent it some shortcomings of the toolset became apparent:

- representing broadcasting
- representing quorum/a majority
- representing express the dynamic aspects such as processes failing, restarting
- express multiple instances of the protocol

The work on Paxos has been presented during the last ABCD meeting in January 2016. Work on a better Paxos representation has been paused for the moment, other tasks taking priority such as improving the tools.

The second protocol chosen was the File Transfer Protocol described by Request for Comments(RFC): 959[6]. FTP is a standard network protocol for transferring files between a client and server on a network. FTP is an unusual protocol in that it utilizes two ports, a data port and a command(control) port. FTP may run in active or passive mode, which determines how the data connection is established. In both cases, the client creates a TCP control connection from a random, usually an unprivileged, port number to the FTP server command port 21. Some representations of the algorithm have been attempted using Scribble, combined with StMungo and Mungo to give a working Java code. However in trying to represent it some shortcomings of StMungo became apparent. Hence, work on an FTP representation has been paused to improve StMungo and Mungo, to allow a better representation. It is planned to be restarted in the autumn.

[5]https://github.com/epsrc-abcd/session-types-use-cases

[6]https://tools.ietf.org/html/rfc959

3.4 User study

Session types good for programmers. Will like/use/find helpful in their day to day programming.

This phd aims to explore how programmers can interact with session types, if they find them useful and in which ways. Make this intrecation more meaningful.

Stuff done in preparation of the user study:

- literature survey of existing studies; some of it summaried
- plan B
- tool develpoment to enable this

Regardless, assessing the utility of typestate through experimentation is likely to be essential in convincing those outwith the research community that it has value, and should be included in contemporary programming languages.

In order to evaluate the practicality of the Hanoi language for typestate modelling, an experimental design was desired with the aim of evaluating the following research questions: Can programmers reason effectively about the semantics of Hanoi? Is the effectiveness with which a programmer can reason about a Hanoi model influ- enced by the presentation of the model? 5.1. Experimentsconsidered 117 Will programmers prefer either the DSL or annotation model types? These questions should be evaluated in the context of one or more of the following tasks: Deciding whether a Hanoi model is semantically valid. Writing code that does not violate constraints in a Hanoi model. Producing a semantically valid Hanoi model against an informal specification. Identifying whether code violates constraints in a Hanoi model. Different experimental designs were considered, oriented around questions related to these tasks, before settling on a final experimental design.

4 Other Activities

Besides staring at code and trying to make sense of greek symbols the following activities were undertaken throughout the year.

Project Group Meetings of various sizes. Weekly meetings consisted mostly of people working on the project in Glasgow, at which I even once lead a paper discussion. Monthly meetings consisted mostly of people working on the project in Edinburgh and Glasgow. As well as three longer meetings that consisted of all project partners: Glasgow, Edinburgh, London and Industry Representatives (which form the Advisory board).

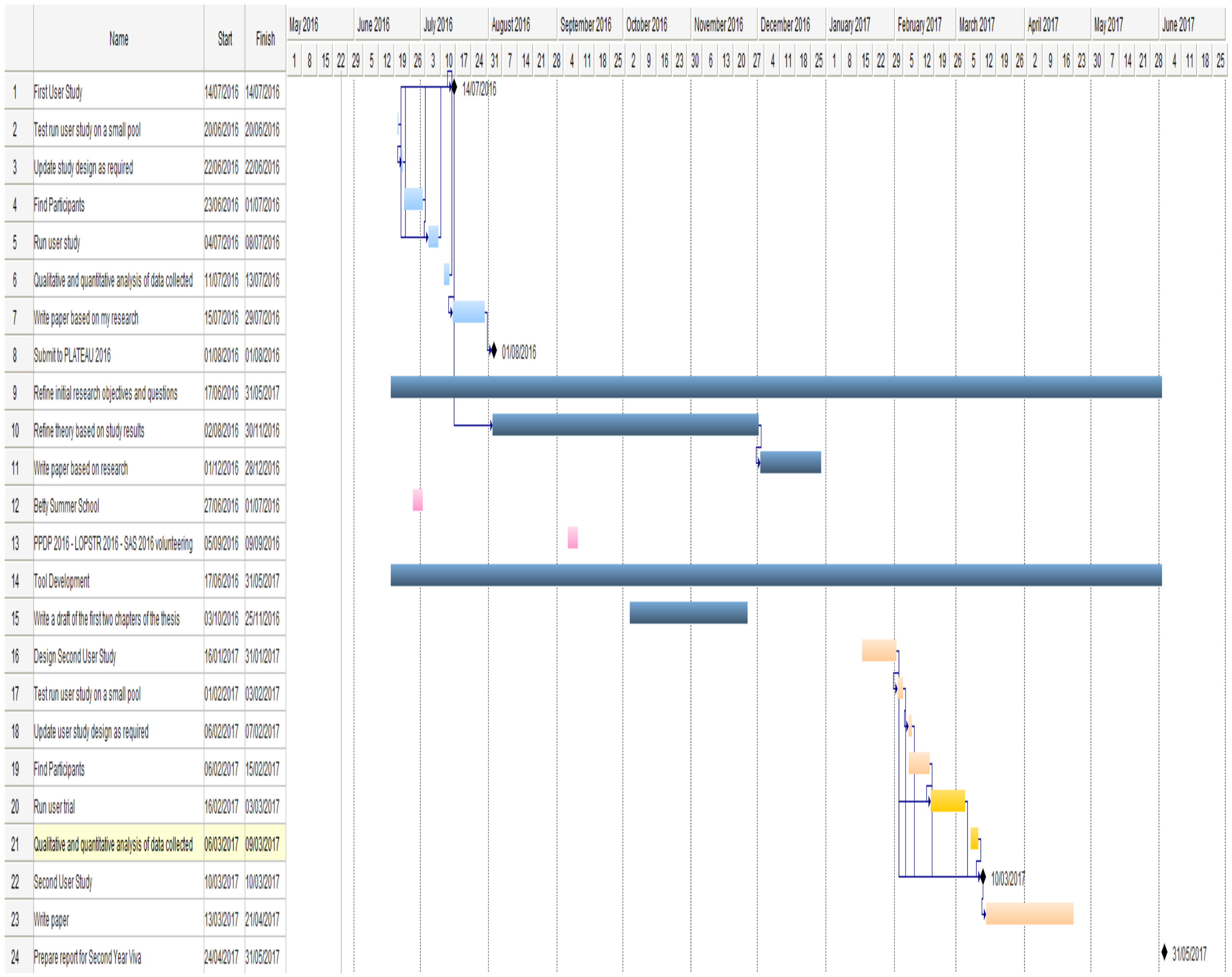
As required by the Graduate School, I attended eight **postgraduate research programmes**, which were designed to help build skills necessary for a researcher in within the University.

As part of the first year of my PhD, I have attended **various seminars, talks and events** in order to get a better understanding of Computing Science as a whole and of my research groups and consecutively the projects place within it.

During this year I have attended various seminars and talks which improved my knowledge of my own ignorance of the field as a whole.

- SPLS The Scottish programming language seminar series
- FATA seminar series.over-view of what is worked on in the research group as well as to enhance my understanding of my own topic area.
- BETTY (Behavioural Types for Reliable Large-Scale Software Systems) meeting 2016.
- Wadlerfest & LFCS30. Event held in celebration of Philip Wadler's 60th birthday. LFCS30, to celebrate the 30th anniversary of the founding of the LFCS.

A Second Year Plan



	Name	Start	Finish
1	First User Study	14/07/2016	14/07/2016
2	Test run user study on a small pool	20/06/2016	20/06/2016
3	Update study design as required	22/06/2016	22/06/2016
4	Find Participants	23/06/2016	01/07/2016
5	Run user study	04/07/2016	08/07/2016
6	Qualitative and quantitative analysis of data collected	11/07/2016	13/07/2016
7	Write paper based on my research	15/07/2016	29/07/2016
8	Submit to PLATEAU 2016	01/08/2016	01/08/2016
9	Refine initial research objectives and questions	17/06/2016	31/05/2017
10	Refine theory based on study results	02/08/2016	30/11/2016
11	Write paper based on research	01/12/2016	28/12/2016
12	Betty Summer School	27/06/2016	01/07/2016
13	PPDP 2016 - LOPSTR 2016 - SAS 2016 volunteering	05/09/2016	09/09/2016
14	Tool Development	17/06/2016	31/05/2017
15	Write a draft of the first two chapters of the thesis	03/10/2016	25/11/2016
16	Design Second User Study	16/01/2017	31/01/2017
17	Test run user study on a small pool	01/02/2017	03/02/2017
18	Update user study design as required	06/02/2017	07/02/2017
19	Find Participants	06/02/2017	15/02/2017
20	Run user trial	16/02/2017	03/03/2017
21	Qualitative and quantitative analysis of data collected	06/03/2017	09/03/2017
22	Second User Study	10/03/2017	10/03/2017
23	Write paper	13/03/2017	21/04/2017
24	Prepare report for Second Year Viva	24/04/2017	31/05/2017

B User Study Plan

1 Introduction

1.1 Research Question Identification & Description

1.2 Justification

1.3 Hypothesis

The survey was designed to validate or dispute the following hypothesis:

H0. There is no correlation between session types and programmers...

H1. Concern about security influences user behaviour (i.e. applications used, precautions taken) in the context of public Wifi

Besides validating or disputing the hypothesis, we added some exploratory questions to get more information about the participantsâĂŹ behaviour and interests when using public Wifi and find any interesting correlations between these and security behaviour.

2 Experimental Description

The next part of this document is concerned with what will be evaluated and how. The following sections will define the methodology and approach that will be taken.

2.1 Tasks to complete

The participants will be given three main tasks to complete.

1. Iterator example in Mungo. Code provided. Spot and correct the error. Add the remove operation.
2. SMTP[1] example in Scribble and Mungo. Code provided. StMungo introduced as well. Spot and correct the error one in scribble specification, second one in Mungo specification.
3. Implement a small example based on an informal specification. Buyer seller would be a good candidate here.

2.2 Variable/Experimental conditions

The purpose of this study is exploratory so many variables will be measured.

Independent variables:

- Programming expertise

Dependent variables to be measured:

- time to complete each task
- how many times the code is compiled
- how many bugs/errors
- is the end result correct

[1] http://www.ietf.org/rfc/rfc2821.txt

2.3 Data Collection/Measurements/Observations

Dependent variables to be measured:

- time to complete each task
- how many times the code is compiled
- how many bugs/errors are encountered in the process
- is the end result correct
- participants' opinion about the tools/session types

2.4 Design

Considering that the aim of this experiment is exploratory and the learning effect (performance improving with experience, even over very short periods of time) is of little concern in this case a within subjects design, every participant perform every task under every condition, will be used.

2.5 Participants/Subjects/Users (Controlled Variable)

The only requirement for participants is that they have some Java experience. Factors like level of programming proficiency and upfront knowledge of behavioural types or session types will be taken into consideration.

2.6 Experimental location

The experiment will be carried out in Sir Alwyn Williams building, room F112. This office has been chosen as it allows a higher degree of control over the environment, programming one in particular.

2.7 Experimental schedule

This section outlines all of the parts of the experiment and a rough estimate how long each will take. The time taken to complete the experiment may be longer or shorter depending on the level of programming expertise.

1. 5 minutes to explain the experiment
2. 15 minute practical tutorial on session types and typestate
3. approximately 10 minutes for task 1
4. approximately 15 minutes for task 2
5. approximately 20 minutes for task 3
6. 5 minute interview
7. 5 minute to complete survey
8. 5 minutes for any questions the participant might have