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 27, 2016

Contents

1 Introduction

Nowadays, distributed systems are ubiquitous and communication is an important feature and reason 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 applications or cloud computing[1].

Communication is usually standardised via protocols that specify the possible interactions between the communicating parties in a specific order. Mainstream programming languages fail to adequately support the development of communication-centred software. Thus, 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[1], 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 are to be carried out, with results being used to improve language and tool design and implementation.

1.1 Research Questions

This PhD will attempt to answer the following questions considering both the theoretical and practical aspects of session types.

- How can programmers use session types in meaningful ways?
  - What are the strengths and weaknesses of the options for programming with session types?
  - Can programmers reason effectively about communication using session types?
  - Does the method of expressing session types influence the programmers' the ability to reason effectively about the system?

  These questions will be explored mainly trough user studies, the first of which is outlined below in 3.4.

- What are the theoretical and practical challenges to expressing real life protocols with session types? How does the theory need to be extended to help overcome these? How do existing tools need to be improved? Is there a need for additional tools?

1.2 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 provide useful additional safeguards and diagnostic information that lead to a system with the expected behaviour with less effort.

1.3 Outline

I have spent the first 9 months of my PhD reviewing existing work within the fields of session types, typestate, and language usability and evaluation. A short literature review of the most relevant work can be found in 2. Alongside that, I have done some implementation work on two session type tools developed at the University of Glasgow as part of the ABCD project[1], namely Mungo, and StMungo[26], as well as looked at some possible usecases. This will be discussed in more detail in section 3. At present my focus is on defining a suitable methodology and carrying out a user study to explore how ”real-world” programmers will interact with session types through Mungo, StMungo and Scribble. More on this can be found in 3.4.

Future work along with a plan for the second year is discussed in section 5.

2 Literature review

2.1 Session types, a short history

Session types have been introduced by Honda[19], and further extended by Takeuchi et al.[32], Honda et al.[20], and others.

Session types are a way of modelling communication between different entities and enforcing the order of communication, as well as the types of the messages that may be sent. A session type describes the types of individual messages and the state transitions of the protocol i.e. the allowed sequences of messages. Each end of a communication channel is associated with a session type, which is the dual of the other, meaning that if the session type for one process prescribes that it may only send a string, then the other party may only receive an string. Type-checking may be used to verify processes compliance to the session type system and establish properties such as deadlock and race freedom, or session fidelity between two parties.

Session types have and continue to be the subject of a wide body of work aimed at extending their theoretical foundations as well as developing new tools and techniques. For example, session types have been augmented with subtyping polymorphism to enable protocols to describe richer behaviours[15]. They have been extended to ensure the progress property and deadlock freedom [11]. Session types have also been extended to multiparty session types to support communication instances with more than two participants while still guaranteeing the absence of deadlock[21]. In other work [8], global session types have been used to detect choreographies that can be realized in the context of web services.

Session types have been successfully applied in functional programming languages [33, 4], object-oriented languages like Java [22, 16], low-level programming languages like C in [28], dynamically-typed languages like Python[27] or Erlang [25], or in the operating systems context with the Sing# language [12].

2.2 Typestate

Typestate is a refinement of the concept of a type in programming. A typestate defines the valid sequence of operations that can be performed on an instance of a certain type by associating state information with variables of that type. This state information can then be used at compile-time to determine the operations that can be invoked with valid results on an instance of a type. A typical example is that of a file, which can be read or written only after being open, and never after being closed. In most cases such constraints on the sequences of method calls are at best informally documented through method descriptions or comments. However in this form, the information cannot be used by the compiler to detect any violations.

Since typestate has been introduced in[30], there have been many efforts to add it to practical programming languages. In [10], the concept of typestate, originally introduced for imperative programs, was extended for the object-oriented paradigm, resulting in the Fugue system. Sing# [13] is an extension of C# which incorporates typestate-like contracts, to specify protocols. Sing# has been successfully used to implement Singularity, a message-passing based operating system used for research at Microsoft. A core calculus for *Sing#* has been introduced in Bono et al. [7] proved type safe.

Plural[6] is another noteworthy example. Based on Java, it has been used to study access control systems[5] and transactional memory[3], and to evaluate the effectiveness of typestate in Java APIs[6]. However, Plural's use of annotations to define typestates is not ideal, while annotations are convenient for attaching extra information that can be consumed by tools, it makes expressing more complex concepts difficult and can be a burden on the programmer. Plural provides static analysis, and does not alter the runtime representation of classes in any way. Consequently, dynamic checking might still be needed if the system will interact with components that are not trusted. Stemmed from the research on Plural, is Plaid, a general purpose programming language that goes one step further and incorporates typestates as native feature of the programming language, and encourages developers to design objects around their protocol [2, 31]. Instead of class definitions, a program consists of state definitions containing methods that cause transitions to other states. Transitions are specified in a similar way to Plural's pre- and post-conditions. Similarly to classes, states can be structured into an inheritance hierarchy.

Typestates are are well-suited to represent behavioral type refinements, such as session types. In [17] Gay et al. proposed the integration of session types and object-oriented programming by using typestate. A session type is considered as being a special case of typestate in [17], constraining the use of send and receive methods, and introduce a session-inspired notation for specifying method sequences.

For the Mungo tool[26], the approach of [17] is followed, to formalise a typestate inference system for a core object-oriented language, Java, and prove its correctness. Mungo is implemented as a front-end typechecking tool for a subset of Java. Inference removes the need for typestate annotations on parameters and return types as in the system by Gay et al [17]. The possible sequences of method calls are explicitly defined, rather than being consequences of pre- and post-conditions as in the Plural system. A typestate in Mungo can depend on the return value of a method call.

The basic difference between Mungo and other typestate approaches is the Java-like syntax for describing state machine protocols, instead of associating pre- and post-conditions with methods. The motivation for developing such a typestate syntax is the ability to globally describe object behaviour and subsequently the general integration of session behaviour in objects.

One of the challenges in typestate analysis is the problem of aliasing. Aliasing occurs when an object has more than one reference or pointer that points to it. For a correct analysis, a state change to a given object must be reflected in all references that point to that object. Otherwise method calls via aliases can cause inconsistent state changes. However tracking all such references is a difficult problem. Fugue follows an “adoption and focus” approach, while Plural and Plaid follow permission-based approaches.

To avoid the possibility of conflicting state changes through different aliases, Mungo goes for linearity and does not allow aliasing of objects that have a typestate protocol. It should however be possible to orthogonally add more a more flexible aliasing control.

2.3 Language usability and evaluation

Programming languages should help programmers develop software effectively. With advances in technology, they should be concerned more with programmer efficiency rather than programming language efficiency. The most successful languages are not necessarily the most efficient ones, but the ones that are the easiest to use to satisfactorily achieve one’s goals. Languages such as Java or Python are widely used, despite known flaws, while languages like Agda, or Idris are rarely even heard of outside academic circles. Language designers seem to overlook the people that will use their language. New language features are advertised as being an improvement over existing work, and, so, desirable, however, little evidence is usually brought forth to support such claims. In recent years, this has become a more visible issue, and there are now more efforts to improve it.

One such effort is the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH[1]. The aim of the workshop is to discuss methods, metrics and techniques for evaluating the usability of languages and language tools. The workshop highlights some areas of interest as: empirical studies of programming languages, methodologies and philosophies behind language and tool evaluation, software design metrics and their relations to the underlying language, user studies of language features and software engineering tools, visual techniques for understanding programming languages, critical comparisons of programming paradigms, tools to support evaluating programming languages, psychology of programming, domain specific language

[1] http://2016.splashcon.org/track/plateau2016

usability and evaluation. As such it seemed to be the perfect place to start when trying to design an empirical study for a new construct.

3 Research Activities

3.1 Mungo

Mungo[26] is a Java front-end tool, developed by Dr. Dimitrios Kouzapas at the University of Glasgow, used to statically check the order of method calls i.e. its typestate. It is implemented using the JastAdd framework[2][18]. 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.

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.

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(such as getting rid of dead code or method extraction).
- 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

Areas of future work:

- aliasing
- typecheck generic types
- support annotations

[2] http://jastadd.org/web/extendj/

3.2 StMungo

StMungo (Scribble to Mungo)[3] is a Java-based tool, developed by Dr. Ornela Dardha at the University of Glasgow, that translates a Scribble[29, 34] local protocol into a Mungo specification and skeleton socket-based implementation code. The resulting code is typechecked using Mungo. Scribble is a protocol description language that can describe how two or more participating entities interact should 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 toolchain 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 have been implemented:

- 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
- 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;
  ```

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

```
              }
          }
      }
  }
```

- 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(such as method extraction or getting rid of dead code) to keep everything simple
- regression testing to find any new bugs introduced

Areas of future work:

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

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[4] 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).

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

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.[23]

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[9], 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.[24]

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[14]. 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.

Work on both usecases is planned to be restarted in the autumn, and together with tool extensions make the base for a paper later on in the year.

3.4 User study

New programming language constructs are more often than not introduced without first exploring how well suited their are for their purpose or how they would be used in the real world. While proving they solve the problem is a good thing, checking how well they solve it would be nice. To explore how well programmers would interact with session types an initial study has been devised. This can be found in appendix **??**. It is intended to explore whether programmers can learn what session types are, identify how they should be used, and reason about session typed code correctly. At this stage there are little constraints on the type of participant. Since one of the claims behind session types is that they make reasoning about communication easier, participants with less programming experience are welcome. It would be interesting to see if there are any differences and what differences there are between participants of different backgrounds. The only requirement in this case being a working knowledge of Java.

Difficulties: choosing meaningful for the user to carry out, deciding what constitutes a good understanding of session types.

4 Other Activities

As part of the first year of my PhD, various additional activities have been undertaken, such as training courses(detailed in appendixC), ABCD group meetings of various sizes, seminars and talks(e.g. Tht Scottish programming language seminar series, FATA seminars) which improved my knowledge of the field and gave me some insight of the exciting research ongoing, as well as my own ignorance. Some notable events attended were the BETTY (Behavioural Types for Reliable Large-Scale Software Systems)[5] meeting in March, and Wadlerfest[6], held in celebration of Philip Wadler's 60th birthday, or LFCS30[7], to celebrate the 30th anniversary of the founding of the LFCS.

5 Future Work

Some of the future work planned has been highlighted throughout this report. The activities planed for next year fall into several interconnected themes:

- Language usability and evaluation. As mentioned above in 3.4 a user study is in plan for the near future, its results to be analysed and written up and submitted to PLATEAU 2016[8]. A second more focused study is also in plan for the later part of the year.
- Tool development. Some of the work that is to be done has already been discussed earlier, however it is expected that the need for various different improvements will arise. Thus this will be an ongoing activity through the year.
- Training and development. This category contains any training courses required by the school as well as any academic events to be attended. Over the summer, I will attend BETTY (Behavioural Types for Reliable Large-Scale Software Systems) in Limassol, Cyprus. The summer school will cover closely related areas of interest such as multiparty session types, linear logic, Practical programming with session types. In September I will help with the PPDP, LOPSTR and SAS conferences in Edinburgh.

[5] http://www.behavioural-types.eu/meetings/wg-mc-meetings-17th-18th-march-2016-in-malta

[6] http://events.inf.ed.ac.uk/wf2016/

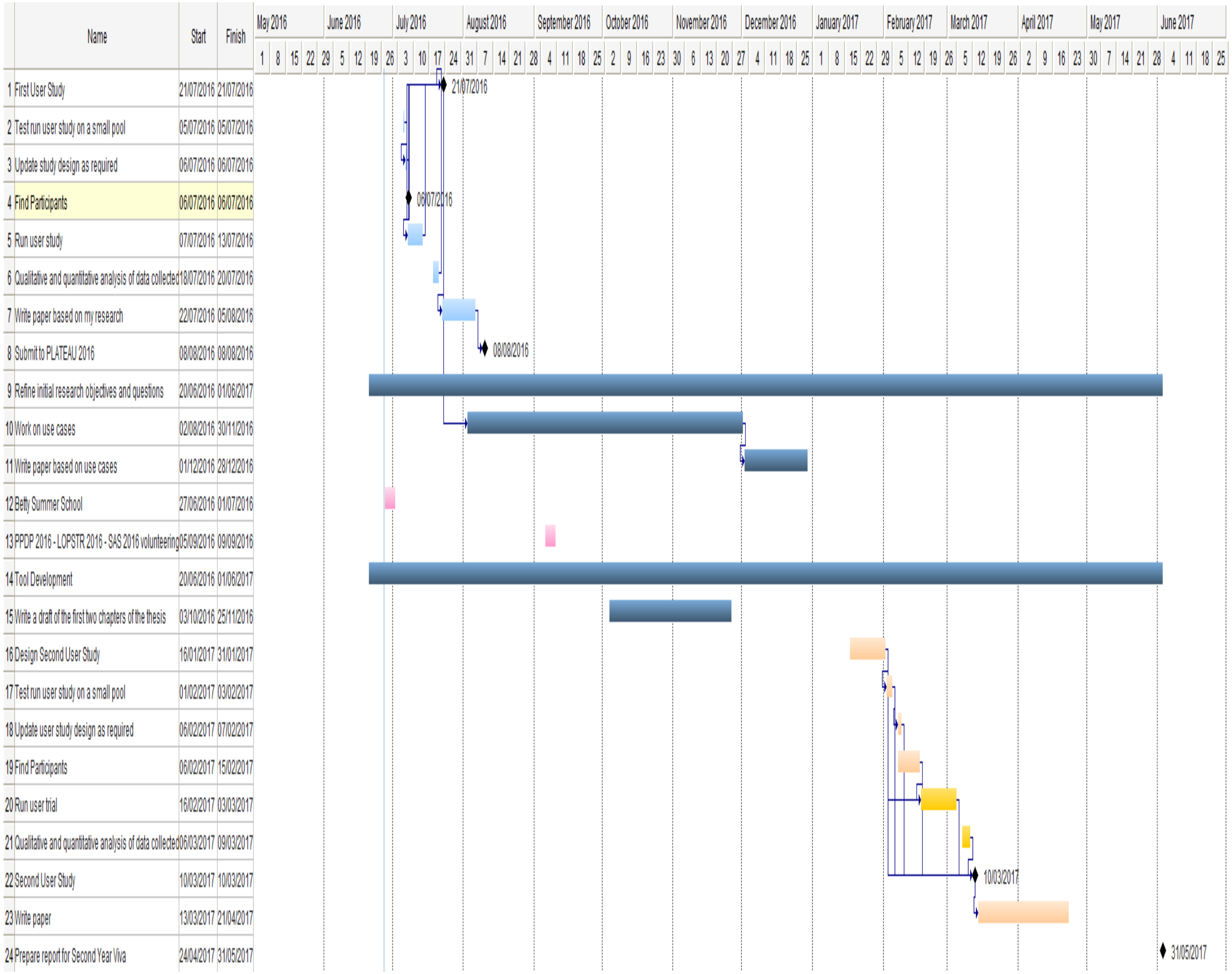
[7] http://events.inf.ed.ac.uk/lfcs30/

[8] http://2016.splashcon.org/track/plateau2016

- Miscellaneous. This category contains work to be continued such as the usecases**??**, and any new ideas that will arise from the user study.

For a detailed plan, a Ganntt chart is available in appendix A.

A Second Year Plan



	Name	Start	Finish
1	First User Study	21/07/2016	21/07/2016
2	Test run user study on a small pool	05/07/2016	05/07/2016
3	Update study design as required	06/07/2016	06/07/2016
4	Find Participants	06/07/2016	06/07/2016
5	Run user study	07/07/2016	13/07/2016
6	Qualitative and quantitative analysis of data collected	18/07/2016	20/07/2016
7	Write paper based on my research	22/07/2016	05/08/2016
8	Submit to PLATEAU 2016	08/08/2016	08/08/2016
9	Refine initial research objectives and questions	20/06/2016	01/06/2017
10	Work on use cases	02/08/2016	30/11/2016
11	Write paper based on use cases	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	20/06/2016	01/06/2017
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 Experimental Description

The following sections will define the methodology and approach that will be taken.

1.1 Tasks to complete

The participants will be given three main tasks to complete with various subtasks.

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(detail).

1.2 Variable/Experimental conditions

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

1.3 Data Collection/Measurements/Observations

The purpose of this study is exploratory. 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

1.4 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.

1.5 Participants/Subjects/Users (Controlled Variable)

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

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

1.6 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, the programming one in particular.

1.7 Experimental schedule

This section outlines all of the parts of the experiment and a rough estimate how long each will take. After completing all tasks, anticipated to take roughly 45 minutes, the participants will be asked to answer a short survey rating their experience using session types. 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, typestate, the tools to be used
3. approximately 10 minutes for task 1
4. approximately 15 minutes for task 2
5. approximately 20 minutes for task 3
6. 5 minute short informal interview
7. 5 minute to complete survey
8. 5 minutes for any questions the participant might have

C Training Needs Analysis Form

University of Glasgow | College of Science & Engineering

Training Needs Assessment and Record 2015

Name	Adriana Laura Voinea	Year of Study	First	School	Computing Science		
Date Discussed with supervisor	03/06/2016	Date Submitted	16/06/2016	Funding Source	EPSRC	Total Credits	11

The areas below correspond to those detailed in the Researcher Development Framework . You should select the areas you would like to develop and discuss these with your supervisors. You may be able to obtain the skills you require through training and/or practical experience. The completed TNA/Record should be submitted to the Graduate School when you submit your paperwork for your annual progress review.

DOMAIN A – KNOWLEDGE AND INTELLECTUAL ABILITIES				
	Development Required	Courses identified from the Doctoral Research training programme and elsewhere and any practical experiences you intend to undertake to develop the skills required	Courses attended and dates	Credits
A1 – Knowledge Base Includes subject knowledge, research methods, information search skills and management, languages	Knowledge of Statistics	RSDA 6107 Introduction to Statistical Inference	Attended on 27/05/16	1
		RSDA 6108 Introduction to Statistical Modelling	Attended on 28/05/16	1
		RSDA 6109 Introduction to Design and Analysis of Experiments	Attended on 29/05/16	1
	Programming language theory	Scottish programming language seminars	Attended on various dates throughout the year	N/A
		Betty meeting	Attended on 17-18 March 2016	N/A
A2 – Cognitive abilities	Literature review	RSDA 6083 Literature critique/ review SC	Not attended as it was cancelled due to illness	N/A

Includes analysing, synthesising, critical thinking, evaluation and problem-solving skills				
A3 – Creativity Includes developing inquiry skills, intellectual insight, innovation, constructing an argument, intellectual risk	Inquiry skills, constructing an argument	Project meetings Scottish programming language seminars FATA seminars	Attended on various dates throughout the year Attended on various dates throughout the year Attended on various dates throughout the year	N/A N/A 1

DOMAIN B – PERSONAL EFFECTIVENESS				
	Development Required	Selected Courses from the Doctoral Researcher Development Training Programme	Date Course(s) Attended	Credits
B1 – Personal qualities Includes enthusiasm, perseverance, integrity, self-confidence, responsibility	Self-confidence, responsibility	Mini Project	10/15-01/16	N/A
B2 – Self management Includes preparation and prioritisation, commitment, time-management, work-life balance, responsiveness to change	Prioritisation, time-management, project management	RSDC 6001 Project management – An Introduction	Attended on 01/03/2016	1
B3 – Professional and career development Includes career management, continuing professional development,	Networking, continuing professional development	Attended Seminars (SPLS) Attended Betty Meeting WadlerFest/LFCS30	Attended on various dates throughout the year Attended on 17-18 March 2016 Attended on 11-13 April 2016	N/A N/A 1

responsiveness to opportunities, networking				

DOMAIN C – RESEARCH GOVERNANCE AND ORGANISATION				
	Development Required	Selected Courses from the Doctoral Researcher Development Training Programme	Date Course(s) Attended	Credits
C1 – Professional conduct Includes health and Safety, ethics, principles and sustainability, IPR/copyright, respect and confidentiality, attribution and co-authorship, appropriate practice	Improve knowledge of IPR/copyright, respect and confidentiality, attribution and co-authorship	RSDC 6023 Research Integrity Equality and Diversity Online Training	Attended on 21/01/2016 Completed on 25/05/2016	1 1
C2 – Research management Research strategy Project planning and delivery Risk management	Improve knowledge of data management	RSDC 6025: Research Data Management	Attended on 17/02/2016	1
C3 – Finance, funding and resources Includes incomes and funding generation, financial management, infrastructure and resources	Funding generation, financial management, infrastructure and resources	FATA seminars	Throughout the year	N/A

DOMAIN D – ENGAGEMENT, INFLUENCE AND IMPACT				
	Development Required	Selected Courses from the Doctoral Researcher Development Training Programme	Date Course(s) Attended	Credits
D1 – Working with others	Collaboration skills	Attended Project Meetings	Multiple dates: September 2015 – now	N/A

Includes collegiality, team-working, people management, supervision, mentoring, influence, leadership, collaboration, equality and diversity				
D2 – Communication and dissemination Includes communication methods, communication media, publication	Improve presentation skills	RSDD6002: Presenting with Impact	Attended on 12/05/2016	1
D3 – Engagement and impact Includes teaching, public engagement, enterprise, policy, society and culture and global citizenship	Teaching Teaching training	Tutored 2nd Year Course: JOOSE2 Tutored 2nd Year Course: WAD2 Graduate Teaching Assistant Statutory Training	10/2015 – 03/2016 01/2016 – 03/2016 Attended on 26/20/2015	N/A 1

References

[1] A basis for concurrency and distribution. http://groups.inf.ed.ac.uk/abcd/.

[2] Jonathan Aldrich, Joshua Sunshine, Darpan Saini, and Zachary Sparks. Typestate-oriented programming. In *Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, (OOPSLA)*, pages 1015–1022. ACM Press, 2009.

[3] Nels E. Beckman, Kevin Bierhoff, and Jonathan Aldrich. Verifying correct usage of atomic blocks and typestate. In *Proceedings of the 23rd ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 227–244. ACM Press, 2008.

[4] Karthikeyan Bhargavan, Ricardo Corin, P-M Deniélou, Cédric Fournet, and James J Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE*, pages 124–140. IEEE, 2009.

[5] Kevin Bierhoff and Jonathan Aldrich. Modular typestate checking of aliased objects. In *Proceedings of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 301–320. ACM Press, 2007.

[6] Kevin Bierhoff, Nels E. Beckman, and Jonathan Aldrich. Practical API protocol checking with access permissions. In *Proceedings of the 23rd European Conference on Object Oriented Programming (ECOOP)*, pages 195–219, 2009.

[7] Viviana Bono, Chiara Messa, and Luca Padovani. Typing copyless message passing. In *Proceedings of the 20th European Symposium on Programming (ESOP)*, volume 6602 of *Springer Lecture Notes in Computer Science*, pages 57–76, 2011.

[8] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8:1–8:78, June 2012.

[9] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407. ACM, 2007.

[10] Robert DeLine and Manuel Fähndrich. Typestates for objects. In *Proceedings of the 18th European Conference on Object-Oriented Programming (ECOOP)*, volume 3086 of *Springer Lecture Notes in Computer Science*, pages 465–490, 2004.

[11] Mariangiola Dezani-Ciancaglini, Ugo de' Liguoro, and Nobuko Yoshida. On Progress for Structured Communications. In Gilles Barthe and Cédric Fournet, editors, *TGC'07*, volume 4912 of *LNCS*, pages 257–275. Springer, 2008.

[12] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R Larus, and Steven Levi. Language support for fast and reliable message-based communication in singularity os. *ACM SIGOPS Operating Systems Review*, 40(4):177–190, 2006.

[13] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi. Language support for fast and reliable message-based communication in Singularity OS. In *Proceedings of the EuroSys Conference*, pages 177–190. ACM Press, 2006.

[14] File transfer protocol, RFC 959. `https://tools.ietf.org/html/rfc959`.

[15] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2):191–225, November 2005.

[16] Simon J Gay, Vasco T Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z Caldeira. Modular session types for distributed object-oriented programming. In *ACM Sigplan Notices*, volume 45, pages 299–312. ACM, 2010.

[17] Simon J. Gay, Vasco T. Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *Proceedings of the 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 299–312. ACM Press, 2010.

[18] Görel Hedin. An introductory tutorial on JastAdd attribute grammars. In *Generative and Transformational Techniques in Software Engineering III*, volume 6491 of *Springer Lecture Notes in Computer Science*, pages 166–200, 2011.

[19] Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, pages 509–523. Springer Berlin Heidelberg, 1993.

[20] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *In ESOP98, volume 1381 of LNCS*, pages 122–138. Springer-Verlag.

[21] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *SIGPLAN Not.*, 43(1):273–284, January 2008.

[22] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *ECOOP 2008–Object-Oriented Programming*, pages 516–541. Springer, 2008.

[23] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[24] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[25] Dimitris Mostrous and VascoT. Vasconcelos. Session typing for a featherweight erlang. In Wolfgang De Meuter and Gruia-Catalin Roman, editors, *Coordination Models and Languages*, volume 6721 of *Lecture Notes in Computer Science*, pages 95–109. Springer Berlin Heidelberg, 2011.

[26] Mungo project homepage. `http://www.dcs.gla.ac.uk/research/mungo/tools.html`.

[27] Rumyana Neykova. Session types go dynamic or how to verify your python conversations. *arXiv preprint arXiv:1312.2704*, 2013.

[28] 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.

[29] Scribble project homepage. `www.scribble.org`.

[30] Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Trans. Softw. Eng.*, 12(1):157–171, 1986.

[31] Joshua Sunshine, Karl Naden, Sven Stork, Jonathan Aldrich, and Éric Tanter. First-class state change in Plaid. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA)*, pages 713–732. ACM Press, 2011.

[32] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *In PARLE94, volume 817 of LNCS*, pages 398–413. Springer-Verlag, 1994.

[33] Vasco T. Vasconcelos, Simon J. Gay, and António Ravara. Type checking a multithreaded functional language with session types. *Theor. Comput. Sci.*, 368(1-2):64–87, December 2006.

[34] Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The Scribble protocol language. In *Proceedings of the 8th International Symposium on Trustworthy Global Computing (TGC)*, volume 8358 of *Springer Lecture Notes in Computer Science*, pages 22–41, 2013.