Pre-Proceedings of WWV 2013

The 9th International Workshop
on Automated Specification and Verification
of Web Systems

António Ravara and Josep Silva (Ed.)

UNIVERSITÀ
DEGLI STUDI
FIRENZE

António Ravara and Josep Silva (Ed.)

Automated Specification and Verification of Web Systems

9th International Workshop, WWV 2013

Firenze, Italy, June 6, 2013

Pre-Proceedings

Università degli Studi di Firenze

Preface

This book contains the papers presented at the 9th International Workshop on Automated Specification and Verification of Web Systems, WWV 2013, which is held June 6, 2013, as part of DisCoTec 2013, the 8th International Federated Conference on Distributed Computing Techniques, that includes DAIS 2013, the 13th IFIP International Conference on Distributed Applications and Interoperable Systems, COORDINATION 2013, the 15th International Conference on Coordination Models and Languages, FMOODS/FORTE 2013, the 15th Formal Methods for Open Object-Based Distributed Systems & Formal Techniques for Networked and Distributed Systems, CS2Bio 2013, the 4th International Workshop on Interactions between Computer Science and Biology, and ICE 2013, the 6th Interaction and Concurrency Experience. Previous WWV workshops were held in Stockholm (2012), Reykjavik (2011), Vienna (2010), Hagenberg (2009), Siena (2008), Venice (2007), Cyprus (2006), and Valencia (2005). Information about the workshop can be found at:

`http://users.dsic.upv.es/~jsilva/wwv2013/`

The Workshop on Automated Specification and Verification of Web Systems (WWV) is a yearly workshop that aims to provide an interdisciplinary forum to facilitate the cross-fertilization and the advancement of hybrid methods that combine Rule-based programming, Automated software engineering, and Web-oriented research. Started in 2005, the series of this workshop established itself as a lively, friendly event with many interactions and discussions.

WWV has a reputation for being a lively, friendly forum for presenting and discussing work in progress. Formal proceedings are produced only after the symposium so that authors can incorporate this feedback in the published papers. The WWV 2013 post-proceedings will be published as a volume of the Electronic Proceedings in Theoretical Computer Science (EPTCS).

This edition, ten papers were submitted to the workshop, and the Programme Committee decided to accept six papers, basing this choice on their scientific quality, originality, and relevance to the workshop. Each paper was reviewed by at least three Program Committee members or external referees. In addition to the six contributed papers, this year the workshop includes the invited talks by two outstanding speakers: Gerhard Friedrich (Universität Klagenfurt, Austria) and François Taïani (Université de Rennes 1, France).

We want to thank the Program Committee members, who worked diligently to produce high-quality reviews for the submitted papers, as well as all the external reviewers involved in the paper selection. We are very grateful to the DisCoTec 2013 General Chair Michele Loreti and the Workshops Chair Rosario Pugliese, and the local organizers for the great job they did in preparing the workshop. We would also like to thank Andrei Voronkov for his excellent EasyChair system that automates many of the tasks involved in chairing a conference.

June, 2013
Firenze, Italy

António Ravara and Josep Silva
Program Chairs

Organization

Program Chairs

António Ravara	Universidade Nova de Lisboa, Portugal
Josep Silva	Universitat Politècnica de València, Spain

Program Committee

Jesús Almendros	University of Almeria, Spain
Maria Alpuente	Universitat Politècnica de València, Spain
Demis Ballis	University of Udine, Italy
Daniela Da Cruz	Universidade do Minho, Portugal
Raymond Hu	Imperial College London, United Kingdom
Ivan Lanese	University of Bologna/INRIA, Italy
Anders Møller	Aarhus University, Denmark
Elie Najm	Telecom ParisTech, France
Kostis Sagonas	University of Uppsala, Sweden
Francesco Tiezzi	IMT, Institute for Advanced Studies Lucca, Italy
Emilio Tuosto	University of Leicester, United Kingdom

Additional Reviewers

Luca Cesari	University of Florence, Italy
Francisco Frechina	Universitat Politècnica de València, Spain
Marco Giunti	Universidade Nova de Lisboa, Portugal
Andrea Margheri	University of Florence, Italy
Rumyana Neykova	Imperial College London, United Kingdom
Sonia Santiago	Universitat Politècnica de València, Spain
Hugo Torres Vieira	Universidade de Lisboa, Portugal

Table of Contents

Self-Healing Service-Based Processes

Gerhard Friedrich

Universität Klagenfurt

Abstract

This talk introduces a self-healing approach to handle exceptions in service-based processes and to repair the faulty activities with a model-based approach. During execution, whenever an exception arises, diagnosis of the failure is performed and repair plans are generated by taking into account information about the process execution, constraints posed by the process structure, dependencies among data, and available repair actions.

However, in practice complete specifications of process activities are not available. Therefore, diagnosis and repair methods for partial behavior models are of great importance. We show that if the assumption of complete behavioral models is lifted, basic diagnosis and repair problems reside on the second level of the Polynomial Hierarchy. The talk also describes the main features of the prototype developed to validate the proposed repair and diagnosis approach for composed Web services. The self-healing architecture for repair handling and the experimental results are illustrated.

Deconstructing Complex Distributed Platforms: A Report From the Trenches

François Taïani

Université de Rennes 1

Abstract

Distributed services play an increasing role in our daily lives and our economy. Unfortunately, as they importance grow, so does their complexity, and the difficulty to analyze, verify, and validate them. In this talk, I will provide an overview of our experience analyzing real-life distributed platforms, and the lessons we learnt doing so. One key problem is the fact that real-life distributed systems usually rely on large stacks of legacy and third-party software. Because of that, they usually cannot be fully analyzed with entirely automated approaches. I will use our experience in this area (which has mainly exploited heuristics, and semi-automatic analysis tools) to suggest potential synergies between verification techniques, and interactive analysis tools.

Local Type Checking for Linked Data Consumers

Gabriel Ciobanu
Romanian Academy, Institute of Computer Science, Blvd. Carol I, no. 8, 700505 Iaşi, Romania
gabriel@info.uaic.ro

Ross Horne
Kazakh British Technical University, Faculty of Information Technology, Tole Bi 59, Almaty, Kazakhstan
ross.horne@gmail.com

Vladimiro Sassone
University of Southampton, Electronics and Computer Science, Southampton, United Kingdom
vs@ecs.soton.ac.uk

The Web of Linked Data is the cumulation of over a decade of work by the Web standards community in their effort to make data more Web-like. We provide an introduction to the Web of Linked Data from the perspective of a Web developer that would like to build an application using Linked Data. We identify a weakness in the development stack as being a lack of domain specific scripting languages for designing background processes that consume Linked Data. To address this weakness, we design a scripting language with a simple but appropriate type system. In our proposed architecture some data is consumed from sources outside of the control of the system and some data is held locally. Stronger type assumptions can be made about the local data than external data, hence our type system mixes static and dynamic typing. Throughout, we relate our work to the W3C recommendations that drive Linked Data, so our syntax is accessible to Web developers.

1 Introduction

Linked data is raw data published on the Web that makes use of URIs to establish links between datasets. The use of URIs to identify resources allows data about resources to be looked up (dereferenced) using a simple protocol, and for the data returned to contain more URIs that can also be looked up. Linked Data consumers can crawl the Web of Linked Data to pull in data that can enrich a Web application. For example the 2012 Olympics Website used Linked Data to help journalists discover and organise statistics about relatively unknown medal winners during the games. Due to links bringing down barriers between datasets, the Web of Linked Data is amongst the worlds largest datasets in the hands of Web developers.

We describe a simple, but appropriate architecture for a Linked Data consumer. We then address the problem of designing a type system for this architecture. Linked Data published on the Web from multiple sources is inherently messy, so data arriving over HTTP must be dynamically type checked. Dynamically type checked data is then loaded into a local triple store, that contains a view of the Web of Linked Data relevant to the Linked Data consumer. Once the consumed Linked Data has been dynamically typed checked, queries and scripts over the local data can be type checked using a mix of dynamic and static type checking. Finally, because we can never have a global view of the Web of Linked Data, we take care to design our type system so that type checking is local. To achieve this we select a useful subset of the W3C standards SPARQL, RDF Schema and OWL to design our type system.

In Section 2, we describe the Linked Data architecture, with an emphasis on consuming Linked Data. In Section 3 we argue for a notion of type that aligns with the relevant W3C recommendations and is as

simple as possible whilst picking up basic programming errors. In Section 4, we define the syntax of our scripting language for consuming Linked Data and the rules of our type system.

2 Application Architecture for Linked Data

2.1 From a Web of Documents to a Web of Data

In 1989, Tim Berners-Lee proposed a hypertext system that became the World Wide Web. In his proposal [2], he observes that hypertext systems from the 1980's failed to gain traction because they attempted to justify themselves on their own data. He described a simple but effective architecture for exposing existing data from file systems and databases as HTML documents that link to other documents using URIs. This architecture is still used today to present documents that link to documents in which form the World Wide Web.

Despite the success of the World Wide Web, Berners-Lee was not completely satisfied. He also wanted to make raw data itself Web-like, not just the documents that present data. The first step of these visions was called the Semantic Web [4], which was an AI textbook vision of a world where intelligent agents would understand data on the Web to do every day tasks on our behalf. As admitted by Berners-Lee and his co-author Hendler, there was much hype but limited success scaling ideas, which are characteristics of an AI winter. Hendler self-critically asked: "Where are all the intelligent agents?" [19]. By 2006 [29], Berners-Lee had come to the conclusion that there had been too much emphasis on deep ontologies and not enough data.

Thus Berners-Lee returned to the grass roots of the Web: the Web developers. He described a simple protocol for publishing raw data on the Web [3]. The protocol makes use of standards, namely URIs as global identifiers, HTTP as a transport layer and RDF as a data format, according to the following principles:

- use URI to identify resources (i.e. anything that might be referred to in data),
- use HTTP URIs to identify resources so we can look them up (using the HTTP GET verb),
- when a URI is looked up, return data about the resource using the standards (RDF),
- include URIs in the data, so they can also be looked up.

Data published according to the above protocol is called *Linked Data*. An HTTP URI that returns data when it is looked up is a *dereferenceable* URI. All URIs that appear in this paper are dereferenceable, so are part of this rapidly growing Web of Linked Data [18].

The Linked Data protocol is one example of a *RESTful* protocol [14]. A RESTful protocol runs directly on the HTTP protocol using the HTTP verbs (including GET, PUT and DELETE) which are suited to services of publishing data. Many data protocols such as the Twitter API[1], Facebook Open Graph protocol[2] and the Google Data API[3] are RESTful, and with some creativity they can be broadly interpreted as Linked Data. Hence, like the Web of hypertext, the Web of Linked Data does not need to justify itself solely on its own data.

[1] Twitter API: `https://dev.twitter.com/docs/api/1.1`
[2] Facebook Open Graph protocol: `https://developers.facebook.com/docs/opengraph/`
[3] Google Data API: `https://developers.google.com/gdata/`

2.2 An Architecture for Consuming Linked Data

Data owners may want to publish their data as Linked Data. Publishing Linked Data is no more difficult than building a traditional Web page. The developer should provide an RDF view of a dataset rather than an HTML view [6]. Data from diverse sources such as Wikipedia [7] and government data bases [30] can be lifted to the Web of Linked Data.

Data consumers may not own data, but have a data centric service to deliver. Data consumers can consume data from many Linked Data sources then exploit links between datasets. Consuming Linked Data is the main focus in our work. In Figure 1 we describe a simple architecture for an application that consumes Linked Data; the architecture is an extension of the traditional Web architecture.

Figure 1: A simple but effective architecture for an application that consumes Linked Data.

At the heart of our application is a *triple store* which replaces the more traditional relational database. A triple store is optimized for storing RDF data in subject-property-object form corresponding to a labelled edge in a graph. There are several commercial grade triple stores including Sesame, Virtuoso and 4store [9, 13, 16], which can operate on scales of billions of triples, i.e. enough for almost any Web application.

The front end of our application follows the traditional Web architecture pattern, where Web pages are generated from a database using scripts. The only difference is that our application uses SPARQL Query instead of SQL to read from the triple store. The syntax of SPARQL is similar to SQL, hence it is easy for an experienced Web developer to develop the front end. Most popular Web development frameworks, such as Ruby on Rails, have been extended to support SPARQL. The main reason that SQL is replaced with SPARQL is that a triple store typically stores data from heterogeneous data sources. Heterogeneous data sources are difficult to combine and query using tables with relational schema; whereas combining and querying graph models is more straightforward. Thus links between datasets can be more easily exploited in queries.

The key novel feature of an application that consumes Linked Data is the back end. At the back end, background processes can crawl the Web of Linked Data to discover new and relevant Linked Data sources. Back end processes also keep local Linked Data up-to-date. For example, data from a news feed may change hourly and changes are made to wikipedia several times every second. To consume data relevant to the application, the background processes should be programmable. This work focuses in particular on high level programming languages that can be used to define the back end of an application. The background processes must be able to dereference Linked Data and update data in the triple store, as well as make decisions based on query results.

2.3 A Low Level Approach to Consuming Linked Data

The back end of the application that consumes Linked Data should keep track of every URI accessed through the HTTP protocol. The HTTP header of an HTTP response contains information that can be used for discovering and maintaining Linked Data about a given URI. We describe the dereferencing of URIs at a low level, to be concrete about what a high level language should abstract.

Consider an illustrative example of dereferencing a URI. If we perform an HTTP GET on the URI *dbpedia:Kazakhstan* with an HTTP header indicating that we accept the mime type `text/n3`, then we

get a *303 See Other response*.[4] A *303 See Other response* means that you can get data of the serialisation type you requested at another location. If we now perform an HTTP GET request at the URI indicated by the *303 See Other response* (`http://dbpedia.org/data/Kazakhstan.n3`), we get an HTTP 200 OK response including the following headers:

```
GET /data/Kazakhstan.n3 HTTP/1.1      HTTP/1.1 200 OK
Host: dbpedia.org                     Date: Tue, 26 Mar 2013 15:39:49 GMT
Accept: text/n3                       Content-Type: text/n3
```

From the response header (above right) we can tell that this URI successfully returned some RDF using the `n3` serialisation format. However, although the data was obtained from the second URI, the resource represented by *dbpedia:Kazakhstan* is described in the data. Information such as the date and time the URI was accessed and caching policies can optimise algorithms that periodically dereference a URI. For example, the dates and times a URI had been accessed can be used to calculate when a process should revisit the URI next to check whether it has changed, thus keeping local data up-to-date.

The *303 See Other response* is one of several ways Linked Data may be published using a RESTful protocol [18]. Furthermore, some systems such as the Virtuoso triple store [13] use wrappers to extract RDF data from other data sources, such as the Google Data API or the Twitter API. Wrappers (such as Virtuoso's Twitter cartridge) broaden the Web of Linked Data by making URIs such as *twitter:kbtu_university* dereferenceable. The programmer of a script that consumes Linked Data should not worry about details such as wrappers or the serialisation formats. Unfortunately, existing libraries for popular programming languages [1, 26] are at a low level of abstraction. We propose that a higher level language can hide the above details in a compiler that tries automatically to dereference URIs.

2.4 A High Level Approach to Consuming Linked Data

Here we consider languages that consume Linked Data at a higher level of abstraction. The only essential piece of information should be the URI that is dereferenced and the data that is extracted from dereferencing a URI. Other features of a high level language include control flow and queries to decide what other URIs to dereference.

Consider the following script (based loosely on SPARQL [17]). The keyword `select` binds a term bound to variable `$x`. The `from named` keyword indicates that we are querying data from the URI indicated. The `where` keyword indicates that we would like to match a given triple pattern. The second `from named` keyword indicates that data from the URI bound to `$x` should also be dereferenced and loaded into the local triple store.

```
select $x
from named dbpedia:Kazakhstan
where
   graph dbpedia:Kazakhstan {dbpedia:Kazakhstan dbp:capital $x}
from named $x
```

The above script should first ensure that data representing *dbpedia:Kazakhstan* is stored in the named graph also named *dbpedia:Kazakhstan* in the local triple store. If the URI has not been accessed before, then the URI is dereferenced. If the URI is dereferenced successfully, then the RDF data is stored in a named graph in the local triple store. This means that if we query from the URI again, then we can

[4] Reproducible using: `curl -v -I -H "Accept:text/n3" http://dbpedia.org/resource/Kazakhstan`

read directly from the local named graph. This gives our applications a local view of the Web of Linked Data. Note that if the URI is not dereferenced successfully, then this information can be recorded to avoid future attempts to dereference the URI.

Secondly, the query indicated in the `where` clause should be executed. The query consists of a named graph that is the graph dereferenced, the subject is the resource *dbpedia:Kazakhstan*, the property is *dbp:capital* and the object is not bound. This query should find exactly one binding for `$x` to proceed. The query proceeds by discovering the resource *dbpedia:Astana*, and substituting this URI for `$x` everywhere in the script. After the substitution, the final line should obtain data about the URI *dbpedia:Astana*, dereferencing the URI and loading the data into the triple store where necessary, as described above.

The above script abstracts away several HTTP GET requests and possibly some redirects and mappings between formats. It also encapsulates details where the following SPARQL Query is sent to the local SPARQL endpoint.

```
select $x from named dbpedia:Kazakhstan where
    { graph dbpedia:Kazakhstan {dbpedia:Kazakhstan dbp:capital $x} } limit 1
```

At a lower level of abstraction, the above query would return a results document indicating that `x` has one mapping. Another programming language would extract the mapping from the results document, and use it in some code that dereferences the URI. Just doing this simple task in Java for example takes several lines of code. The Java program also involves treating parts of the code, such as the above query as a raw string of characters, which means that even basic errors parsing the syntax cannot be checked at compile time.

We argue that using the high level script presented at the beginning of this section is simpler than using a general purpose language that is concerned about details of HTTP GET requests, constructs queries from strings and extracts variable bindings from query results. Furthermore, it is worth noting that the syntax of the script does not significantly depart from the syntax of the SPARQL recommendations. In this way, we explore the idea of a domain specific scripting language for background processes of an application that consumes Linked Data.

3 Appropriate Types for Linked Data

3.1 Simple Types in W3C Recommendations

The W3C recommendations do not explicitly introduce a type system for Linked Data. However, there are some ideas in the RDF Schema [8] and OWL [20] recommendations that can be used as a basis of a type system. Here we identify one of the simplest notion of a type, and justify the choice with respect to recommendations. The chosen notion of a type fits with types in Facebook's Open Graph.

Simple Datatypes. The only common component of the W3C recommendations in this section is the notion of a simplej datatype. Simple datatypes are specified as part of the XML Schema recommendation [5]. A type system for Linked Data should be aware of the simple datatypes that most commonly appear, in particular *xsd:string*, *xsd:integer*, *xsd:decimal* and *xsd:dateTime*. All these types draw from disjoint lexical spaces, except *xsd:integer* is a subtype of *xsd:decimal*. Note that we assume that *xsd:decimal*, *xsd:float* and *xsd:double* are different lexical representations of a greatest numeric type.

In the XML Schema recommendation, there is a simple datatype *xsd:anyURI*. This type is rarely actively used in ontologies – the ontology for DBpedia only uses this type once as the range of the property *dbp:review*. However, it unambiguously refers to any URI and nothing else, unlike *rdfs:Resource* and *owl:Thing* which, depending on the interpretation, may refer to more than just URIs or a subset of URIs. In this way, our type system is based on unambiguous simple datatypes, that frequently appear in datasets, such as DBpedia [7].

Resource Description Framework. A URI is successfully dereferenced when we get a document from which we can extract RDF data [23]. The basic unit of RDF is the *triple*. An RDF triple consists of a subject, a property and an object. The subject, property and object may be URIs, and the object may also be a simple datatype. Most triple stores support *quadruples*, where a fourth URI represents either the *named graph* [10] or the *context* from where the triple was obtained.

Note that the RDF recommendation allows nodes with a local identifier, called a blank node, in place of a URI. Blank nodes are frequently debated in the community [25], due to several problems they introduce. Firstly, deciding the equality of graphs with blank nodes is an NP-complete problem; and more seriously, when data with blank nodes is consumed more than once each time the blank node is treated as a new blank node. This can cause many unnecessary copies of the same data to be created, when Linked Data is gathered. We assume that our system assigns a new URI to each blank node in data consumed, hence do not introduce blank nodes into the local data model.

It is also important to note that the RDF specification has a vocabulary of URIs that have a distinguished role. Most notably, the property *rdf:type* is used to indicate a URI that classifies the resource. For example, the triple *dbpedia:Kazakhstan rdf:type dbp:Country* classifies the resource *dbpedia:Kazakhstan* as a *dbp:Country*. Although the word "type" is part of the URI for the property, we consider this triple to be part of the data format rather than part of our type system. In RDF, the word "type" is used in the AI sense of a semantic network [31], rather than in a type theoretic sense. Since these "types" can be changed like any other data, we make the design decision not to include them in our type system, because a type system is used for static analysis.

RDF Schema. The RDF Schema recommendation [8] provides a core vocabulary for classifying resources using RDF. From this vocabulary we borrow only the top level class *rdfs:Resource* and the property *rdfs:range*. All URIs are considered to identify resources, hence we can equate *rdfs:Resource* and *xsd:anyURI*. We can also define property types for URIs that are used in the property position of a triple. A property type restricts the type of term that can be used in the object position of a triple. For example, according to the DBpedia ontology, the property *dbp:populationDensity* has a range *xsd:decimal*. Thus our type system should accept that *dbpedia:Kazakhstan dbp:populationDensity* 5.94 is well typed. However, the type system should reject a triple with object `"5.94"` which is a term of type *xsd:string*. We use the notation *range*(*xsd:decimal*) for URIs of this type.

The RDF Schema *rdfs:domain* of a property is redundant for our type system because only URIs can appear as the subject of a triple, and all URIs are of type *xsd:anyURI*. Note that properties are resources because they may appear in data. For example, the triple *dbp:populationDensity rdfs:label* `"population density (/sqkm)"` provides a (human) readable description of the property.

Web Ontology Language. The Web Ontology Language (OWL) [20] is mostly concerned with classifying resources, which is not part of our type system. The OWL classes that are related to our type system are *owl:ObjectProperty*, *owl:DataTypeProperty* and *owl:Thing*. An *owl:ObjectProperty* is

a property where restricts corresponding objects must be URIs, i.e. the type *range*(*xsd:anyURI*) in our type system. An *owl:DataTypeProperty* is a property with one of the simple datatypes as its value. In OWL [20], *owl:Thing* represents resources that are neither properties nor classes. We decide to equate *owl:Thing* with *xsd:anyURI* in our type system. This way we unify *xsd:anyURI* , *rdfs:Resource* and *owl:Thing* as the top level of all resources.

We do not consider any further features of OWL to be part of our type system.

SPARQL Protocol and RDF Query Language. The SPARQL suite of recommendations makes reference only to simple types. SPARQL Query [17] specifies the types of basic operations that are used to filter queries. For example, a regular expression can only apply to a string, and the sum to two integers is an integer. SPARQL Query treats all URIs as being of type URI. We also adopt this approach.

Open Graph Protocol. Facebook's Open Graph protocol uses an approach to Linked Data called microformats, where bits of RDF data are embedded into HTML documents. Microformats help machines to understand the content of the Web pages, which can be used to drive powerful searches. The Open Graph documentation states the following: "properties have 'types' which determine the format of their values."[5] In the terminology of the Open Graph documentation, the value of a property is the object of an RDF triple. The documentation explicitly includes the simple data types *xsd:string*, *xsd:integer*, etc, as types permitted to appear in the object position. This corresponds to the notion of type throughout this section.

3.2 Local Type Checking for Dereferenced Linked Data

We design our system such that, when a URI is dereferenced, only well typed queries are loaded into the triple store. This means that we can guarantee that all triples in the triple store are well typed.

Suppose that after dereferencing the URI *dbpedia:Kazakhstan* we obtain the following two triples:

dbpedia:Kazakhstan dbp:demonym `"Kazakhstani"@en` .
dbpedia:Kazakhstan dbp:demonym dbpedia:Kazakhstani .

Suppose also that the property *dbp:demonym* has the type *range*(*xsd:string*). The first triple is well typed, hence it is loaded into the store in the named graph *dbpedia:Kazakhstan* . However, the second triple is not well typed, hence would be ignored. No knowledge of other triples loaded into the store is required, i.e. our type system is local.

Since only well typed triples are loaded into the local triple store, scripts that use data in the local triple store can rely on type properties. Thus, for example, if a script consumes the object of any triple with *dbp:demonym* as the property, then the script can assume that the term returned will be a string. This allows some static analysis to be performed by a type system for scripts. This observation is the basis of the type system in the next section.

4 A Typed Scripting Language for Linked Data Consumers

4.1 Syntax

We introduce a syntax for a typed high level scripting language that is used to consume Linked Data.

[5] `https://developers.facebook.com/docs/opengraph/property-types/` accessed on 27 March 2013.

The Syntax of Types. A type is either a simple datatype, or it is a property that allows a simple datatype as its range, as follows:

$$\textit{datatype} ::= \textit{xsd:anyURI} \mid \textit{xsd:string} \mid \textit{xsd:decimal} \mid \textit{xsd:dateTime} \mid \textit{xsd:integer}$$

$$\textit{type} ::= \textit{datatype} \mid \textit{range}(\textit{datatype}) \qquad \textit{variable} ::= \texttt{\$x} \mid \texttt{\$y} \mid \ldots \qquad \Gamma ::= \epsilon \mid \texttt{\$x}\colon \textit{type}, \Gamma$$

If a URI with a property type is used in the property position of a triple, then the object of that triple can only take the value indicated by the property type. Property types are assigned to URIs using the finite partial function $O(\cdot)$ from URIs to property types. This partial function can be derived from ontologies or inferred from data.

Type environments are defined by lists of assignments of variables to types. As in popular scripting languages such as Perl and PHP, variables begin with a dollar sign.

The Syntax of Terms and Expressions. Terms are used to construct RDF triples. Terms can be URIs, variables or literals of a simple datatype, each of which is drawn from a disjoint pool of lexemes.

$$\textit{uri} ::= \textit{dbpedia:URI} \mid \ldots \qquad \textit{integer} ::= 99 \mid \ldots \qquad \textit{decimal} ::= 99.9 \mid 0.999e2 \ldots$$

$$\textit{string} ::= \texttt{"WWV2013"} \mid \texttt{"workshop"@en-gb} \ldots \qquad \textit{dateTime} ::= \texttt{2013-06-6T13:00:00+01:00} \mid \ldots$$

$$\textit{language-range} ::= \texttt{*} \mid \texttt{en} \mid \texttt{en-gb} \mid \ldots \qquad \textit{term} ::= \textit{variable} \mid \textit{uri} \mid \textit{string} \mid \textit{integer} \mid \textit{decimal} \mid \textit{dateTime}$$

$$\textit{regex} ::= \texttt{WWV.*} \mid \ldots \qquad \textit{expr} ::= \textit{term} \mid \texttt{now} \mid \texttt{str}(\textit{expr}) \mid \texttt{abs}(\textit{expr}) \mid \textit{expr} + \textit{expr} \mid \textit{expr} - \textit{expr} \mid \ldots$$

Notice that strings may have a language tag, as defined by RFC4646 [27]. A language tag can be matched by a simple pattern, called a language range, where * matches any language tag (e.g., `en` matches any dialect of English). Regular expressions over strings conform to the XPath recommendation [24].

Expressions are formed by applying unary and binary functions over terms. The SPARQL Query recommendation [17] defines several standard terms including `str` which maps any term to a string, and `abs` which takes the absolute value of a number. The expression `now` represents the current date and time. The vocabulary of functions may be extended as required.

The Syntax of Scripts. Scripts are formed from boolean expressions, data and queries. They define a sequence of operations that use queries to determine what URIs to dereference.

$$\begin{aligned}
\textit{boolean} ::=\ & \textit{boolean} \texttt{ || } \textit{boolean} \mid \textit{boolean} \texttt{ \&\& } \textit{boolean} \mid \neg \textit{boolean} \\
\mid\ & \texttt{regex}(\textit{expr}, \textit{regex}) \mid \texttt{langMatches}(\textit{expr}, \textit{language-range}) \mid \textit{expr} = \textit{expr} \mid \textit{expr} < \textit{expr} \mid \ldots \\
\textit{triples} ::=\ & \textit{term term term .} \mid \textit{triples triples} \qquad \textit{data} ::= \texttt{graph}\, \textit{term}\, \{\textit{triples}\} \mid \textit{data data} \\
\textit{query} ::=\ & \textit{data} \mid \textit{boolean} \mid \textit{query query} \mid \textit{query}\, \texttt{union}\, \textit{query}
\end{aligned}$$

$$\begin{aligned}
\textit{script} ::=\ & \texttt{where}\ \textit{query script} && \textit{Satisfy a query pattern before continuing.} \\
\mid\ & \texttt{from named}\ \textit{term script} && \textit{Dereference a URI and load it into the local triple store.} \\
\mid\ & \texttt{select}\ \textit{variable}\colon \textit{type script} && \textit{Select a binding for a variable to enable progress.} \\
\mid\ & \texttt{do}\ \textit{script} && \textit{Iteratively execute the script using separate data.} \\
\mid\ & \texttt{skip} && \textit{Successfully terminate.}
\end{aligned}$$

Data is represented as quadruples of terms, which always indicate the named graph. In SPARQL, the `graph` and `from named` keywords work in tandem. The `from named` keyword makes data from a named graph available, whilst keeping track of the context. The keyword `graph` allows the query to directly refer to the context. This contrasts to the `from` keyword in SPARQL which fetches data without keeping the context. We extend the meaning of the `from named` keyword so that the URI is dereferenced and makes the data available from that point onwards in the context of the URI that is referenced.

Boolean expressions are called filters in the terminology of SPARQL. We drop the keyword `filter`, because the syntax is unambiguous without it. Filters can be used to match a string with a regular expression or language tag, or compare two expressions of the same type.

Queries are constructed from filters and basic graph patterns indicated by the keyword `where`. The basic graph pattern should be matched using data in the local triple store. The basic graph pattern may contain variables. Variables in a script must be bound using the `select` keyword. In this scripting language, the `select` keyword acts like an existential quantifier. The variables bound by `select` are annotated with a type. However these type annotations may be omitted by the programmer, since they can be algorithmically inferred using the type system.

4.2 The Type System

For existing implementations of SPARQL, a query that violates types silently fails, returning an empty result. However, a type error is generally caused by an oversight by the programmer. It would be helpful if a type error is provided, indicating that the query has been designed such that the types guarantee that no result will ever be returned. This is the purpose for which we introduce the type system presented here.

Subtypes. We define a subtype relation, that defines when one type can be treated as another type. The system indicates that *xsd:integer* is a subtype of *xsd:decimal*. It also defines property types to be contravariant, i.e. they reverse the direction of subtyping. In particular, if a property permits decimal numbers in the object position, then it also permits integers in the object position.

$$\frac{}{\vdash \mathit{xsd{:}integer} \leq \mathit{xsd{:}decimal}} \qquad \frac{}{\vdash \mathit{type} \leq \mathit{type}}$$

$$\frac{\vdash \mathit{datatype}_1 \leq \mathit{datatype}_2}{\vdash \mathit{range}(\mathit{datatype}_2) \leq \mathit{range}(\mathit{datatype}_1)} \qquad \frac{}{\vdash \mathit{range}(\mathit{datatype}) \leq \mathit{xsd{:}anyURI}}$$

The subtype relations between types that can be assigned to URIs are summarised in Figure 2.

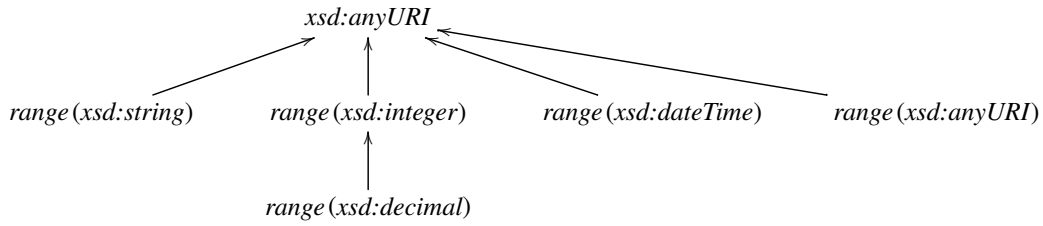

Figure 2: Subtype relations between types that can be assigned to URIs.

The type system for terms and expressions. Types for terms assign types to lexical tokens and assign types to properties using the partial function $O(\cdot)$ from URIs to types. Types for expressions ensure that operations are only applied to resources of the correct type.

$$\frac{\vdash type_0 \leq type_1}{\Gamma, \texttt{\$x}: type_0 \vdash \texttt{\$x}: type_1} \qquad \frac{\vdash O(uri) \leq type}{\Gamma \vdash uri: type} \qquad \frac{\vdash xsd{:}integer \leq datatype}{\Gamma \vdash integer: datatype}$$

$$\frac{}{\Gamma \vdash decimal: xsd{:}decimal} \qquad \frac{}{\Gamma \vdash string: xsd{:}string} \qquad \frac{}{\Gamma \vdash dateTime: xsd{:}dateTime}$$

$$\frac{}{\Gamma \vdash \texttt{now}: xsd{:}dateTime} \qquad \frac{\Gamma \vdash expr_1: datatype \quad \Gamma \vdash expr_2: datatype \quad \vdash datatype \leq xsd{:}decimal}{\Gamma \vdash expr_1 + expr_2: datatype}$$

$$\frac{\Gamma \vdash expr: datatype}{\Gamma \vdash \texttt{str}(expr): xsd{:}string} \qquad \frac{\Gamma \vdash expr: datatype \quad \vdash datatype \leq xsd{:}decimal}{\Gamma \vdash \texttt{abs}(expr): datatype}$$

The above types can easily be extended to cover all functions in the SPARQL recommendation, such as `seconds` which maps an *xsd:dateTime* to a *xsd:decimal*. Our examples include the custom function `haversine` which maps four expressions of type *xsd:decimal* to one *xsd:decimal*.

The type system for filters follows a similar pattern to expressions. For example, the type system ensures that only terms of the same type can be compared.

$$\frac{\Gamma \vdash expr: xsd{:}string}{\Gamma \vdash \texttt{regex}(expr, regex)} \qquad \frac{\Gamma \vdash expr: xsd{:}string}{\Gamma \vdash \texttt{langMatches}(expr, language\text{-}range)}$$

$$\frac{\Gamma \vdash expr_1: datatype \quad \Gamma \vdash expr_2: datatype}{\Gamma \vdash expr_1 = expr_2} \qquad \frac{\Gamma \vdash expr_1: datatype \quad \Gamma \vdash expr_2: datatype}{\Gamma \vdash expr_1 < expr_2}$$

$$\frac{\Gamma \vdash boolean_0 \quad \Gamma \vdash boolean_1}{\Gamma \vdash boolean_0 \;\&\&\; boolean_1} \qquad \frac{\Gamma \vdash boolean_0 \quad \Gamma \vdash boolean_1}{\Gamma \vdash boolean_0 \;||\; boolean_1} \qquad \frac{\Gamma \vdash boolean}{\Gamma \vdash !boolean}$$

The type system for data. The types for terms are used to restrict the subject of triples to URIs, and the object to the type prescribed by the property. For quadruples, the named graph is always a URI, as prescribed by the type system.

$$\frac{\Gamma \vdash term_1: xsd{:}anyURI \quad \Gamma \vdash term_2: range(datatype) \quad \Gamma \vdash term_3: datatype}{\Gamma \vdash term_1 \;\; term_2 \;\; term_3}$$

$$\frac{\vdash triples_1 \quad \vdash triples_2}{\vdash triples_1 \; triples_2} \qquad \frac{\Gamma \vdash term_0: xsd{:}anyURI \quad \Gamma \vdash triples}{\Gamma \vdash \texttt{graph}\, term_0 \{triples\}}$$

$$\frac{\Gamma \vdash uri: range(datatype)}{\Gamma \vdash uri\ rdfs{:}range\ datatype\ .} \qquad \frac{\Gamma \vdash uri: range(xsd{:}anyURI)}{\Gamma \vdash uri\ rdf{:}type\ owl{:}ObjectProperty\ .}$$

We include special type rules for two particular forms of triples. The first, from the RDF Schema vocabulary [8], allows the range of a property to be explicitly prescribed as a datatype. The second, from

the OWL vocabulary [20], prescribes when the range of a property is a URI. These two rules are useful for type inference when we infer the minimum partial function $O(\cdot)$ that allows part of a dataset to be typed.

The Type System for scripts. Scripts can be type checked using our type system. The rule for the `from named` keyword checks that the term to dereference is a URI. The rule for the `select` makes use of an assumption in the type environment to ensure that the variable is used consistently in the rest of the script, where the variable is bound.

$$\frac{\Gamma \vdash \mathit{query}_1 \quad \Gamma \vdash \mathit{query}_2}{\Gamma \vdash \mathit{query}_1\ \mathit{query}_2} \qquad \frac{\Gamma \vdash \mathit{query}_1 \quad \Gamma \vdash \mathit{query}_2}{\Gamma \vdash \mathit{query}_1\ \texttt{union}\ \mathit{query}_2} \qquad \frac{\Gamma \vdash \mathit{query} \quad \Gamma \vdash \mathit{script}}{\Gamma \vdash \texttt{where}\ \mathit{query}\ \mathit{script}}$$

$$\frac{\Gamma \vdash \mathit{term} : \mathit{xsd{:}anyURI} \quad \Gamma \vdash \mathit{script}}{\Gamma \vdash \texttt{from named}\ \mathit{term}\ \mathit{script}} \qquad \frac{\Gamma, \texttt{\$x} : \mathit{type} \vdash \mathit{script}}{\Gamma \vdash \texttt{select \$x}\ : \mathit{type}\ \mathit{script}} \qquad \frac{\Gamma \vdash \mathit{script}}{\Gamma \vdash \texttt{do}\ \mathit{script}} \qquad \Gamma \vdash \texttt{skip}$$

Note that if a script is well typed with an empty type environment, then all the variables must be bound using the rule for the `select` quantifier. Furthermore, since the `select` quantifiers does not appear in data, no variables can appear in well typed data. We assume that scripts and data are executable only if they are well typed with respect to an empty type environment.

4.3 Examples of Well Typed Scripts

We consider some well typed scripts, and suggest some errors that the type check can avoid.

The following script is well typed. The script finds resources in any named graph that have a label in the Russian language. It then dereferences the resources. The script is iterated as many times as the implementation feels necessary, without revisiting data.

```
do  select $g: xsd:anyURI, $x: xsd:anyURI, $y: xsd:string
    where
       graph $g {$x rdfs:label $y}
       langMatches($y, ru)
    from named $x
```

Note that if we use the variable `$y` instead of `$x` in the `from named` clause, then the script could not be typed. The variable `$y` would need to be both a string and a URI, fact which is not possible without explicit coercions. We assume that $O(\mathit{rdfs{:}label}) = \mathit{range}(\mathit{xsd{:}string})$.

The following well typed script looks in two named graphs. In the named graph *dbpedia:Kazakhstan* it looks for properties with *dbpedia:Kazakhstan* as the object, and in the named graph *dbp:* it looks for properties that have either a label or comment that contains the string `"location"`.

```
select $p: range(xsd:anyURI), $y: xsd:string, $z: xsd:string
where
   {graph dbp: {$p rdfs:label $y} union graph dbp: {$p rdfs:comment $y}}
   graph dbpedia:Kazakhstan {$z $p dbpedia:Kazakhstan}
   regex($y, location) && langMatches($y, en)
from named $p
```

We must assume that $O(\textit{rdfs:comment}) = \textit{range}(\textit{xsd:string})$. If we assume otherwise, then the above query is not well typed. Note also that property `$p` must be of type $\textit{range}(\textit{xsd:anyURI})$. We cannot assign it a more general type such as *xsd:anyURI* , although we can use it as a resource.

```
from named dbpedia:Almaty
select $almalat: xsd:decimal, $almalong: xsd:decimal
where
   graph dbpedia:Almaty {dbpedia:Almaty geo:lat $almalat}
   graph dbpedia:Almaty {dbpedia:Almaty geo:long $almalong}
from named dbpedia:Kazakhstan
do select $loc: xsd:anyURI
   where
      graph dbpedia:Kazakhstan {$loc dbp:location dbpedia:Kazakhstan}
   from named $loc
   select $lat: xsd:decimal, $long: xsd:decimal
   where
      graph $loc {$loc geo:lat $lat}
      graph $loc {$loc geo:long $long}
      haversine($lat, $long, $almalat, $almalong) < 100
   do select $person: xsd:anyURI
      where
         graph $loc {$person dbp:birthPlace $loc}
      from named $person
```

Figure 3: Get data about people born in places in Kazakhstan less than 100km from Almaty.

Finally, consider the substantial example of Figure 3. We assume that the function `haversine` calculates the distance (in km) between two points identified by their latitude and longitude. The script pulls in data about Kazakhstan, Almaty and places located in Kazakhstan. It then uses this data to pull in more data about people born in places less than 100km from Almaty.

4.4 Operational Semantics for the System

In related work, we extensively study the operational semantics of languages for Linked Data that are related to the language proposed in this work [11, 12, 21, 22]. In the remaining space, we briefly sketch the operational semantics for our scripting language.

Systems are data and scripts composed in parallel by using the $\|$ operator. The main rules are the rules for dereferencing URIs, for selecting bindings, and for interactions between a query and data. The rules can be applied in any context.

$$\frac{\vdash \texttt{graph}\, \textit{uri}\, \{\textit{triples}\}}{\texttt{from named}\, \textit{uri}\, \textit{script} \longrightarrow \textit{script} \,\|\, \texttt{graph}\, \textit{uri}\, \{\textit{triples}\}}$$

$$\frac{\vdash \textit{term} : \textit{type}}{\texttt{select \$x}: \textit{type}\, \textit{script} \longrightarrow \textit{script}\left\{^{\textit{term}}/_{\texttt{\$x}}\right\}} \qquad \frac{\textit{data} \leq \textit{query}}{\texttt{where}\, \textit{query}\, \textit{script} \,\|\, \textit{data} \longrightarrow \textit{script} \,\|\, \textit{data}}$$

The rules for dereferencing data involves a dynamic type check of arbitrary data that arrives as a result of dereferencing a URI. The `select` rule also performs a dynamic check to ensure that the term substituted for a variable is of the correct type. The query rule reads data that matches the query pattern (using a preorder $\leq$ over queries), without removing the data.

The following result proves that a well typed term will always reduce to a well typed term. Thus the type system is sound with respect to the operational semantics.

Theorem 4.1. *If* $\vdash$ *system*$_1$ *and system*$_1 \longrightarrow$ *system*$_2$, *then* $\vdash$ *system*$_2$.

Proof. We provide a proof sketch covering the cases for only the three rules above.

Consider the operational rule for `from named`. Assume that both $\vdash$ `from named` *uri script* and $\vdash$ `graph` *uri* {*triples*} hold. By the type rule for `from named`, $\vdash$ *script* must hold. Thus, by the type rule for parallel composition, $\vdash$ *script* $\|$ `graph` *uri* {*triples*}.

Lemma. If $\vdash$ *term*: *type* and `$x`: *type* $\vdash$ *script*, then $\vdash$ *script*$\{^{term}/_{\texttt{\$x}}\}$, by structural induction. □

Consider the operational rule for `select`. Assume that $\vdash$ *term*: *type* and $\vdash$ `select $x`: *type script* hold. By the type rule for `select`, `$x`: *type* $\vdash$ *script* must hold. Hence, by the above lemma, $\vdash$ *script*$\{^{term}/_{\texttt{\$x}}\}$ holds.

Consider the operational rule for `where`. Assume that $\vdash$ `where` *query script* $\|$ *data* holds. By the type rule for parallel composition, $\vdash$ `where` *query script* and $\vdash$ *data* must hold, and, by the type rule for `where`, $\vdash$ *query* and $\vdash$ *script* must hold. Hence $\vdash$ *script* $\|$ *data* holds. □

Further cases and results will be covered in an extended paper. Future work includes implementing an interpreter for the language based on the operational semantics, and developing a minimal type inference algorithm [28] based on the type system.

5 Conclusion

As the scope of the Web of Linked Data rapidly grows, the state of the art for commercial Linked Data solutions is also advancing. The state of the art of triple stores is impressive, allowing efficient execution of queries at scale. Furthermore, the back end of commercial solutions such as Virtuoso [13] and the Information Workbench [15] can extract data from diverse sources. This allows us to take a liberal view of Linked Data that draws from data APIs covering popular services from Twitter, Facebook and Google. Through experience with master students, we also found that developers with experience of a Web development platform such as .NET or Ruby-on-Rails can assemble a front end in a matter of days, simply by shifting their query language from SQL to SPARQL.

Among the novel features of Linked Data is that processes can programmatically crawl the Web of Linked Data to pull data from diverse sources on the Web. The data pulled in forms a local view of the Web of Linked Data that is tailored to a particular application. We found that existing programming environments consisting of a general purpose language and a library obstructed swift development of such processes with many low level details. This exposes the need for a high level language that makes easy scripting background processes that consume Linked Data.

In this work, we introduce a domain specific high level language for consuming Linked Data. Domain specific languages are designed at a level of abstraction that simplifies programming tasks in the domain. In our domain specific language, key operations such as queries are primitive, meaning that basic syntactic checks can be performed. It is also easier to perform static analysis over a domain specific

language. We also take care to design the syntax of the scripting language such that it resembles the SPARQL recommendations [17], to appeal to the target Web developers.

We introduce a simple but effective type system for our language. The type system is based on the fragment of the SPARQL, RDF Schema and OWL recommendations that deals with simple data types. The applications can statically identify simple errors such as attempts to dereference a number, or attempts to match the language tag of a URI. For static type checking of scripts, the data loaded into the system must be dynamically type checked to ensure that properties have the correct literal value or a URI as the object. The dynamic type checks do not impose significant restrictions on the data consumed. Most datasets, including data from DBpedia, conform to this typing pattern. Further weight is added by the Facebook Open Graph protocol which demands typing at exactly the level we deliver.

Acknowledgements. The work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919.

References

[1] David Beckett (2002): *The design and implementation of the Redland RDF application framework*. Computer Networks 39(5), pp. 577–588.

[2] Tim Berners-Lee (1989): *Information management: A proposal*. Available at `http://www.w3.org/History/1989/proposal.html`.

[3] Tim Berners-Lee (2006): *Linked data — design issues*. Available at `http://www.w3.org/DesignIssues/LinkedData.html`.

[4] Tim Berners-Lee, James Hendler & Ora Lassila (2001): *The Semantic Web*. Scientific American 284(5), pp. 28–37.

[5] Paul V. Biron & Ashok Malhotra (2004): *XML Schema part 2: Datatypes Second Edition*. W3C, MIT, Cambridge, MA.

[6] Christian Bizer & Andy Seaborne (2004): *D2RQ-treating non-RDF databases as virtual RDF graphs*. In: Proceedings of the 3rd International Semantic Web Conference (ISWC2004), p. 26, doi:10.1038/npre.2011.5660.1.

[7] Christian Bizer et al. (2009): *DBpedia: A Crystallization Point for the Web of Data*. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), pp. 154–165, doi:DOI: 10.1016/j.websem.2009.07.002.

[8] Dan Brickley & Ramanathan V. Guha (2004): *RDF Vocabulary Description Language 1.0: RDF Schema*. recommendation REC-rdf-schema-20040210, W3C, MIT, Cambridge, MA. Available at `http://www.w3.org/TR/rdf-schema/`.

[9] Jeen Broekstra, Arjohn Kampman & Frank van Harmelen (2002): *Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema*. In: International Semantic Web Conference, pp. 54–68, doi:10.1007/3-540-48005-6_7.

[10] Jeremy J. Carroll, Christian Bizer, Pat Hayes & Patrick Stickler (2005): *Named graphs*. Web Semantics: Science, Services and Agents on the World Wide Web 3(4), pp. 247–267, doi:10.1016/j.websem.2005.09.001.

[11] Gabriel Ciobanu & Ross Horne (2012): *A Provenance Tracking Model for Data Updates*. In: FOCLASA, pp. 31–44, doi:10.4204/EPTCS.91.3.

[12] Mariangiola Dezani, Ross Horne & Vladimiro Sassone (2012): *Tracing where and who provenance in Linked Data: a calculus*. Theoretical Computer Science 464, pp. 113–129, doi:10.1016/j.tcs.2012.06.020.

[13] Orri Erling & Ivan Mikhailov (2007): *RDF Support in the Virtuoso DBMS*. In: CSSW, pp. 59–68, doi:10.1007/978-3-642-02184-8_2.

[14] Roy T. Fielding & Richard N. Taylor (2002): *Principled design of the modern Web architecture*. *ACM Transactions on Internet Technology* 2(2), pp. 115–150.

[15] Peter Haase, Michael Schmidt Christian Hütter & Andreas Schwarte (2012): *The Information Workbench as a Self-Service Platform for Linked Data Applications*. In: *WWW 2012. Lyon, France.*

[16] Steve Harris, Nick Lamb & Nigel Shadbolt (2009): *4store: The design and implementation of a clustered RDF store*. In: *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pp. 94–109.

[17] Steve Harris & Andy Seaborne (2013): *SPARQL 1.1 Query Language*. Recommendation REC-sparql11-query-20130321, W3C, MIT, MA. Available at `http://www.w3.org/TR/sparql11-query/`.

[18] Tom Heath & Christian Bizer (2011): *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers. Available at `http://dx.doi.org/10.2200/S00334ED1V01Y201102WBE001`.

[19] James A. Hendler (2007): *Where Are All the Intelligent Agents?* *IEEE Intelligent Systems* 22(3), pp. 2–3. Available at `http://doi.ieeecomputersociety.org/10.1109/MIS.2007.62`.

[20] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider & Sebastian Rudolph (2012): *OWL 2 Web Ontology Language Primer (Second Edition)*. Recommentation REC-owl2-primer-20121211, W3C. Available at `www.w3.org/TR/owl2-primer/`.

[21] Ross Horne & Vladimiro Sassone (2011): *A Verified Algebra for Linked Data*. In: *FOCLASA*, pp. 20–33, doi:10.4204/EPTCS.58.2.

[22] Ross Horne, Vladimiro Sassone & Nicholas Gibbins (2012): *Operational Semantics for SPARQL Update*. In: *The Semantic Web, Lecture Notes in Computer Science* 7185, Springer, pp. 242–257, doi:10.1007/978-3-642-29923-0_16.

[23] Graham Klyne & Jeremy Carroll (2004): *Resource Description Framework: Concepts and Abstract Syntax*. recommendation REC-rdf-concepts-20040210, W3C, MIT, MA. Available at `http://www.w3.org/TR/rdf-concepts/`.

[24] Ashok Malhotra, Jim Melton, Norman Walsh & Michael Kay (2010): *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)*. recommendation REC-xpath-functions-20101214, W3C, MIT, MA. Available at `www.w3.org/TR/xpath-functions/`.

[25] Alejandro Mallea, Marcelo Arenas, Aidan Hogan & Axel Polleres (2011): *On Blank Nodes*. In: *International Semantic Web Conference (1)*, pp. 421–437, doi:10.1007/978-3-642-25073-6_27.

[26] Brian McBride (2002): *Jena: A Semantic Web Toolkit*. *IEEE Internet Computing* 6(6), pp. 55–59, doi:10.1109/MIC.2002.1067737.

[27] A. Phillips & M. Davis (2006): *Tags for Identifying Languages*. Best Current Practice RFC4646, IETF. Available at `http://www.ietf.org/rfc/rfc4646`.

[28] Michael I Schwartzbach (1991): *Type inference with inequalities*. In: *Proceedings of the international joint conference on theory and practice of software development on Colloquium on trees in algebra and programming (CAAP '91): vol 1*, TAPSOFT '91, Springer, pp. 441–455.

[29] Nigel Shadbolt, Wendy Hall & Tim Berners-Lee (2006): *The Semantic Web revisited*. *IEEE intelligent systems* 21(3), pp. 96–101, doi:10.1109/MIS.2006.62.

[30] Nigel Shadbolt, Kieron O'Hara, Tim Berners-Lee, Nicholas Gibbins, Hugh Glaser, Wendy Hall & m. c. schraefel (2012): *Linked Open Government Data: Lessons from Data.gov.uk*. *IEEE Intelligent Systems* 27(3), pp. 16–24, doi:10.1109/MIS.2012.23.

[31] John F. Sowa (2000): *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA.

Amending Choreographies

Ivan Lanese
Focus Team, University of Bologna/INRIA, Italy
lanese@cs.unibo.it

Fabrizio Montesi
IT University of Copenhagen, Denmark
fmontesi@itu.dk

Gianluigi Zavattaro
Focus Team, University of Bologna/INRIA, Italy
zavattar@cs.unibo.it

Choreographies are global descriptions of system behaviors, from which the local behavior of each endpoint entity can be obtained automatically through projection. To guarantee that its projection is correct, i.e. it has the same behaviors of the original choreography, a choreography usually has to respect some coherency conditions. This restricts the set of choreographies that can be projected.

In this paper, we present a transformation for amending choreographies that do not respect common syntactic conditions for projection correctness. Specifically, our transformation automatically reduces the amount of concurrency, and it infers and adds hidden communications that make the resulting choreography respect the desired conditions, while preserving its behavior.

1 Introduction

Choreography-based programming is a powerful paradigm where the programmer defines the communication behavior of a system from a global viewpoint, instead of separately specifying the behavior of each endpoint entity. Then, the local behavior of each endpoint can be automatically generated through a notion of *projection* [3, 6, 7, 1, 4]. A projection procedure is usually a homomorphism from choreographies to endpoint code. However, projections of some choreographies may lead to undesirable behavior. To characterize the class of choreographies that can be correctly projected some syntactic conditions have been put forward. Consider the following choreography C:

$$C = \mathtt{a} \to \mathtt{b}{:}o_1;\ \mathtt{c} \to \mathtt{d}{:}o_2$$

Here, $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$ and $\mathtt{d}$ are *participants* and o_1 and o_2 are *operations* (labels for communications). C specifies a system where $\mathtt{a}$ sends to $\mathtt{b}$ a message on operation o_1 ($\mathtt{a} \to \mathtt{b}{:}o_1$), and then (; is sequential composition) $\mathtt{c}$ sends to $\mathtt{d}$ a message on operation o_2 ($\mathtt{c} \to \mathtt{d}{:}o_2$). We can naturally project C onto a system S composed of CCS-like processes annotated with roles [7]:

$$S = [\overline{o_1}]_{\mathtt{a}} \parallel [o_1]_{\mathtt{b}} \parallel [\overline{o_2}]_{\mathtt{c}} \parallel [o_2]_{\mathtt{d}}$$

Here, $[\cdot]_{\mathtt{a}}$ specifies the behavior of participant $\mathtt{a}$, $\parallel$ is parallel composition, $\overline{o_1}$ denotes an output on operation o_1, and o_1 the corresponding input. Unfortunately, the projected system S does not implement C correctly. Indeed, $\mathtt{c}$ could send its message on o_2 to $\mathtt{d}$ before the communication on o_1 between $\mathtt{a}$ and $\mathtt{b}$ has occurred. This is in contrast with the intuitive meaning of the sequential composition operator ; in the choreography, which explicitly models sequentiality between the two communications. In [7], we provide syntactic conditions on choreographies for ensuring that projection will behave correctly. Similar conditions are used in other works (e.g., [3, 6]). Such conditions would reject choreography C

above, by recognizing that it is not *connected*, i.e., the order of communications specified in C cannot be enforced by the projected roles.

In this paper, we present a procedure for automatically transforming, or *amending*, a choreography that is not connected into a behaviorally equivalent one that is connected. Our transformation acts in two ways. In most of the cases, it automatically infers and adds some extra hidden communications. In a few cases instead the problem cannot be solved by adding communications, and the amount of concurrency in the system has to be decreased. For example, we can transform C in the connected choreography C' below:

$$C' = \mathtt{a} \to \mathtt{b}{:}o_1;\ \mathtt{b} \to \mathtt{e}{:}o_3^*;\ \mathtt{e} \to \mathtt{c}{:}o_4^*;\ \mathtt{c} \to \mathtt{d}{:}o_2$$

Here, we have added an extra role e that interacts with b and c to ensure the sequentiality of their respective communications. This is the first kind of transformation discussed above. We refer to Example 2 in Section 3.4 for an example of the second kind.

Our transformation brings several benefits. First, designers could use choreographies by defining only the desired behavior, leaving to our transformation the burden of filling in the necessary details on how the specified orderings should be enforced. Moreover, our solution does not change the standard definition of projection, since we operate only at the level of choreographies. Hence, we retain simplicity in the definition of projection. Finally, our transformation offers an automatic way of ensuring correctness when composing different choreographies developed separately into a single one.

Structure of the paper: Section 2 introduces choreographies and their semantics. Section 3 contains the main contribution of the paper, namely the description and the proof of correctness of the amending techniques. Section 4 applies the developed theory to a well-known example, the two-buyers protocol. Finally, Section 5 discusses related work and future directions. Appendix A and Appendix B describe projected systems and the projection operation, respectively.

2 Choreography semantics

We introduce here our language for modeling choreographies. The set of participants in a choreography, called *roles*, is ranged over by $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$. We also consider two kinds of *operations* for communications: *public operations*, ranged over by o, which represent observable activities of the system, and *private operations*, ranged over by o^*, used for internal synchronization. We use $o^?$ to range over both public and private operations.

Choreographies, ranged over by $C, C', \ldots$, are defined as follows:

$$C ::= \mathtt{a} \to \mathtt{b}{:}o^? \mid \mathbf{1} \mid \mathbf{0} \mid C;C' \mid C \parallel C' \mid C + C'$$

An interaction $\mathtt{a} \to \mathtt{b}{:}o^?$ means that role a sends a message on operation $o^?$ to role b (we assume that $\mathtt{a} \neq \mathtt{b}$). Besides, there are the empty choreography $\mathbf{1}$, the deadlocked choreography $\mathbf{0}$, sequential and parallel composition of choreographies, and nondeterministic choice between choreographies. Deadlocked choreography $\mathbf{0}$ is only used at runtime, and it is not part of the user syntax.

The semantics of choreographies is the smallest labeled transition system (LTS) closed under the rules in Figure 1. Symmetric rules for parallel composition and choice have been omitted. We use σ to range over labels. We have two kinds of labels: label $\mathtt{a} \to \mathtt{b}{:}o^?$ denotes the execution of an interaction $\mathtt{a} \to \mathtt{b}{:}o^?$ while label $\surd$ represents the termination of the choreography. Rule INTERACTION executes an interaction. Rule END terminates an empty choreography. Rule SEQUENCE executes a step in the

$$\text{(INTERACTION)}\quad \mathtt{a} \to \mathtt{b}{:}o^? \xrightarrow{\mathtt{a}\to\mathtt{b}:o^?} \mathbf{1} \qquad \text{(END)}\quad \mathbf{1} \xrightarrow{\surd} \mathbf{0}$$

$$\text{(SEQUENCE)}\quad \frac{C \xrightarrow{\sigma} C' \quad \sigma \neq \surd}{C;C'' \xrightarrow{\sigma} C';C''} \qquad \text{(PARALLEL)}\quad \frac{C \xrightarrow{\sigma} C' \quad \sigma \neq \surd}{C \parallel C'' \xrightarrow{\sigma} C' \parallel C''}$$

$$\text{(CHOICE)}\quad \frac{C \xrightarrow{\sigma} C'}{C + C'' \xrightarrow{\sigma} C'} \qquad \text{(SEQ-END)}\quad \frac{C_1 \xrightarrow{\surd} C_1' \quad C_2 \xrightarrow{\sigma} C_2'}{C_1;C_2 \xrightarrow{\sigma} C_2'} \qquad \text{(PAR-END)}\quad \frac{C_1 \xrightarrow{\surd} C_1' \quad C_2 \xrightarrow{\surd} C_2'}{C_1 \parallel C_2 \xrightarrow{\surd} C_1' \parallel C_2'}$$

Figure 1: Choreographies, semantics (symmetric rules omitted).

first component of a sequential composition. Rule PARALLEL executes an interaction from a component of a parallel composition while rule CHOICE starts the execution of an alternative in a nondeterministic choice. Rule SEQ-END acknowledges the termination of the first component of a sequential composition, starting the second component. Rule PAR-END synchronizes the termination of two parallel components.

We use the semantics to build some notions of traces for choreographies, which will be used later on for stating our results.

Definition 1 (Choreography traces)**.** *A (strong maximal) trace of a choreography C_1 is a sequence of labels $\tilde{\sigma} = \sigma_1, \ldots, \sigma_n$ such that there is a sequence of transitions $C_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_n} C_{n+1}$ and that C_{n+1} has no outgoing transitions.*

A weak trace of a choreography C is a sequence of labels $\tilde{\sigma}$ obtained by removing all the labels of the form $\mathtt{a} \to \mathtt{b}{:}o^*$ from a strong trace of C.

Two choreographies C and C' are (weak) trace equivalent iff they have the same set of (weak) traces.

3 Amending choreographies

In this section we consider three different connectedness properties, and we show how to enforce each one of them, preserving the set of weak traces of the choreography. We refer to [8] (a technical report extending [7]) for the description of why these properties guarantee projectability. Actually, in [7], different notions of projectability are considered according to whether the semantic model is synchronous or asynchronous, and on which behavioral relation between a choreography and its projection is required. The conditions presented and enforced below are correct and ensure projectability according to all the notions considered in [7].

All the conditions below are stated at the level of choreographies. Nevertheless, definitions of projection, projected system and of its semantics may help in understanding why they are needed. We collect these definitions in the Appendix.

We discuss the connectedness conditions in increasing order of difficulty to help the understanding. Then in Section 3.4 we combine them to make a general choreography connected.

3.1 Connecting sequences

Connectedness for sequence ensures that, in a sequential composition $C;C'$, the choreography C' starts its execution only after C terminates. To formalize this property, we first need to define auxiliary functions

transI and transF, which compute respectively the sets of initial and final interactions in a choreography:

$$\text{transI}(\mathtt{a} \to \mathtt{b}:o^?) = \text{transF}(\mathtt{a} \to \mathtt{b}:o^?) = \{\mathtt{a} \to \mathtt{b}:o^?\}$$
$$\text{transI}(\mathbf{1}) = \text{transI}(\mathbf{0}) = \text{transF}(\mathbf{1}) = \text{transF}(\mathbf{0}) = \emptyset$$
$$\text{transI}(C \parallel C') = \text{transI}(C + C') = \text{transI}(C) \cup \text{transI}(C')$$
$$\text{transF}(C \parallel C') = \text{transF}(C + C') = \text{transF}(C) \cup \text{transF}(C')$$
$$\text{transI}(C;C') = \text{transI}(C) \cup \text{transI}(C') \text{ if } \exists C'' \text{ such that } C \xrightarrow{\surd} C'', \text{ transI}(C) \text{ otherwise}$$
$$\text{transF}(C;C') = \text{transF}(C) \cup \text{transF}(C') \text{ if } \exists C'' \text{ such that } C' \xrightarrow{\surd} C'', \text{ transF}(C') \text{ otherwise}$$

Intuitively, connectedness for sequence can be ensured by guaranteeing that a role waits for termination of all the interactions in C and only then starts the execution of the interactions in C'. We can now formally state this property.

Definition 2 (Connectedness for sequence)**.** *A choreography C is* connected for sequence *iff each subterm of the form $C';C''$ satisfies* $\forall \mathtt{a} \to \mathtt{b}:o_1^? \in \text{transF}(C'), \forall \mathtt{c} \to \mathtt{d}:o_2^? \in \text{transI}(C'').\mathtt{b} = \mathtt{c}$.

Our transformation reconfigures the subterms $C';C''$ failing to meet the condition. Take one such term $C';C''$. Choose a fresh role $\mathtt{e}$. Consider all the interactions $\mathtt{a} \to \mathtt{b}:o^?$ contributing to $\text{transF}(C')$ in the term. For each of them choose a fresh operation o_f^* and replace $\mathtt{a} \to \mathtt{b}:o^?$ with $\mathtt{a} \to \mathtt{b}:o^?;\ \mathtt{b} \to \mathtt{e}:o_f^*$. Similarly, for each interaction $\mathtt{c} \to \mathtt{d}:o^?$ contributing to $\text{transI}(C'')$ choose a fresh operation o_f^* and replace $\mathtt{c} \to \mathtt{d}:o^?$ with $\mathtt{e} \to \mathtt{c}:o_f^*;\ \mathtt{c} \to \mathtt{d}:o^?$. The essential idea is that role $\mathtt{e}$ waits for all the threads in C' to terminate before starting threads in C''.

Proposition 1. *Given a choreography C we can derive using the pattern above a choreography C' which is connected for sequence such that C and C' are weak trace equivalent.*

Proof. We have to apply the pattern to all the subterms failing to satisfy the condition. We start from smaller subterms, and then move to larger ones. We have to show that at the end all the subterms satisfy the condition. The new subterms introduced by the transformation have the form $\mathtt{a} \to \mathtt{b}:o^?;\mathtt{b} \to \mathtt{e}:o^*$ and $\mathtt{e} \to \mathtt{c}:o^*;\mathtt{c} \to \mathtt{d}:o^?$, thus they satisfy the condition by construction. Let us consider a subterm $D';D''$ obtained by transforming a subterm $E';E''$ which already satisfied the condition (but whose subterms may have been transformed). It is easy to check that $\text{transF}(D') = \text{transF}(E')$ and $\text{transI}(D'') = \text{transI}(E'')$, thus the term is still connected for sequence. For subterms obtained by transforming subterms that did not satisfy the condition, the thesis holds by construction.

Weak trace equivalence is ensured since only interactions on private operations, which have no impact on weak trace equivalence, are added by the transformation. □

3.2 Connecting choices

Unique points of choice ensure that in a choice all the participants agree on the chosen branch. Intuitively, unique points of choice can be guaranteed by ensuring that a single participant performs the choice and informs all the other involved participants.

Definition 3 (Unique points of choice)**.** *A choreography C has* unique points of choice *iff each subterm of the form $C' + C''$ satisfies:*

1. $\forall \mathtt{a} \to \mathtt{b}:o_1^? \in \text{transI}(C'), \forall \mathtt{c} \to \mathtt{d}:o_2^? \in \text{transI}(C'').\mathtt{a} = \mathtt{c}$;
2. $\text{roles}(C') = \text{roles}(C'')$;

where $\text{roles}(\bullet)$ *computes the set of roles in a choreography.*

We present below a pattern able to ensure unique points of choice. We apply the pattern to all the subterms of the form $C' + C''$ that do not satisfy one of the conditions. Take one such term $C' + C''$. If condition 1 is not satisfied then choose a fresh role $\mathtt{e}$. Consider all the interactions $\mathtt{a} \to \mathtt{b}:o^?$ contributing to $\mathrm{transI}(C')$ or to $\mathrm{transI}(C'')$. For each of them choose a fresh operation o^*_f and replace $\mathtt{a} \to \mathtt{b}:o^?$ with $\mathtt{e} \to \mathtt{a}:o^*_f$; $\mathtt{a} \to \mathtt{b}:o^?$.

Suppose now that condition 1 is satisfied, while condition 2 is not. Then we can assume a role $\mathtt{e}$ which is the sender of all the interactions in $\mathrm{transI}(C' + C'')$. Consider each role $\mathtt{a}$ that occurs in C' but not in C'' (the symmetric case is analogous). For each of them add in parallel to C'' the interaction $\mathtt{e} \to \mathtt{a}:o^*_f$ where o^*_f is a fresh operation.

Proposition 2. *Given a choreography C we can derive using the pattern above a choreography C' which has unique points of choice such that C and C' are weak trace equivalent.*

Proof. We have to apply the pattern to all the subterms failing to satisfy the condition. We start from smaller subterms, and then move to larger ones. We have to show that at the end all the subterms satisfy the conditions. We consider the transformation ensuring condition 1 first, then the one ensuring condition 2.

For the first transformation, given a subterm $D' + D''$ there are two cases: either some initial interactions inside D' and D'' have been modified or not. In the second case the thesis follows by inductive hypothesis. In the first case all the initial interactions have been changed, and the freshly introduced role is the new sender in all of them. Thus the condition is satisfied.

Let us consider the second transformation. The transformation has no impact on subterms, since it only adds interactions in parallel. For the whole term the thesis holds by construction. Note that the term continues to satisfy the other condition.

Weak trace equivalence is ensured since only interactions on private operations, which have no impact on weak trace equivalence, are added by the transformation. □

3.3 Connecting repeated operations

If different interactions use the same operation, one has to ensure that messages do not get mixed, i.e. that the output of one interaction is not matched with the input of a different interaction on the same operation. Since in our model outputs do not contain the target participant, and inputs do not contain the expected sender, the problem is particularly relevant. The same problem however may occur also if the output contains the target participant and the input the expected sender, but only for interactions between the same participants, as shown by the example below.

Example 1. *Consider the choreography C below:*

$$C = (\mathtt{a} \to \mathtt{b}:o_1;\mathtt{b} \to \mathtt{c}:o;\mathtt{c} \to \mathtt{d}:o_2) \parallel (\mathtt{a} \to \mathtt{b}:o_3;\mathtt{b} \to \mathtt{c}:o;\mathtt{c} \to \mathtt{d}:o_4)$$

The two interactions $\mathtt{b} \to \mathtt{c}:o$, both between role $\mathtt{b}$ and role $\mathtt{c}$, may interfere, since the send from one of them could be received by the other one.

Clearly, given two interactions on the same operation, the problem does not occur if the output of the first is never enabled together with the input of the second, and vice versa.

To ensure this, in [7] we required causal dependencies between the input in one interaction and the outputs of different interactions on the same operation, and vice versa. To formalize this concept we introduce the notion of *event*. For each interaction $\mathtt{a} \to \mathtt{b}:o^?$ we distinguish two events: a sending event at role $\mathtt{a}$ and a receiving event at role $\mathtt{b}$. The sending event and the receiving event in the same interaction

are called matching events. We use e to range over events, s to range over sending events and r to range over receiving events. We denote by $\overline{e}$ the event matching event e. If event e precedes event e' in the causal relation then e' can become enabled only after e has been executed, thus they cannot be matched. However, this requirement alone is not suitable for our aims, since this condition is too strong to be enforced by a transformation preserving weak traces.

Luckily, also events in a suitable relation of conflict cannot be matched. Requiring that events are in opposite branches of a choice, however, is not enough, since they may be both enabled. For instance, in $\mathtt{a} \to \mathtt{b}:o + \mathtt{c} \to \mathtt{d}:o$ the output from $\mathtt{a}$ can be captured by $\mathtt{d}$[1]. However, if the role containing the event is already aware of the choice, then the events are never enabled together and thus cannot be matched, since when one of them becomes enabled the other one has already been discarded. We call *full conflict* such a relation. For instance, in

$$(\mathtt{a} \to \mathtt{b}:o';\mathtt{b} \to \mathtt{a}:o;\mathtt{a} \to \mathtt{c}:o') + (\mathtt{a} \to \mathtt{b}:o'';\mathtt{b} \to \mathtt{a}:o;\mathtt{a} \to \mathtt{c}:o'')$$

the two interactions on o do not interfere. Note that an interference could cause the wrong continuation to be chosen.

Thus we require here that a send and a receive in different interactions but on the same operation are either in a causality relation or in a full conflict relation. We call this condition *causality safety*. Causality safety can be enforced by a transformation preserving weak traces. We refer to [8] for the proof that causality safety ensures the desired properties of choreography projection.

To define causality safety we thus need to define a *causality relation* and a *full conflict relation*.

Definition 4 (Causality relation)**.** *Let us consider a choreography C. A* causality relation $\leq_C$ *is a partial order among events of C. We define $\leq_C$ as the minimum partial order satisfying:*

sequentiality: *for each subterm of the form $C';C''$ and each role* $\mathtt{a}$*, if r' is a receive event in C' at role* $\mathtt{a}$*, e'' is a generic event in C'' at role* $\mathtt{a}$ *then $r' \leq_C e''$;*

synchronization: *for each receive event r and generic event e', $r \leq_C e'$ implies $\overline{r} \leq_C e'$.*

Definition 5 (Full conflict relation)**.** *Let us consider a choreography C. A* full conflict relation ${}^{\mathrm{f}}\#_C$ *is a relation among events of C. We define ${}^{\mathrm{f}}\#_C$ as the smallest symmetric relation satisfying:*

choice: *for each subterm of the form $C' + C''$ and each role* $\mathtt{a}$*, if e' is an event in C' at role* $\mathtt{a}$*, e'' is an event in C'' at role* $\mathtt{a}$ *then $e' \, {}^{\mathrm{f}}\#_C \, e''$;*

causality: *if $e' \, {}^{\mathrm{f}}\#_C \, e''$ and $e'' \leq_C e'''$ then $e' \, {}^{\mathrm{f}}\#_C \, e'''$.*

We can now define causality safety.

Definition 6 (Causality safety)**.** *A choreography C is* causality safe *iff for each pair of interactions $\mathtt{a} \to \mathtt{b}:o^?$ (with events s_1 and r_1) and $\mathtt{c} \to \mathtt{d}:o^?$ (with events s_2 and r_2) on the same operation $o^?$ we have that:*

- $s_1 \leq_C r_2 \vee r_2 \leq_C s_1 \vee s_1 \, {}^{\mathrm{f}}\#_C \, r_2$;
- $s_2 \leq_C r_1 \vee r_1 \leq_C s_2 \vee s_2 \, {}^{\mathrm{f}}\#_C \, r_1$.

As an example, choreography $C = \mathtt{a} \to \mathtt{b}:o \parallel \mathtt{c} \to \mathtt{d}:o$ is not causality safe, since the output on o at $\mathtt{a}$ is enabled together with the input on o at $\mathtt{d}$, thus enabling a communication from $\mathtt{a}$ to $\mathtt{d}$ which should not be allowed.

[1] This choreography however has not unique points of choice.

A causality safety issue is immediately solved by renaming one of the operations. However, this changes the specification. We show how to solve the causality safety issue while sticking to the original (weak) behavior. We have different approaches according to the top-level operator of the smaller term including the conflicting interactions. Thus we distinguish *parallel causality safety*, *sequential causality safety* and *choice causality safety*.

To solve parallel causality safety issues we apply a form of *expansion law* that transforms the parallel composition into nondeterminism, thus either removing completely the causality safety issue or transforming it into sequential or choice causality safety, discussed later on. Using the expansion law one can transform any choreography into a *normal form* defined as below.

Definition 7 (Normal form)**.** *A choreography C is in* normal form *if it is written as:*

$$\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?; C_i$$

where $\sum_i$ is n-ary nondeterministic choice (we can see the empty sum as $\mathbf{0}$) and for each i also C_i is in normal form.

The expansion law is defined below.

Definition 8 (Expansion law)**.**

$$\begin{aligned}(\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?;\, C_i) \parallel (\sum_j \mathtt{a}_j \to \mathtt{b}_j : o_j^?; C_j) \;=\; & (\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?; (C_i \parallel (\sum_j \mathtt{a}_j \to \mathtt{b}_j : o_j^?; C_j))) \\ & + (\sum_j \mathtt{a}_j \to \mathtt{b}_j : o_j^?; (C_j \parallel (\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?; C_i)))\end{aligned}$$

The expansion law is correct w.r.t. trace equivalence, in the sense that applying the expansion law does not change the set of traces (neither strong nor weak).

Lemma 1. *Let C and C' be choreographies, with C' obtained by applying the expansion law to a subterm of C. Then C and C' are (strong and weak) trace equivalent.*

Proof. Labels not involving the subterm are easily mimicked. Consider the first label involving the subterm. If no such label exists then the thesis follows. Otherwise, the label corresponds to the execution of one of the interactions $\mathtt{a}_i \to \mathtt{b}_i : o_i^?$ or $\mathtt{a}_j \to \mathtt{b}_j : o_j^?$. Executing any of these interactions reduces both the terms to the same term. The thesis follows. □

Using the expansion law we can put any choreography C in normal form.

Proposition 3 (Normalization)**.** *Given a choreography C there is a choreography C' in normal form such that C and C' are weak trace equivalent.*

Proof. The proof is by structural induction on the number of interactions occurring in C. The cases of interactions and $\mathbf{0}$ are trivial. Choreography $\mathbf{1}$ can be replaced by any interaction on a private operation without changing the set of weak traces. For sequential composition note that $(\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?; C_i); C'$ and $(\sum_i \mathtt{a}_i \to \mathtt{b}_i : o_i^?; C_i; C')$ have the same set of traces. $C_i; C'$ can be transformed in normal form by inductive hypothesis. For nondeterministic choice the thesis is trivial (it is easy to check that nondeterministic choice is associative). For parallel composition one can apply the expansion law, and the thesis follows from Lemma 1 and inductive hypothesis. □

Let us now consider sequential causality safety. The term should have the form $C';C''$, with an interaction $\mathtt{a} \to \mathtt{b}:o^?$ in C' and an interaction $\mathtt{c} \to \mathtt{d}:o^?$ in C''. If the term satisfies connectedness for sequence then the send at c cannot become enabled before the receive at b. We thus have to ensure that the receive at d becomes enabled after the message from the send at a has been received. Choose fresh operations o^*_f and o^*_g and replace $\mathtt{a} \to \mathtt{b}:o^?$ with $\mathtt{a} \to \mathtt{b}:o^?;\mathtt{b} \to \mathtt{d}:o^*_f;\mathtt{d} \to \mathtt{b}:o^*_g$.

The pattern for choice causality safety is similar. The term should have the form $C' + C''$, with an interaction $\mathtt{a} \to \mathtt{b}:o^?$ in C' and an interaction $\mathtt{c} \to \mathtt{d}:o^?$ in C''. Assume that there is an interference between the send at a and the receive at d (the symmetric case is similar). Choose fresh operations o^*_f and o^*_g and replace $\mathtt{c} \to \mathtt{d}:o^?$ with $\mathtt{c} \to \mathtt{d}:o^*_f;\mathtt{d} \to \mathtt{c}:o^*_g;\mathtt{c} \to \mathtt{d}:o^?$

To prove the correctness of the transformation we need two auxiliary lemmas.

Lemma 2. *Let C be a choreography which is connected for sequence. Then all sending events in C depend on sending events in initial interactions in C.*

Proof. By structural induction on C. The only difficult case is sequential composition, when $C = C';C''$. For events in C' the thesis follows by inductive hypothesis. For events in C'', if they are in an initial interaction, from connectedness for sequence they are in the same role as receiving events in the final interactions in C', thus they depend on them. By synchronization they also depend on sending events in C'. The thesis follows by induction and by transitivity. For events not in initial interactions in C'', by inductive hypothesis they depend on events in initial interactions in C'', thus the thesis follows from what proved above by transitivity. □

Lemma 3. *Let $C = C';C''$ be a choreography which is connected for sequence. Then there are no causality safety issues between receiving events in C' and sending events in C''.*

Proof. From connectedness for sequence all the receiving events in final interactions of C' and all the sending events in initial interactions of C'' are performed by the same role. Thus, from the sequentiality condition in the definition of causality relation, they are causally related. Now the more difficult case is when the send s in C'' is not initial and the receive r in C' is not final. Thanks to Lemma 2 the send s depends on an initial send, and by transitivity on each final receive. Thanks to synchronization it depends also on each final send. Assume r is not final. Thanks to connectedness for sequence it is in the same role of a following send s', thus s' depends on r. If s' is final the thesis follows. Otherwise we iterate the procedure on a smaller term, and the thesis follows by induction. □

We can now prove the correctness of the transformation.

Proposition 4. *Let C be a choreography which is connected for sequence, has unique points of choice and is parallel causality safe. We can derive using the pattern above a choreography C' which has no sequential or choice causality safety issue such that C and C' are weak trace equivalent. Also, C' is still connected for sequence, has unique points of choice and is parallel causality safe.*

Proof. We prove the thesis for just one causality safety issue, the more general result follows by iteration. Take a causality safety issue, and consider the smallest subterm including the two conflicting interactions. We have two cases according to whether the subterm has the form $C';C''$ or $C' + C''$.

Let us consider the first case. Thanks to Lemma 3 the issue is between the send at a in an interaction $\mathtt{a} \to \mathtt{b}:o^?$ in C' and a receive at d in an interaction $\mathtt{c} \to \mathtt{d}:o^?$ in C''.

After the transformation the receive at d in $\mathtt{c} \to \mathtt{d}:o^?$ depends on the receive at d in $\mathtt{b} \to \mathtt{d}:o^*_f$ by sequentiality. The receive at d in $\mathtt{b} \to \mathtt{d}:o^*_f$ depends on the send at b inside the same interaction by

synchronization, on the receive at b in $\mathtt{a} \to \mathtt{b}:o^?$ by sequentiality, and on the send at a inside the same interaction again by synchronization. This proves that the issue is solved.

Let us consider the second case. Assume that the issue is between a send in an interaction $\mathtt{a} \to \mathtt{b}:o^?$ in C' and a receive in an interaction $\mathtt{c} \to \mathtt{d}:o^?$ in C''. The sends at a and at c are causally dependent on sends in an initial interaction of the term thanks to Lemma 2. The transformation adds a dependency between the send at c and the receive at d inside the same interaction. Thus the send at a and the receive at d are in full conflict, since they are both causally dependent on sends from initial interactions of the term, which are all in the same role thanks to unique points of choice. Again, this proves that the issue is solved.

Weak trace equivalence is ensured since only interactions on private operations, which have no impact on weak trace equivalence, are added by the transformation.

We have to show that the other properties are preserved. All of them hold by construction for the newly introduced terms. Connectedness for sequence is preserved since the transformation preserves senders of initial interactions and receivers of final interactions. Unique points of choice are preserved for the same reason, and since the transformation does not add new roles. Parallel causality safety is preserved since the transformation only introduces interactions on fresh operations. □

3.4 Combining the amending techniques

Till now we have shown that given a choreography which fails to satisfy one of the connectedness conditions, we can transform it into an equivalent one that satisfies the chosen connectedness condition. Some care is required to avoid that, while ensuring that one condition is satisfied, violations of other conditions are introduced. In fact, if such a situation would occur repeatedly, the connecting procedure may not terminate.

The following theorem proves that we can combine the connecting patterns presented in the previous sections to define a terminating algorithm transforming any choreography into a connected choreography.

Theorem 1 (Connecting choreographies)**.** *There is a terminating procedure that given any choreography C creates a new choreography D such that:*

- *D is connected;*
- *C and D are weak trace equivalent.*

Proof. We can apply the normalization procedure to all the subterms of choreography C that do not satisfy parallel causality safety, starting from the smallest subterms to the largest, to get a choreography C' which is parallel causality safe (since the undesired parallel compositions have been removed) and which is weak trace equivalent to C thanks to Proposition 3.

Now, again from the smallest subterms to the largest, we can apply to C' the procedure for ensuring unique points of choice to those subterms which have a top-level nondeterministic choice operator, and the procedure for making them connected for sequence to those subterms which have a top-level sequential composition operator obtaining a choreography C''.

For terms of the first kind, thanks to Proposition 2, we obtain terms which have unique points of choice. They are also parallel causality safe and connected for sequence, since the transformation may not create these issues. The same holds for terms of the second kind thanks to Proposition 1. In both the cases, the resulting term is weak trace equivalent to the starting one. By transitivity C'' is weak trace equivalent to C.

Finally, to each sequential and choice causality safety issue we can apply the procedure to solve it. From Proposition 4 we know that the resulting choreography D has no sequential or choice causality

safety issue, still satisfies the other properties and is weak trace equivalent to C''. The thesis follows by transitivity. □

Example 2. *We now apply our procedure to the paradigmatic choreography* $C = \mathtt{a} \to \mathtt{b}:o \parallel \mathtt{c} \to \mathtt{d}:o$. *First note that C does not satisfy parallel causality safety. By application of the expansion law we obtain:*

$$C_1 = \mathtt{a} \to \mathtt{b}:o;\mathtt{c} \to \mathtt{d}:o + \mathtt{c} \to \mathtt{d}:o;\mathtt{a} \to \mathtt{b}:o$$

Proceeding from smaller to larger subterms, we first encounter the subterms $\mathtt{a} \to \mathtt{b}:o;\mathtt{c} \to \mathtt{d}:o$ *and* $\mathtt{c} \to \mathtt{d}:o;\mathtt{a} \to \mathtt{b}:o$ *which are not connected for sequence (and are not sequential causality safe). By applying the corresponding pattern to the two subterms, we obtain:*

$$C_2 = \mathtt{a} \to \mathtt{b}:o;\mathtt{b} \to \mathtt{e}':o_1^*;\mathtt{e}' \to \mathtt{c}:o_2^*;\mathtt{c} \to \mathtt{d}:o + \mathtt{c} \to \mathtt{d}:o;\mathtt{d} \to \mathtt{e}'':o_3^*;\mathtt{e}'' \to \mathtt{a}:o_4^*;\mathtt{a} \to \mathtt{b}:o$$

Now, the whole term does not have asynchronous unique points of choice, and is not choice causality safe. By applying the first of the transformations ensuring unique points of choice, we obtain:

$$\begin{aligned} C_3 \quad = \quad & \mathtt{e} \to \mathtt{a}:o_5^*;\mathtt{a} \to \mathtt{b}:o;\mathtt{b} \to \mathtt{e}':o_1^*;\mathtt{e}' \to \mathtt{c}:o_2^*;\mathtt{c} \to \mathtt{d}:o + \\ & \mathtt{e} \to \mathtt{c}:o_6^*;\mathtt{c} \to \mathtt{d}:o;\mathtt{d} \to \mathtt{e}'':o_3^*;\mathtt{e}'' \to \mathtt{a}:o_4^*;\mathtt{a} \to \mathtt{b}:o \end{aligned}$$

By applying the transformation ensuring that both the branches have the same set of roles, we obtain:

$$\begin{aligned} C_4 \quad = \quad & \big(\mathtt{e} \to \mathtt{a}:o_5^*;\mathtt{a} \to \mathtt{b}:o;\mathtt{b} \to \mathtt{e}':o_1^*;\mathtt{e}' \to \mathtt{c}:o_2^*;\mathtt{c} \to \mathtt{d}:o \parallel \mathtt{e} \to \mathtt{e}'':o_7^*\big) + \\ & \big(\mathtt{e} \to \mathtt{c}:o_6^*;\mathtt{c} \to \mathtt{d}:o;\mathtt{d} \to \mathtt{e}'':o_3^*;\mathtt{e}'' \to \mathtt{a}:o_4^*;\mathtt{a} \to \mathtt{b}:o \parallel \mathtt{e} \to \mathtt{e}':o_8^*\big) \end{aligned}$$

We are now left with sequential and choice causality safety issues. We start by solving the sequential ones, which are one for each branch. As expected, they are between the sender of the first interaction on o and the receiver of the second. By applying the pattern we get:

$$\begin{aligned} C_5 \quad = \quad & \big(\mathtt{e} \to \mathtt{a}:o_5^*;\mathtt{a} \to \mathtt{b}:o;\mathtt{b} \to \mathtt{d}:o_9^*;\mathtt{d} \to \mathtt{b}:o_{10}^*;\mathtt{b} \to \mathtt{e}':o_1^*;\mathtt{e}' \to \mathtt{c}:o_2^*;\mathtt{c} \to \mathtt{d}:o \parallel \mathtt{e} \to \mathtt{e}'':o_7^*\big) + \\ & \big(\mathtt{e} \to \mathtt{c}:o_6^*;\mathtt{c} \to \mathtt{d}:o;\mathtt{d} \to \mathtt{b}:o_{11}^*;\mathtt{b} \to \mathtt{d}:o_{12}^*;\mathtt{d} \to \mathtt{e}'':o_3^*;\mathtt{e}'' \to \mathtt{a}:o_4^*;\mathtt{a} \to \mathtt{b}:o \parallel \mathtt{e} \to \mathtt{e}':o_8^*\big) \end{aligned}$$

We are left with choice causality safety issues between the interaction $\mathtt{a} \to \mathtt{b}:o$ *in the first branch and the interaction* $\mathtt{c} \to \mathtt{d}:o$ *in the second branch. By applying the pattern we get:*

$$\begin{aligned} C_6 \quad = \quad & \big(\mathtt{e} \to \mathtt{a}:o_5^*;\mathtt{a} \to \mathtt{b}:o_{13}^*;\mathtt{b} \to \mathtt{a}:o_{14}^*;\mathtt{a} \to \mathtt{b}:o;\mathtt{b} \to \mathtt{d}:o_9^*;\mathtt{d} \to \mathtt{b}:o_{10}^*; \\ & \quad \mathtt{b} \to \mathtt{e}':o_1^*;\mathtt{e}' \to \mathtt{c}:o_2^*;\mathtt{c} \to \mathtt{d}:o \parallel \mathtt{e} \to \mathtt{e}'':o_7^*\big) + \\ & \big(\mathtt{e} \to \mathtt{c}:o_6^*;\mathtt{c} \to \mathtt{d}:o_{15}^*;\mathtt{d} \to \mathtt{c}:o_{16}^*;\mathtt{c} \to \mathtt{d}:o;\mathtt{d} \to \mathtt{b}:o_{11}^*;\mathtt{b} \to \mathtt{d}:o_{12}^*; \\ & \quad \mathtt{d} \to \mathtt{e}'':o_3^*;\mathtt{e}'' \to \mathtt{a}:o_4^*;\mathtt{a} \to \mathtt{b}:o \parallel \mathtt{e} \to \mathtt{e}':o_8^*\big) \end{aligned}$$

This example shows that solving a parallel causality safety issue may require a considerable amount of auxiliary communications. Thus, it is advised to change the name of the operation if possible. If not possible, our approach will solve the problem. Other issues are solved more easily, since they involve no duplication of terms.

4 Application: Two-buyers protocol

We show now how our transformation for connecting choreographies can be used as an effective design tool for the programming of multiparty choreographies. We model the example reported in [6], the two-buyers protocol, where two buyers – $\mathtt{b}_1$ and $\mathtt{b}_2$ – combine their finances for buying a product from a seller

s. The protocol starts with $\mathtt{b}_1$ asking the price for the product of interest to s. Then, s communicates the price to both $\mathtt{b}_1$ and $\mathtt{b}_2$. Subsequently, $\mathtt{b}_1$ notifies $\mathtt{b}_2$ of how much she is willing to contribute to the purchase. Finally, $\mathtt{b}_2$ chooses whether to confirm the purchase and ask s to send a delivery date for the product; otherwise, the choreography terminates. We do not deal here with how this choice is performed, as our choreographies abstract from data.

To create a quick prototype choreography C for the two-buyers protocol, we focus only on the main interactions. When writing the choreography we do not worry about connectedness conditions, since we will enforce them automatically later on. The code follows naturally:

$$\begin{array}{rl} C = & \mathtt{b}_1 \to \mathtt{s}:\mathsf{price};\ (\ \mathtt{s} \to \mathtt{b}_1:\mathsf{quote1}\ \|\ \mathtt{s} \to \mathtt{b}_2:\mathsf{quote2}\ );\ \mathtt{b}_1 \to \mathtt{b}_2:\mathsf{contrib};\\ & (\ \mathtt{b}_2 \to \mathtt{s}:\mathsf{ok}; \mathtt{s} \to \mathtt{b}_2:\mathsf{delivery}\ +\ \mathbf{1}\ ) \end{array}$$

The code above is just a direct translation of our explanation in natural language into a choreography. We can immediately observe that the choreography is not connected, since it contains 2 violations of the connectedness conditions:

- the subterm $(\ \mathtt{s} \to \mathtt{b}_1:\mathsf{quote1}\ \|\ \mathtt{s} \to \mathtt{b}_2:\mathsf{quote2}\ );\ \mathtt{b}_1 \to \mathtt{b}_2:\mathsf{contrib}$ is not connected for sequence; thus, e.g., $\mathtt{b}_1$ may send the contrib message before $\mathtt{b}_2$ receives the message for quote2[2];
- the subterm $(\ \mathtt{b}_2 \to \mathtt{s}:\mathsf{ok}; \mathtt{s} \to \mathtt{b}_2:\mathsf{delivery}\ +\ \mathbf{1}\ )$ has not unique points of choice.

We can apply our transformation to amend our choreography prototype, transforming it into a connected choreography which is weak trace equivalent to C, obtaining:

$$\begin{array}{rl} C_1 = & \mathtt{b}_1 \to \mathtt{s}:\mathsf{price};\ (\ \mathtt{s} \to \mathtt{b}_1:\mathsf{quote1};\ \mathtt{b}_1 \to \mathtt{e}_1:o_1^*\ \|\ \mathtt{s} \to \mathtt{b}_2:\mathsf{quote2};\ \mathtt{b}_2 \to \mathtt{e}_1:o_2^*\ );\\ & \mathtt{e}_1 \to \mathtt{b}_1:o_3^*;\ \mathtt{b}_1 \to \mathtt{b}_2:\mathsf{contrib};\ (\ \mathtt{b}_2 \to \mathtt{s}:\mathsf{ok}; \mathtt{s} \to \mathtt{b}_2:\mathsf{delivery}\ +(\ \mathbf{1}\ \|\ \mathtt{b}_2 \to \mathtt{s}:o_4^*\ )\ ) \end{array}$$

The choreography C_1 above is connected, thus it can be projected, and the projection will be trace equivalent to C_1 itself (thanks to the results in [7]), and weak trace equivalent to the original choreography C.

5 Conclusions

In previous work [7] we have defined syntactic conditions that guarantee choreography connectedness, i.e. that the endpoints obtained by the natural projection of a choreography correctly implement the global specification given by the choreography itself. In this paper we have presented a procedure for amending non connected choreographies by reducing parallelism and adding interactions on private operations in such a way that all the above conditions are satisfied, while preserving the observational semantics. To the best of our knowledge, only two other papers consider the possibility to add messages to a choreography in order to make it correctly projectable [10, 9].

In [10] non connected sequences are connected by adding a synchronization from every role involved in a final interaction before the sequential composition to every role involved in an initial interaction after the sequential composition, while non connected choices are modified by selecting a dominant role responsible for taking the choice and communicating it to the other participants. Our approach reduces the number of added synchronizations in the case of sequential compositions, and allows for a symmetric projection that treats all the roles in the same way in the case of choices. In [10] there is no discussion about repeated operations, and to the best of our understanding of the paper this is problematic. In fact,

[2] This may happen only with the asynchronous semantics.

the choreography C in Example 1 (written according to our syntax, which is slightly different w.r.t. the one adopted in [10]) is well-formed according to the conditions in [10], but it is not projectable.

Collaboration and *Sequence diagrams* are popular visual representations of choreographies [5]. In this area, the work more similar to ours, is [9] where the problem of amending collaboration diagrams is addressed. In collaboration diagrams the interactions are labels of edges in a graph which has one node for each role, and one edge for each pair of interacting roles. Interactions are organized in threads representing sub-choreographies. Messages are totally ordered within the same thread, while a partial order can be defined among interactions belonging to different threads. The problem of amending non connected choreographies is simplified in this context: collaboration diagrams do not have a global choice composition operator among sub-choreographies, and two distinct threads use two disjoint sets of operations. For this reason, only the problem of non connected sequences is addressed in [9].

Another paper amending choreographies is [2], however they consider a different problem. In fact, they consider choreographies including annotations on the communicated data, and they concentrate on amending those annotations.

Possible extensions of the present work include its application to existing choreography languages, such as [4], where however the intended semantics of sequential composition is weaker, since independent interactions can be freely swapped. The main difficulty in extending the approach is dealing with infinite behaviors. Having an iteration construct should not be a problem: roughly one has to make connected the choice between iterating and exiting the iteration using the techniques for choice, and the sequential execution of different iterations using the techniques for sequence. General recursion would be more difficult to deal with, since subterms belonging to different iterations may execute in parallel. Adding data to our language would not change the issues discussed in the present paper.

References

[1] Laura Bocchi, Kohei Honda, Emilio Tuosto & Nobuko Yoshida (2010): *A Theory of Design-by-Contract for Distributed Multiparty Interactions*. In: *Proc. of CONCUR 2010*, *LNCS* 6269, Springer, pp. 162–176.

[2] Laura Bocchi, Julien Lange & Emilio Tuosto (2012): *Three Algorithms and a Methodology for Amending Contracts for Choreographies*. *Sci. Ann. Comp. Sci.* 22(1), pp. 61–104.

[3] Marco Carbone, Kohei Honda & Nobuko Yoshida (2007): *Structured Communication-Centred Programming for Web Services*. In: *Proc. of ESOP'07*, *LNCS* 4421, Springer, pp. 2–17.

[4] Marco Carbone & Fabrizio Montesi (2013): *Deadlock-freedom-by-design: multiparty asynchronous global programming*. In: *Proc. of POPL 2013*, ACM, pp. 263–274.

[5] Martin Fowler (2003): *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)*. Addison Wesley.

[6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In: *Proc. of POPL'08*, ACM Press, pp. 273–284.

[7] Ivan Lanese, Claudio Guidi, Fabrizio Montesi & Gianluigi Zavattaro (2008): *Bridging the Gap between Interaction- and Process-Oriented Choreographies*. In: *Proc. of SEFM'08*, IEEE Press, pp. 323–332.

[8] Ivan Lanese, Fabrizio Montesi & Gianluigi Zavattaro: *Classifying Relationships between Interaction- and Process-Oriented Choreographies*. Available at `http://www.cs.unibo.it/~lanese/publications/fulltext/ioc.pdf`.

[9] Gwen Salaun, Tevfik Bultan & Nima Roohi (2012): *Realizability of Choreographies Using Process Algebra Encodings*. *IEEE Transactions on Services Computing* 5(3), pp. 290–304.

[10] Qiu Zongyan, Zhao Xiangpeng, Cai Chao & Yang Hongli (2007): *Towards the Theoretical Foundation of Choreography*. In: *Proc. of WWW'07*, ACM Press, pp. 973–982.

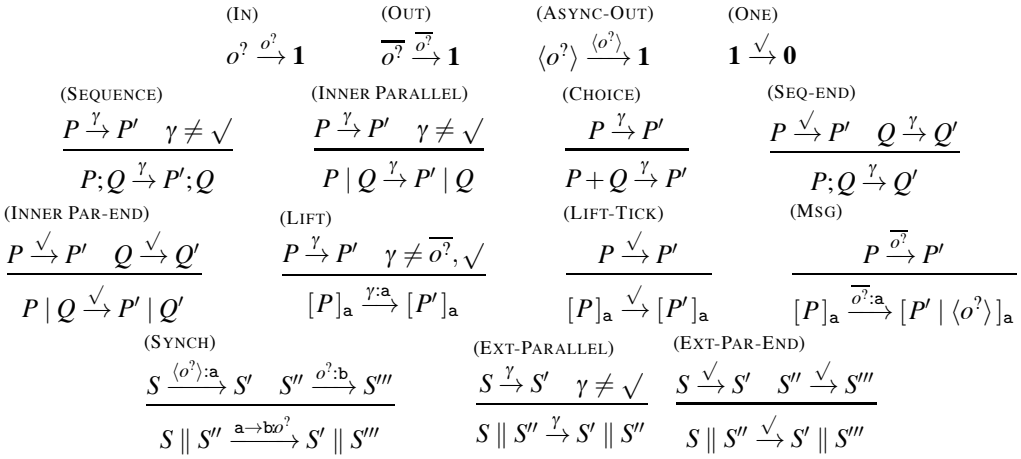

Table 1: Projected systems asynchronous semantics (symmetric rules omitted).

A Projected systems

We describe here the syntax and the operational semantics of *projected systems*. Projected systems, ranged over by $S, S', \ldots$, are composed by *processes*, ranged over by $P, P', \ldots$, describing the behavior of participants,

$$\begin{array}{rcl} P & ::= & o^? \mid \overline{o^?} \mid \mathbf{1} \mid P;P' \mid \ P \mid P' \ \mid P+P' \mid \langle o^? \rangle \mid \mathbf{0} \\ S & ::= & [P]_{\mathtt{a}} \mid S \parallel S' \end{array}$$

Processes include input action $o^?$ and output action $\overline{o^?}$ on a specific operation $o^?$ (either public or private), the empty process $\mathbf{1}$, sequential and parallel composition, and nondeterministic choice. The runtime syntax includes also messages $\langle o^? \rangle$, used in the definition of the asynchronous semantics, and the deadlocked process $\mathbf{0}$. Projected systems are parallel compositions of roles. Each role has a role name and executes a process. We require role names to be unique.

We define two LTS semantics for projected systems, one *synchronous* and one *asynchronous*. In the synchronous semantics input actions and output actions interact directly, while in the asynchronous one the sending event creates a message that, later, may interact with the corresponding input, generating a receiving event.

The asynchronous LTS for projected systems is the smallest LTS closed under the rules in Table 1. We use γ to range over labels. Symmetric rules for parallel compositions (both internal and external) and choice have been omitted.

Rules IN and OUT execute input actions and output actions, respectively. Rule ASYNCH-OUT makes messages available for a corresponding input action. Rule ONE terminates an empty process. Rule SEQUENCE executes a step in the first component of a sequential composition. Rule INNER PARALLEL executes an action from a component of a parallel composition while rule CHOICE starts the execution of an alternative in a nondeterministic choice. Rule SEQ-END acknowledges the termination of the first component of a sequential composition, starting the second component. Rule INNER PAR-END synchronizes the termination of two parallel components. Rule LIFT lifts actions to the system level,

tagging them with the name of the role executing them. Action $\surd$ instead is dealt with by rule LIFT-TICK, which lifts it without adding the role name. Outputs instead are stored as messages by rule MSG. Rule SYNCH synchronizes a message with the corresponding input action, producing an interaction. Rule EXT-PARALLEL allows parallel systems to stay idle. Finally, rule EXT-PAR-END synchronizes the termination of parallel systems.

The synchronous LTS for projected systems is the smallest LTS closed under the rules in Table 1, where rules OUT, ASYNC-OUT and MSG are deleted and the new rule SYNC-OUT below is added:

$$\frac{}{\overline{o^?} \xrightarrow{\langle o^? \rangle}_s \mathbf{1}} \text{(SYNC-OUT)}$$

This rule allows outputs in the synchronous semantics to send messages that can directly interact with the corresponding input at the system level.

Synchronous transitions are denoted as $\xrightarrow{\gamma}_s$ instead of $\xrightarrow{\gamma}$, to distinguish them from the asynchronous ones.

As for choreographies, we define *traces*. We have different possibilities: in addition to the distinction between strong and weak traces, we distinguish *synchronous* and *asynchronous* traces.

Definition 9 (Projected systems traces)**.** *A (strong maximal) synchronous trace of a projected system S_1 is a sequence of labels $\gamma_1, \ldots, \gamma_n$, where γ_i is of the form $\mathtt{a} \to \mathtt{b}:o^?$, or $\surd$ for each $i \in \{1, \ldots, n\}$, such that there is a sequence of synchronous transitions $S_1 \xrightarrow{\gamma_1}_s \ldots \xrightarrow{\gamma_n}_s S_{n+1}$ and such that S_{n+1} has no outgoing transitions of the same form.*

A (strong maximal) asynchronous trace of a projected system S_1 is a sequence of labels $\gamma_1, \ldots, \gamma_n$, where γ_i is of the form $\overline{o^?} : \mathtt{a}$, $\mathtt{a} \to \mathtt{b}:o^?$, or $\surd$ for each $i \in \{1, \ldots, n\}$, such that there is a sequence of asynchronous transitions $S_1 \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_n} S_{n+1}$ and such that S_{n+1} has no outgoing transitions of the same form. A weak (synchronous/asynchronous) trace of a projected system S_1 is obtained by removing all labels $\overline{o^*} : \mathtt{a}$ and $\mathtt{a} \to \mathtt{b}:o^*$ from a strong (synchronous/asynchronous) trace of S_1.

In the definition of traces, input actions and messages are never considered, since they represent interactions with the external world, while we are interested in the behavior of closed systems.

We have shown in [7] that a connected choreography and the corresponding projected system have the same set of strong traces, if the synchronous semantics is used for the projected system. The correspondence with the asynchronous semantics is more complex, and we refer to [7] for a precise description.

B Projection

In this section we show how to derive from a choreography C a projected system S implementing it. The idea is to project the choreography C on the different roles, and build the system S as parallel composition of the projections on the different roles. We consider here the most natural projection, which is essentially an homomorphism on most operators. As shown in [7], if C satisfies the connectedness conditions, the resulting projected system is behaviorally related to the starting choreography C.

Definition 10 (Projection function)**.** *Given a choreography C and a role $\mathtt{a}$, the projection $\mathrm{proj}(C, \mathtt{a})$ of choreography C on role $\mathtt{a}$ is defined by structural induction on C:*

$$
\begin{array}{rcl}
\text{proj}(\mathtt{a} \to \mathtt{b}:o^?, \mathtt{a}) & = & \overline{o^?} \\
\text{proj}(\mathtt{a} \to \mathtt{b}:o^?, \mathtt{b}) & = & o^? \\
\text{proj}(\mathtt{a} \to \mathtt{b}:o^?, \mathtt{c}) & = & \mathbf{1} \textit{ if } \mathtt{c} \neq \mathtt{a}, \mathtt{b} \\
\text{proj}(\mathbf{1}, \mathtt{a}) & = & \mathbf{1} \\
\text{proj}(\mathbf{0}, \mathtt{a}) & = & \mathbf{0} \\
\text{proj}(C;C', \mathtt{a}) & = & \text{proj}(C, \mathtt{a});\text{proj}(C', \mathtt{a}) \\
\text{proj}(C \parallel C', \mathtt{a}) & = & \text{proj}(C, \mathtt{a}) \mid \text{proj}(C', \mathtt{a}) \\
\text{proj}(C + C', \mathtt{a}) & = & \text{proj}(C, \mathtt{a}) + \text{proj}(C', \mathtt{a})
\end{array}
$$

We denote with $\|_{i \in I}\, S_i$ the parallel composition of systems S_i for each $i \in I$.

Definition 11. *Given a choreography C, the projection of C is the system S defined by:*

$$\text{proj}(C) = \|_{\mathtt{a} \in \text{roles}(C)}\, [\text{proj}(C, \mathtt{a})]_{\mathtt{a}}$$

where $\text{roles}(C)$ *is the set of roles in* C.

Blind-date Conversation Joining *

Luca Cesari[1,2]
[1]Università di Pisa, Italy
cesari@di.unipi.it

Rosario Pugliese[2]
[2]Università degli Studi di Firenze, Italy
luca.cesari@unifi.it
rosario.pugliese@unifi.it

Francesco Tiezzi[3]
[3]IMT Advanced Studies Lucca, Italy
francesco.tiezzi@imtlucca.it

We focus on a form of joining conversations among multiple parties in service-oriented applications where a client may asynchronously join an existing conversation without need to know in advance any information about it. More specifically, we show how the correlation mechanism provided by orchestration languages enables a form of conversation joining that is completely transparent to clients and that we call 'blind-date joining'. We provide an implementation of this strategy by using the standard orchestration language WS-BPEL. We then present its formal semantics by resorting to COWS, a process calculus specifically designed for modelling service-oriented applications. We illustrate our approach by means of a simple, but realistic, case study from the online games domain.

1 Introduction

The increasing success of e-business, e-learning, e-government, and other similar emerging models, has led the World Wide Web, initially thought of as a system for human use, to evolve towards an architecture for supporting automated use. A new computing paradigm has emerged, called Service-Oriented Computing (SOC), that advocates the use of loosely-coupled *services*. These are autonomous, platform-independent, computational entities that can be described, published, discovered, and assembled as the basic blocks for building interoperable and evolvable systems and applications. Currently, the most successful instantiation of the SOC paradigm are *Web Services*, i.e. sets of operations that can be published, located and invoked through the Web via XML messages complying with given standard formats.

In SOC, service definitions are used as templates for creating service instances that deliver application functionalities to either end-user applications or other instances. Upon service invocation, differently from what usually happens in traditional client-server paradigms, the caller (i.e., a service client) and the callee (i.e., a service provider) can engage in a *conversation* during which they exchange the information needed to complete all the activities related to the specific service. For instance, a client of an airplane ticket reservation service usually interacts several times with the service before selecting the specific flight to be reserved. Although initially established between a caller and callee, a conversation can dynamically accommodate and dismiss participants. Therefore, a conversation is typically a loosely-coupled, multiparty interaction among a (possibly dynamically varying) number of participants.

The loosely-coupled nature of SOC implies that, from a technological point of view, the connection between communicating partners cannot be assumed to persist for the duration of a whole conversation. Even the execution of a simple request-response message exchange pattern provides no built-in means of automatically associating the response message with the original request. It is up to each single message to provide a form of context thus enabling partners to associate the message with others. This is achieved by including values in the message which, once located, can be used to correlate the message with others logically forming the same stateful conversation. The link among partners is thus determined by so called *correlation values*: only messages containing the 'right' correlation values are processed by the partner.

*This work has been partially sponsored by the EU project ASCENS (257414).

Message correlation is an essential part of messaging within SOC as it enables the persistence of activities' context and state across multiple message exchanges while preserving service statelessness and autonomy, and the loosely-coupled nature of SOC systems. It is thus at the basis of Web Services interaction which is implemented on top of stateless Internet protocols, such as the transfer protocol HTTP. For example, the Internet Cookies used by web sites in order to relate an HTTP request to a user profile, thus enabling to return a customized HTML file to the user, are correlation information. Besides being useful to implement stateful communication on top of stateless protocols, correlation is also a flexible and user programmable mechanism for managing loosely-coupled, multiparty conversations. Indeed, correlation data can be communicated to other partners in order to allow them to join a conversation or to delegate the task of carrying out an ongoing conversation.

In this paper, we show another evidence of the flexibility of the correlation mechanism that involves creation of and joining conversations. More specifically, we demonstrate how correlation allows a partner to asynchronously join an existing conversation without need to know in advance any information about the conversation itself, such as e.g. its identifier or the other participants. Since this particular kind of conversation joining is completely transparent to participants we call it *blind-date joining*. It can be used in various domains like e-commerce, events organization, bonus payments, gift lists, etc. In the context of e-commerce, for instance, a social shopping provider (e.g., Groupon [1]) can activate deals only if a certain number of buyers adhere. In this scenario, the blind-date joining can be used to activate these deals. As another example, blind-date joining can be used to organize an event only if a given number of participants is reached. Specifically, the organizers can wait for the right amount of participants and, when this is reached, they send the invitation with the location details to each of them.

We illustrate the blind-date joining strategy by exploiting a simple but realistic case study from the online games domain. We then implement the case study via a well-established orchestration language for web services, i.e. the OASIS standard WS-BPEL [21]. To better clarify and formally present the semantics of the blind-date joining strategy we also introduce a specification of the case study, and its step-by-step temporal evolution, using the process calculus COWS [16, 22], a formalism specifically designed for specifying and combining SOC applications, while modelling their dynamic behaviour.

The rest of the paper is structured as follows. Section 2 surveys syntax and semantics of WS-BPEL and COWS. Section 3 presents how blind-date conversation can be expressed in WS-BPEL by implementing a case study from the online games domain. Section 4 formally describes, by using COWS, the creation and joining phase of blind-date conversations. Finally, Section 5 concludes the paper by also reviewing more closely related work.

2 A glimpse of WS-BPEL and COWS

This section presents a survey of WS-BPEL and COWS. The overview of WS-BPEL gives a high-level description of the aspects captured by its linguistic constructs. Due to lack of space, the overview of COWS gives only a glimpse of its semantics, a full account of which can be found in [22].

2.1 An overview of WS-BPEL

WS-BPEL [21] is essentially a linguistic layer on top of WSDL for describing the structural aspects of Web Services 'orchestration', i.e. the process of combining and coordinating different Web Services to obtain a new, customised service. In practice, and briefly, WSDL [9] is a W3C standard that permits to express the functionalities offered and required by web services by defining, akin object interfaces in

Object-Oriented Programming, the structure of input and output messages of operations.

In WS-BPEL, the logic of interaction between a service and its environment is described in terms of structured patterns of communication actions composed by means of control flow constructs that enable the representation of complex structures. For the specification of orchestration, WS-BPEL provides many different activities that are distinguished between *basic activities* and *structured activities*. Orchestration exploits state information that is stored in variables and is managed through message correlation. In fact, when messages are sent/received, the value of their parameters is stored in variables. Likewise block structured languages, the scope of variables extends to the whole immediately enclosing `<scope>`, or `<process>`, activity (whose meaning is clarified below).

The basic activities are: `<invoke>`, to invoke an operation offered by a Web Service; `<receive>`, to wait for an invocation to arrive; `<reply>`, to send a message in reply to a previously received invocation; `<wait>`, to delay execution for some amount of time; `<assign>`, to update the values of variables with new data; `<throw>`, to signal internal faults; `<exit>`, to immediately end a service instance; `<empty>`, to do nothing; `<compensate>` and `<compensateScope>`, to invoke compensation handlers; `<rethrow>`, to propagate faults; `<validate>`, to validate variables; and `<extensionActivity>`, to add new activity types. Notably, `<reply>` can be combined with `<receive>` to model synchronous request-response interactions.

The structured activities describe the control flow logic of a business process by composing basic and/or structured activities recursively. The structured activities are: `<sequence>`, to execute activities sequentially; `<if>`, to execute activities conditionally; `<while>` and `<repeatUntil>`, to repetitively execute activities; `<flow>`, to execute activities in parallel; `<pick>`, to execute activities selectively; `<forEach>`, to (sequentially or in parallel) execute multiple activities; and `<scope>`, to associate handlers for exceptional events to a primary activity. Activities within a `<flow>` can be further synchronised by means of *flow links*. These are conditional transitions connecting activities to form directed acyclic graphs and are such that a target activity may only start when all its source activities have completed and the condition on the incoming flow links evaluates to true.

The handlers within a `<scope>` can be of four different kinds: `<faultHandler>`, to provide the activities in response to faults occurring during execution of the primary activity; `<compensationHandler>`, to provide the activities to compensate the successfully executed primary activity; `<terminationHandler>`, to control the forced termination of the primary activity; and `<eventHandler>`, to process message or timeout events occurring during execution of the primary activity. If a fault occurs during execution of a primary activity, the control is transferred to the corresponding fault handler and all currently running activities inside the scope are interrupted immediately without involving any fault/compensation handling behaviour. If another fault occurs during a fault/compensation handling, then it is re-thrown, possibly, to the immediately enclosing scope. Compensation handlers attempt to reverse the effects of previously successfully completed primary activities (scopes) and have been introduced to support Long-Running (Business) Transactions (LRTs). Compensation can only be invoked from within fault or compensation handlers starting the compensation either of a specific inner (completed) scope, or of all inner completed scopes in the reverse order of completion. The latter alternative is also called the *default* compensation behaviour. Invoking a compensation handler that is unavailable is equivalent to perform an empty activity.

A WS-BPEL program, also called *(business) process*, is a `<process>`, that is a sort of `<scope>` without compensation and termination handlers.

WS-BPEL uses the basic notion of *partner link* to directly model peer-to-peer relationships between services. Such a relationship is expressed at the WSDL level by specifying the roles played by each of the services in the interaction. This information, however, does not suffices to deliver messages to a

service. Indeed, since multiple instances of the same service can be simultaneously active because service operations can be independently invoked by several clients, messages need to be delivered not only to the correct partner, but also to the correct instance of the service that the partner provides. To achieve this, WS-BPEL relies on the business data exchanged rather than on specific mechanisms, such as WS-Addressing [11] or low-level methods based on SOAP headers. In fact, WS-BPEL exploits *correlation sets*, namely sets of *correlation variables* (called *properties*), to declare the parts of a message that can be used to identify an instance. This way, a message can be delivered to the correct instance on the basis of the values associated to the correlation variables, independently of any routing mechanism.

2.2 An overview of COWS

COWS [16, 22] is a formalism for modelling (and analysing) service-oriented applications. It provides a novel combination of constructs and features borrowed from well-known process calculi such as non-binding receiving activities, asynchronous communication, polyadic synchronization, pattern matching, protection, and delimited receiving and killing activities. As a consequence of its careful design, the calculus makes it easy to model many important aspects of service orchestrations *à la* WS-BPEL, such as service instances with shared state, services playing more than one partner role, stateful conversations made by several correlated service interactions, and long-running transactions. For the sake of simplicity, we present here a fragment of COWS (called μCOWS in [22]) without linguistic constructs for dealing with 'forced termination', since such primitives are not used in this work.

The syntax of COWS is presented in Table 1. We use two countable disjoint sets: the set of *values* (ranged over by v, v', ...) and the set of 'write once' *variables* (ranged over by x, y, ...). The set of values is left unspecified; however, we assume that it includes the set of *names* (ranged over by n, m, p, o, ...) mainly used to represent partners and operations. We also use a set of *expressions* (ranged over by ε), whose exact syntax is deliberately omitted; we just assume that expressions contain, at least, values and variables. As a matter of notation, w ranges over values and variables and u ranges over names and variables. Notation $\bar{\cdot}$ stands for tuples, e.g. $\bar{x}$ means $\langle x_1, \ldots, x_n \rangle$ (with $n \geq 0$) where variables in the same tuple are pairwise distinct. We will omit trailing occurrences of $\mathbf{0}$, writing e.g. $p \bullet o?\bar{w}$ instead of $p \bullet o?\bar{w}.\mathbf{0}$, and write $[\langle u_1, \ldots, u_n \rangle]\, s$ in place of $[u_1] \ldots [u_n]\, s$. We will write $I \triangleq s$ to assign a name I to the term s.

Services are structured activities built from basic activities, i.e. the empty activity $\mathbf{0}$, the invoke activity $\_ \bullet \_!\_$ and the receive activity $\_ \bullet \_?\_$, by means of prefixing $\_.\_$, choice $\_ + \_$, parallel composition $\_ \mid \_$, delimitation $[\_]\, \_$ and replication $* \_$. We adopt the following conventions about the operators precedence: monadic operators bind more tightly than parallel composition, and prefixing more tightly than choice.

Invoke and *receive* are the communication activities, which permit invoking an operation offered by a service and waiting for an invocation to arrive, respectively. Besides output and input parameters, both activities indicate an *endpoint*, i.e. a pair composed of a partner name p and an operation name o, through which communication should occur. An endpoint $p \bullet o$ can be interpreted as a specific implementation of operation o provided by the service identified by the logic name p. An invoke $p \bullet o!\langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ can proceed as soon as all expression arguments ε_1, ..., ε_n can be successfully evaluated to values. A receive $p \bullet o?\langle w_1, \ldots, w_n \rangle.s$ offers an invocable operation o along a given partner name p and thereafter (due to the *prefixing* operator) the service continues as s. An inter-service communication between these two activities takes place when the tuple of values $\langle v_1, \ldots, v_n \rangle$, resulting from the evaluation of the invoke argument, matches the template $\langle w_1, \ldots, w_n \rangle$ argument of the receive. This causes a substitution of the variables in the receive template (within the scope of variables declarations) with the corresponding values produced by the invoke. Partner and operation names can be exchanged in communication, thus enabling many different interaction patterns among service instances. However, dynamically re-

Expressions: $\varepsilon, \varepsilon', \ldots$
Values: $v, v', \ldots$ — *Variables/Values:* $w, w', \ldots$
Names: $n, m, \ldots$ — *Variables/Names:* $u, u', \ldots$
Partners: $p, p', \ldots$
Operations: $o, o', \ldots$
Variables: $x, y, \ldots$

Services:

s ::=		
	$u \bullet u'!\bar{\varepsilon}$	(invoke)
$\mid$	g	(receive-guarded choice)
$\mid$	$s \mid s$	(parallel composition)
$\mid$	$[u]\,s$	(delimitation)
$\mid$	$*s$	(replication)

Receive-guarded choice:

g ::=		
	$\mathbf{0}$	(empty)
$\mid$	$p \bullet o?\bar{w}.s$	(receive)
$\mid$	$g + g$	(choice)

Table 1: COWS syntax

ceived names cannot form the endpoints used to receive further invocations. Indeed, endpoints of receive activities are identified statically because their syntax only allows using names and not variables.

The *empty* activity does nothing, while *choice* permits selecting for execution one between two alternative receives.

Execution of *parallel* services is interleaved. However, if more matching receives are ready to process a given invoke, only one of the receives that generate a substitution with smallest domain (see [22] for further details) is allowed to progress (namely, this receive has an higher priority to proceed than the others). This mechanism permits to model the precedence of a service instance over the corresponding service specification when both can process the same request (see also Section 4).

Delimitation is the only binding construct: $[u]\,s$ binds the element u in the scope s. It is used for two different purposes: to regulate the range of application of substitutions produced by communication, if the delimited element is a variable, and to generate fresh names, if the delimited element is a name.

Finally, the *replication* construct $*s$ permits to spawn in parallel as many copies of s as necessary. This, for example, is exploited to implement recursive behaviours and to model business process definitions, which can create multiple instances to serve several requests simultaneously.

3 Blind-date conversations in WS-BPEL

We present here a (web) service capable of arranging matches of 4-players[1] online (card) games, such as e.g. *burraco* or *canasta*. To create or join a match, a player has only to indicate the kind of game he/she would like to play and his/her endpoint reference (i.e., his/her address). Thus, players do not need to know in advance any further information, such as the identifier of the table or the identifiers of other players. Moreover, players do not either know if a new match is created as a consequence of their request to play, or if they join an already existing match. Therefore, the arrangement of tables is *completely transparent* to players.

Figure 1 shows a graphical representation[2] We report below the code of the process where, to sim-

[1] It is worth noticing that the blind-date conversation approach works as well with a number of players not fixed a priori. Of course, this would require using some extra activities. Therefore, to better highlight the specificities of the proposed approach, we have preferred to keep the example as simple as possible and, hence, we have fixed the number of players to four.

[2] The representation has been created by using Eclipse BPEL Designer (`http://www.eclipse.org/bpel/`).

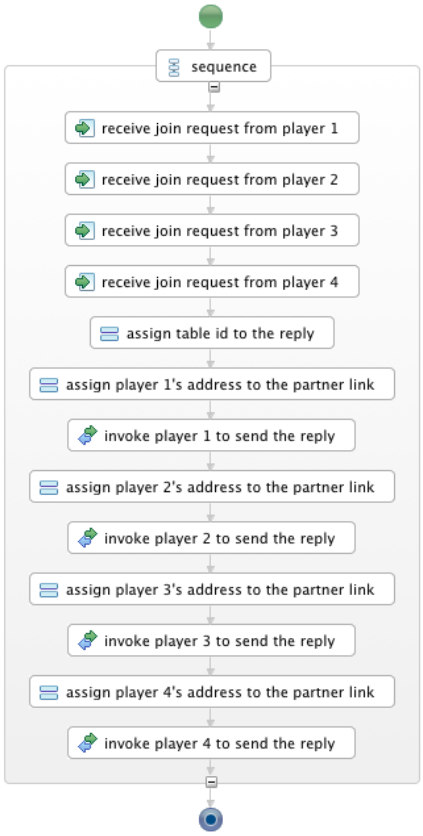

Figure 1: Graphical representation of the WS-BPEL process `TableManager`

plify the reading of the code, we have omitted irrelevant details and highlighted the basic activities `<receive>`, `<invoke>` and `<assign>` by means of a grey background.

```
<process name="TableManager" ... >
<partnerLinks>
  <partnerLink name="tableManager"
               myRole="table" partnerRole="player"
               initializePartnerRole="no"
               partnerLinkType="tableManager:table"/>
</partnerLinks>
<variables>
  <variable messageType="tableManager:request" name="Player1"/>
  <variable messageType="tableManager:request" name="Player2"/>
  <variable messageType="tableManager:request" name="Player3"/>
  <variable messageType="tableManager:request" name="Player4"/>
  <variable messageType="tableManager:reply" name="PlayersReply"/>
</variables>
<correlationSets>
  <correlationSet name="GameCorrelationSet"
                  properties="tableManager:gameNameProperty"/>
</correlationSets>
```

```
<sequence>
  <receive partnerLink="tableManager"
           operation="join"
           variable="Player1"
           createInstance="yes">
        <correlations>
          <correlation initiate="yes" set="GameCorrelationSet"/>
        </correlations>
  </receive>
  <receive partnerLink="tableManager"
           operation="join"
           variable="Player2">
        <correlations>
          <correlation initiate="no" set="GameCorrelationSet"/>
        </correlations>
  </receive>
  <receive partnerLink="tableManager"
           operation="join"
           variable="Player3">
        <correlations>
          <correlation initiate="no" set="GameCorrelationSet"/>
        </correlations>
  </receive>
  <receive partnerLink="tableManager"
           operation="join"
           variable="Player4">
        <correlations>
          <correlation initiate="no" set="GameCorrelationSet"/>
        </correlations>
  </receive>
  <assign>
    ...
    <copy ...>
      <from>getTableID()</from>
      <to>$PlayersReply.payload/tableManager:tableID</to>
    </copy>
  </assign>
  <assign>
    <copy>
      <from>$Player1.payload/tableManager:playerAddress</from>
      <to partnerLink="tableManager"/>
    </copy>
  </assign>
  <invoke inputVariable="PlayersReply"
          operation="start"
          partnerLink="tableManager"/>
  <assign>
    <copy>
      <from>$Player2.payload/tableManager:playerAddress</from>
      <to partnerLink="tableManager"/>
    </copy>
  </assign>
  <invoke inputVariable="PlayersReply"
          operation="start"
          partnerLink="tableManager"/>
```

```
  <assign>
    <copy>
      <from>$Player3.payload/tableManager:playerAddress</from>
      <to partnerLink="tableManager"/>
    </copy>
  </assign>
  <invoke inputVariable="PlayersReply"
          operation="start"
          partnerLink="tableManager"/>
  <assign>
    <copy>
      <from>$Player4.payload/tableManager:playerAddress</from>
      <to partnerLink="tableManager"/>
    </copy>
  </assign>
  <invoke inputVariable="PlayersReply"
          operation="start"
          partnerLink="tableManager"/>
  </sequence>
</process>
```

The process uses only one partner link, namely `tableManager`, that provides two roles: `table` and `player`. The former is used by the process to receive the players' requests, while the latter is used by players to receive the table identifier. Five variables are used for storing data of the exchanged messages: one for each player request and one for the manager response. Notably, the used message style[3] is *document*, thus messages are formed by a single part, called `payload`, that contains all message data. Therefore, we use XPath expressions of the form `$VariableName.payload/Path` to extract or store data in message variables.

The process starts with a `<receive>` activity waiting for a message from a player; the message contains a request (stored in the variable `Player1`) to participate to a match of a given game. Whenever prompted by a player's request, the process creates an instance (see the option `createInstance="yes"` in the first `<receive>`), corresponding to a new (virtual) card-table of the game specified by the player, and is immediately ready to concurrently serve other requests. Service instances are indeed the WS-BPEL counterpart of inter-service conversations. In order to deliver each request to an existing instance corresponding to a table of the requested game (if there exists one), the name of the game is used as correlation datum. Thus, each `<receive>` activity specifies the correlation set `GameCorrelationSet`, which is instantiated by the initial `<receive>`, in order to receive only requests for the same game indicated by the first request. The `<property>` defining the correlation set is declared in the WSDL document associated to the process as follows:

```
<prop:property name="gameNameProperty" type="xs:string"/>

<prop:propertyAlias messageType="this:request" part="payload"
                    propertyName="this:gameNameProperty">
  <prop:query queryLanguage=...>//this:RequestElement/this:gameName</prop:query>
</prop:propertyAlias>
```

[3] The SOAP message style configuration is specified in the *binding* section of the WSDL document associated to the WS-BPEL process. We have preferred to use the document-style rather than the RPC-style, because the former minimizes coupling between the interacting parties.

A <property> specifies an element of a correlation set and relies on one (or more) <property-Alias> to identify correlation values inside messages. In our specification, the <propertyAlias> extracts from <request> messages the needed element by using an XPath <query>. Then, the correlation set GameCorrelationSet is defined by the property gameNameProperty that identifies the string element gameName of the messages sent by players.

Once the initial <receive> is executed and an instance is created, other three <receive> activities are sequentially performed by such instance, in order to complete the card-table for the new match. Notice that the correlation mechanism ensures that only players that want to play the same game are put together in a table.

When four players join a conversation for a new match (which, in WS-BPEL, corresponds to a process instance), a unique table identifier is generated, by means of the custom XPath expression getTableID(), and inserted into the variable PlayersReply. This variable contains the message that will be sent back to each player via four <invoke> activities using the tableManager partner link. Before every <invoke>, an activity <assign> is executed to extract (by means of an XPath expression) the endpoint reference of the player, contained within the player's request, and to store it into the partnerRole of the tableManager partner link. These assignments allow the process to properly reply in an asynchronous way to the players.

Now, the new table is arranged and, therefore, the players can start to play by using the received table identifier and by interacting with another service dedicated to this purpose (which, of course, is out of the scope of this work).

Experimenting with the WS-BPEL process. This WS-BPEL process can be experimented via the Web interface at http://reggae.dsi.unifi.it/blinddatejoining/, or by downloading from the same address the source and binary code.

More specifically, the process is configured to be deployed in a WS-BPEL engine Apache ODE [3]. Indeed, we have equipped the process with the corresponding WSDL and deployment descriptor files, which provide typing and binding information, respectively. Notably, in order to call the custom XPath expression getTableID() within the process, its definition must be previously installed in the ODE engine as a Java library. For the sake of simplicity, we have defined such expression as a random function.

To experiment with the WS-BPEL program, we have developed a sort of testing environment consisting of a few Java classes implementing the service clients. Such classes rely on the artifacts automatically generated by JAX-WS [10] from the WSDL document of the process, and simply exchange SOAP messages with the process. These clients are instantiated and executed by a Web application developed by using the Play framework [25]; a screenshot of such application is shown in Figure 2. Notably, players created in a given browser session could be assigned to tables together with players created in other sessions.

4 Semantics mechanisms underlying blind-date conversation joining

In this section, to clarify the behaviour at runtime of the TableManager process (and of its instances), we formally specify a scenario involving the manager service by means of the process calculus COWS. The aim is to shed light on the effect of the blind-date joining and to show how it can be easily programmed through the correlation approach. Moreover, this formal account also permits clarifying the mechanisms underlying message correlation (i.e., shared input variables, pattern-matching and priority among receive activities).

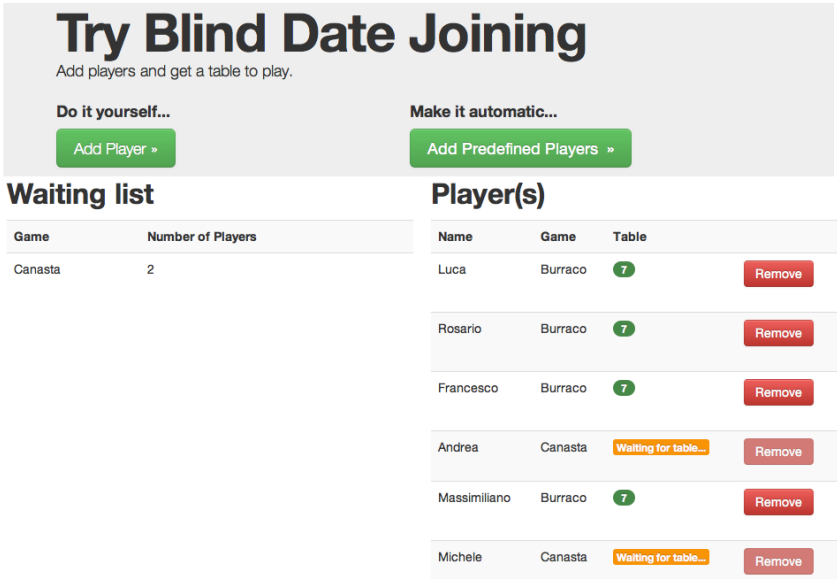

Figure 2: A screenshot of the Web application for experimenting with the process `TableManager`

The `TableManager` process can be rendered in COWS as follows:

$$
\begin{aligned}
\mathit{TableManagerProcess} \triangleq\ * [x_{game}, x_{player1}, x_{player2}, x_{player3}, x_{player4}] \\
& \mathit{manager} \bullet \mathit{join}?\langle x_{game}, x_{player1} \rangle . \\
& \mathit{manager} \bullet \mathit{join}?\langle x_{game}, x_{player2} \rangle . \\
& \mathit{manager} \bullet \mathit{join}?\langle x_{game}, x_{player3} \rangle . \\
& \mathit{manager} \bullet \mathit{join}?\langle x_{game}, x_{player4} \rangle . \\
& \quad [\mathit{tableId}] \, (\, x_{player1} \bullet \mathit{start}!\langle \mathit{tableId} \rangle \\
& \qquad \mid x_{player2} \bullet \mathit{start}!\langle \mathit{tableId} \rangle \\
& \qquad \mid x_{player3} \bullet \mathit{start}!\langle \mathit{tableId} \rangle \\
& \qquad \mid x_{player4} \bullet \mathit{start}!\langle \mathit{tableId} \rangle \,)
\end{aligned}
$$

The replication operator $*$ specifies that the above term represents a service definition, which acts as a persistent service capable of creating multiple instances to simultaneously serve concurrent requests. The delimitation operator $[\,]$ declares the scope of variables x_{game} and $x_{player\,i}$, with $i = 1..4$. The endpoint $\mathit{manager} \bullet \mathit{join}$ (composed by the partner name *manager* and the operation *join*) is used by the service to receive join requests from four players. When sending their request, the players are required to provide only the kind of game, stored in x_{game}, and their partner names, stored in $x_{player\,i}$, that they will then use to receive the table identifier. Player's requests are received through receive activities of the form $\mathit{manager} \bullet \mathit{join}?\langle x_{game}, x_{player\,i} \rangle$, which are correlated by means of the shared variable x_{game}. Then, the delimitation operator is used to create a fresh name *tableId* that represents an unique table identifier. Such identifier will be then communicated to each player by means of four invoke activities $x_{player\,i} \bullet \mathit{start}!\langle \mathit{tableId} \rangle$. Notably, differently from the WS-BPEL specification of the process, in the COWS definition the assign activities are not necessary, because their role is played by the substitutions generated by the interactions along the endpoint $\mathit{manager} \bullet \mathit{join}$.

Consider now the following system

$$Luca \mid Rosario \mid Francesco \mid \ldots \mid TableManagerProcess$$

where the players are defined as follows

$$Luca \triangleq manager \bullet join!\langle burraco, p_L\rangle \mid [x_{id}]\, p_L \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Luca\rangle$$

$$Rosario \triangleq manager \bullet join!\langle canasta, p_R\rangle \mid [x_{id}]\, p_R \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Rosario\rangle$$

$$Francesco \triangleq manager \bullet join!\langle burraco, p_F\rangle \mid [x_{id}]\, p_F \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Francesco\rangle$$

$$\ldots$$

If *Luca* requests to join a match, since there are not tables under arrangement, *TableManager-Process* initialises a new match instance (highlighted by grey background) and the system evolves to:

$$\begin{array}{l}
[x_{id}]\, p_L \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Luca\rangle \\
\mid Rosario \mid Francesco \mid \ldots \mid TableManagerProcess \\
\mid\ [x_{player2}, x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle burraco, x_{player2}\rangle. \\
\quad manager \bullet join?\langle burraco, x_{player3}\rangle. \\
\quad manager \bullet join?\langle burraco, x_{player4}\rangle. \\
\qquad [tableId]\, (\, p_L \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player2} \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player3} \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player4} \bullet start!\langle tableId\rangle\,)
\end{array}$$

Now, if *Rosario* invokes *TableManagerProcess*, a second match instance (highlighted by dark grey background) is created:

$$\begin{array}{l}
[x_{id}]\, p_L \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Luca\rangle \\
\mid\ [x_{id}]\, p_R \bullet start?\langle x_{id}\rangle.\, \langle \text{rest of } Rosario\rangle \\
\mid Francesco \mid \ldots \mid TableManagerProcess \\
\mid\ [x_{player2}, x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle burraco, x_{player2}\rangle. \\
\quad manager \bullet join?\langle burraco, x_{player3}\rangle. \\
\quad manager \bullet join?\langle burraco, x_{player4}\rangle. \\
\qquad [tableId]\, (\, p_L \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player2} \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player3} \bullet start!\langle tableId\rangle \\
\qquad\qquad \mid x_{player4} \bullet start!\langle tableId\rangle\,)
\end{array}$$

$$
\begin{array}{l}
\mid\ [x_{player2}, x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle canasta, x_{player2}\rangle . \\
\quad manager \bullet join?\langle canasta, x_{player3}\rangle . \\
\quad manager \bullet join?\langle canasta, x_{player4}\rangle . \\
\quad\quad [tableId'] \, (\, p_R \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player2} \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player3} \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player4} \bullet start!\langle tableId'\rangle\,)
\end{array}
$$

When *Francesco* invokes *TableManagerProcess*, the process definition and the first created instance, being both able to receive the same message $\langle burraco, p_F\rangle$ along the endpoint $manager \bullet join$, compete for the request $manager \bullet join!\langle burraco, p_F\rangle$. COWS's (prioritized) semantics precisely establishes how this sort of race condition is dealt with: only the existing instance is allowed to evolve, as required by WS-BPEL. This is done through the dynamic prioritised mechanism of COWS, i.e. assigning the receives performed by instances (having a more defined pattern and requiring less substitutions) a greater priority than the receives performed by a process definition. In fact, in the above COWS term, the first instance can perform a receive matching the message and containing only one variable in its argument, while the initial receive of *TableManagerProcess* contains two variables. In this way, the creation of a new instance is prevented. Moreover, pattern-matching permits delivering the request to the appropriate instance, i.e. that corresponding to a *burraco* match. Therefore, the only feasible computation leads to the following term

$$
\begin{array}{l}
[x_{id}]\, p_L \bullet start?\langle x_{id}\rangle . \langle \text{rest of } Luca\rangle \\
\mid\ [x_{id}]\, p_R \bullet start?\langle x_{id}\rangle . \langle \text{rest of } Rosario\rangle \\
\mid\ [x_{id}]\, p_F \bullet start?\langle x_{id}\rangle . \langle \text{rest of } Francesco\rangle \\
\mid \ldots \mid\ TableManagerProcess \\
\mid\ [x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle burraco, x_{player3}\rangle . \\
\quad manager \bullet join?\langle burraco, x_{player4}\rangle . \\
\quad\quad [tableId] \, (\, p_L \bullet start!\langle tableId\rangle \\
\quad\quad\quad\quad \mid\ p_F \bullet start!\langle tableId\rangle \\
\quad\quad\quad\quad \mid\ x_{player3} \bullet start!\langle tableId\rangle \\
\quad\quad\quad\quad \mid\ x_{player4} \bullet start!\langle tableId\rangle\,) \\
\mid\ [x_{player2}, x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle canasta, x_{player2}\rangle . \\
\quad manager \bullet join?\langle canasta, x_{player3}\rangle . \\
\quad manager \bullet join?\langle canasta, x_{player4}\rangle . \\
\quad\quad [tableId'] \, (\, p_R \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player2} \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player3} \bullet start!\langle tableId'\rangle \\
\quad\quad\quad\quad \mid\ x_{player4} \bullet start!\langle tableId'\rangle\,)
\end{array}
$$

where *Francesco* joined to the *burraco* table under arrangement.

Eventually, with the arrival of other requests from players that want to play *burraco*, the *TableManagerProcess* completes to arrange the *burraco* table and contacts the players for communicating them the table identifier:

$$
\begin{array}{l}
[tableId]\,(\,\langle \text{rest of } Luca\rangle \mid \langle \text{rest of } Francesco\rangle \mid \ldots) \\
\mid\ [x_{id}]\,p_R \bullet start?\langle x_{id}\rangle.\,\langle \text{rest of } Rosario\rangle \\
\mid\ \ldots \mid TableManagerProcess \\
\mid\ [x_{player2}, x_{player3}, x_{player4}] \\
\quad manager \bullet join?\langle canasta, x_{player2}\rangle. \\
\quad manager \bullet join?\langle canasta, x_{player3}\rangle. \\
\quad manager \bullet join?\langle canasta, x_{player4}\rangle. \\
\qquad [tableId']\,(\,p_R \bullet start!\langle tableId'\rangle \\
\qquad\qquad \mid x_{player2} \bullet start!\langle tableId'\rangle \\
\qquad\qquad \mid x_{player3} \bullet start!\langle tableId'\rangle \\
\qquad\qquad \mid x_{player4} \bullet start!\langle tableId'\rangle\,)
\end{array}
$$

Therefore, the players of table *tableId* (including *Luca* and *Francesco*) can start to play, while *Rosario* continues to wait for other *canasta* players.

Of course, *TableManagerProcess* can be easily tailored to arrange matches for n-players online games with $n \neq 4$ (e.g. poker, bridge, ...). Notably, a player can play concurrently in more than one match. However, for the sake of simplicity, we assume here that a player can ask to play the same game more than once only if each time it waits for the response to the previous request, otherwise it could be assigned more than once to the same table. To make the scenario more realistic, we could add timeouts to receives activities of *TableManagerProcess* and players' services in order to avoid deadlocks. This can be done both in WS-BPEL, by exploiting `onAlarm` events within `pick` activities, and in COWS, by using *wait* activities in conjunction with choice activities (as shown in [22]).

5 Concluding remarks

We have illustrated blind-date conversation joining, a strategy allowing a participant to join a conversation without need to know information about the conversation itself, such as e.g. its identifier or the other participants, but the endpoint of the service provider. This is possible because some values in the request define which conversation the participant will join. In case such a conversation does not exist, a new one associated to these values will be created.

We showed how the blind-date conversation joining strategy can be implemented in the well-established industrial standard WS-BPEL. In order to accomplish this task, we used WS-BPEL correlation sets to filter the client requests and create conversations identified by means of the filtered values.

We also presented a formal semantics for the strategy by using the process calculus COWS. The semantics permits to highlight what the effect of the strategy is, by clarifying the mechanism underlying message correlation and showing how the strategy can be easily programmed in a correlation-based environment. This also permits appreciating the conciseness and cleanness of the COWS specification of the `TableManager` process, especially when compared with its WS-BPEL counterpart.

We developed a case study to show how our strategy can be used in a realistic scenario. The case study defines an online games provider that arranges online matches of 4-players card games. This scenario is implemented using WS-BPEL and executed on Apache ODE. As we showed in Section 2.1, despite the XML markup used by WS-BPEL, the blind-date joining strategy permits to have a smooth and clean implementation. We also equipped the case study with a 'testing environment' in order to facilitate its testing.

Related work. The peculiar form of conversation joining studied in this paper, which we call 'blind-date', originates from the message correlation mechanism used for delivering messages to the appropriate service instances in both orchestration languages and formalisms for SOC. This joining strategy is, at least in principle, independent from the specific language or formalism used to enact it. In this paper, we have used the language WS-BPEL and the formalism COWS, but different choices could have been made. For example, Jolie [20] could be used as correlation-based orchestration language and SOCK [12] as the formalism, being the former a Java-based implementation of the latter. However, our choice fell on WS-BPEL because it is an OASIS open standard well-accepted by industries and, hence, supported by well-established and maintained engines. Instead, COWS has been selected because of its strict correspondence with WS-BPEL while, at the same time, being a core calculus consisting of just a few constructs, which makes it more suitable than WS-BPEL to reason on applications' behaviour.

In the SOC literature, two main approaches have been considered to connect the interaction protocols of clients and of the respective service instances. That based on the correlation mechanism was first exploited in [24] where, however, only interaction among different instances of a single service are taken into account. Another correlation-based formalism, besides COWS and SOCK mentioned above, is the calculus *Corr* [18], which is a sort of value-passing CCS without restriction and enriched with constructs for expressing services and their instances. *Corr* has been specifically designed to capture behaviours related to correlation aspects in a specific WS-BPEL engine (namely, ActiveBPEL [2]). Another work with an aim similar to ours, i.e. to show an exploitation of the correlation-based mechanism for dealing with issues raised by practical scenarios, is presented in [17]. Such work proposes an implementation of a correlation-based primitive allowing messages to be delivered to more than one service conversation. Anyway, WS-BPEL (and COWS as well) natively provides such kind of broadcast primitive.

A large strand of work, instead, relies on the explicit modelling of interaction *sessions* and their dynamic creation. A session corresponds to a private channel (*à la* π-calculus [19]) which is implicitly instantiated when calling a service: it binds caller and callee and is used for their future conversation. Although this the technology underling SOC, its abstraction level has proved convenient for reasoning about SOC applications. Indeed, session-based conversation can be regulated by so called *session types* that can statically guarantee a number of desirable properties, such as communication safety, progress, predictability. Therefore, an important group of calculi for modeling and proving properties of services is based on the explicit notion of interaction session. Most of such work (among which we mention [13, 7, 15, 4]) has been devoted to study *dyadic* sessions, i.e. interaction sessions between only two participants.

Recently, another body of work (as, e.g., [14, 5, 6, 8]) focussed on a more general form of sessions, called *multiparty* sessions/conversations, which are closer to the notion of conversation that we have considered in this paper. The multiparty session approach proposed in [14] permits expressing a conversation by means of channels shared among the participants. However, the conversation is created through a single synchronization among all participants, whose number is fixed at design time. This differs from our notion of blind-date joining, where once a conversation has been created the participants can asyn-

chronously join. Moreover, although in our case study we consider a conversation of five participants, in principle this number can change at runtime. The μse language [5] permits the declaration of multiparty sessions in a way transparent to the user. However, rather than relying on the correlation mechanism, as in our approach, μse creates multiparty conversations by using a specific primitive that permits to merge together previously created conversations. Instead, when relying on correlation, the conversation merging is automatically performed for each correlated request. Another formalism dealing with multiparty interactions is the Conversation Calculus [23, 6]. It uses the conversation-based mechanism for inter-session communication, which permits to progressively accommodate and dismiss participants from the same conversation. This is realized by means of named containers for processes, called conversation contexts. However, processes in unrelated conversations cannot interact directly. Moreover, conversation joining is not transparent to a new participant, because he has to know the name of the conversation. Instead, in our correlation-based approach, a conversation is represented by a service instance, which is not accessed via an identifier, but via the correlation values specified by the correlation set.

In conclusion, the lower-level mechanism based on correlation sets, that exploits business data to correlate different interactions, is more robust and fits the loosely-coupled world of Web Services better than that based on explicit session references. It turned out to be powerful and flexible enough to fulfill the need of SOC, e.g. it easily allows a single message to participate in multiple conversations, each (possibly) identified by separate correlation values, which instead requires non-trivial workarounds by using the session-based approach. It is not a case that also the standard WS-BPEL uses correlation sets.

Future work. We plan to study and develop type theories for describing and regulating the correlation mechanism. This will permit to guarantee desired properties like those defined for dyadic and multiparty session types. We expect that to develop type theories for ensuring properties of the correlation mechanism be more difficult than doing so for session-based interaction because the communication media are not necessarily private and the correlation values are determined at runtime.

References

[1] *Groupon.* Web site: `http://www.groupon.com/`.

[2] Active Endpoints (2008): *ActiveBPEL 5.0.2*. Available at `http://www.activevos.com/community-open-source.php`.

[3] Apache Software Foundation (2011): *Apache ODE 1.3.5*. Available at `http://ode.apache.org/`.

[4] M. Boreale, R. Bruni, R. De Nicola & M. Loreti (2008): *Sessions and Pipelines for Structured Service Programming*. In: *FMOODS*, *LNCS* 5051, Springer, pp. 19–38.

[5] R. Bruni, I. Lanese, H.C. Melgratti & E. Tuosto (2008): *Multiparty Sessions in SOC*. In: *COORDINATION*, *LNCS* 5052, Springer, pp. 67–82.

[6] L. Caires & H.T. Vieira (2010): *Conversation types*. *Theor. Comput. Sci.* 411(51-52), pp. 4399–4440.

[7] M. Carbone, K. Honda & N. Yoshida (2007): *Structured Communication-Centred Programming for Web Services*. In: *ESOP*, *LNCS* 4421, Springer, pp. 2–17.

[8] M. Carbone & F. Montesi (2013): *Deadlock-freedom-by-design: multiparty asynchronous global programming*. In: *POPL*, ACM, pp. 263–274.

[9] E. Christensen, F. Curbera, G. Meredith & S. Weerawarana (2001): *Web Services Description Language (WSDL) 1.1*. Technical Report, W3C. Available at `http://www.w3.org/TR/wsdl/`.

[10] GlassFish community (2012): *JAX-WS 2.2.7*. Available at `http://jax-ws.java.net/`.

[11] M. Gudgin, M. Hadley & T. Rogers (2006): *Web Services Addressing 1.0 - Core*. Technical Report, W3C. W3C Recommendation.

[12] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi & G. Zavattaro (2006): *SOCK: A Calculus for Service Oriented Computing*. In: *ICSOC*, *LNCS* 4294, Springer, pp. 327–338.

[13] K. Honda, V. T. Vasconcelos & M. Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *ESOP*, *LNCS* 1381, Springer, pp. 122–138.

[14] K. Honda, N. Yoshida & M. Carbone (2008): *Multiparty asynchronous session types*. In: *POPL*, ACM Press, pp. 273–284.

[15] I. Lanese, F. Martins, A. Ravara & V.T. Vasconcelos (2007): *Disciplining Orchestration and Conversation in Service-Oriented Computing*. In: *SEFM*, IEEE Computer Society Press, pp. 305–314.

[16] A. Lapadula, R. Pugliese & F. Tiezzi (2007): *A Calculus for Orchestration of Web Services*. In: *ESOP*, *LNCS* 4421, Springer, pp. 33–47.

[17] J. Mauro, M. Gabbrielli, C. Guidi & F. Montesi (2011): *An Efficient Management of Correlation Sets with Broadcast*. In: *COORDINATION*, *LNCS* 6721, Springer, pp. 80–94.

[18] H.C. Melgratti & C. Roldán (2012): *On Correlation Sets and Correlation Exceptions in ActiveBPEL*. In: *TGC*, *LNCS* 7173, Springer, pp. 212–226.

[19] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, I and II*. *Information and Computation* 100(1), pp. 1–40, 41–77.

[20] F. Montesi, C. Guidi, R. Lucchi & G. Zavattaro (2007): *JOLIE: a Java Orchestration Language Interpreter Engine*. In: *MTCoord*, *Electronic Notes in Theoretical Computer Science* 181, Elsevier, pp. 19–33.

[21] OASIS WSBPEL TC (2007): *Web Services Business Process Execution Language Version 2.0*. Technical Report, OASIS. Available at `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[22] R. Pugliese & F. Tiezzi (2012): *A calculus for orchestration of web services*. *J. Applied Logic* 10(1), pp. 2–31.

[23] H.T. Vieira, L. Caires & J. C. Seco (2008): *The Conversation Calculus: A Model of Service-Oriented Computation*. In: *ESOP*, *LNCS* 4960, Springer, pp. 269–283.

[24] M. Viroli (2004): *Towards a Formal Foundation to Orchestration Languages*. In: *WS-FM*, *Electronic Notes in Theoretical Computer Science* 105, Elsevier, pp. 51–71.

[25] Zenexity and Typesafe (2013): *Play framework 2.1.0*. Available at `http://www.playframework.com/`.

Proving Properties of Rich Internet Applications

James Smith
Imperial College
London, United Kingdom
jecs@imperial.ac.uk

Abstract

We introduce application layer specifications, which allow us to reason about the state and transactions of rich Internet applications. We define variants of the state/event based logic UCTL* along with two example applications to demonstrate this approach, and then look at a distributed, rich Internet application, proving properties about the information it stores and disseminates. Our approach enables us to justify proofs about abstract properties that are preserved in the face of concurrent, networked inputs by proofs about concrete properties in an Internet setting. We conclude that our approach makes it possible to reason about the programs and protocols that comprise the Internet's application layer with reliability and generality.

1 Introduction

A glance at the existing literature shows that the Internet's transport layer is well understood from a mathematical point of view, with the inductive analysis of the TLS protocol [15] being perhaps the best example. On the other hand, the application layer that sits above it encompases a plethora of languages and frameworks, and understanding seems to be incomplete. At the highest level, the behaviour of Internet programs and protocols has been analysed [4, 13, 11], often for security reasons, but nevertheless there appears to be much work still to do.

These days, Internet applications are made up of both server side and client side parts. These so called "rich" Internet applications are now so commonplace as to be the norm. Any formalism that helps us to understand them mathematically seems a worthy goal, therefore. A complete formal treatment would include a formalisation of JavaScript [10] and much more, however, so here we attempt a loss lofty goal, namely giving formal specifications and proofs for that part of their behaviour related to communication over HTTP. To give an example, consider a "shoot-em-up" style game implemented in JavaScript and run in a web page. The functionality related to the unfolding of the gameplay would encompass game logic and the rendering of each scene, functionality not directly related to the Internet. Suppose the game includes a high score table, however, with high scores stored on the server. Maintaining this table would encompass issues relating to concurrency and communication over HTTP. It is the formal specifications and proofs for this kind of behaviour that is the subject of this paper.

Our motivation in studying this kind of behaviour is a broad understanding of the programs and protocols that comprise the Internet's application layer. With this in mind, this paper demonstrates that it is possible to formalise certain properties of rich Internet applications using a combination of mathematical models, logics and related techniques, which we call collectively application layer specifications. The approach is based on the conventional HTTP request-response mechanism whose participants are usually the web server on the one hand and the

web browser on the other. Crucially, however, since communication between the client side part of an Internet application with its server side part, usually carried out by way of the XMLHTTPRequest object, uses the same HTTP request-response mechanism, application layer specifications encompass the behaviour of rich Internet applications, too.

Our contributions are therefore threefold. Firstly, we formalise HTTP transactions and Internet application state, and introduce the concept of transition rules which describe the relation between the two. This constitutes our model of the Internet's application layer. Secondly, we define variants of the logic UCTL* [19], whose semantics are defined over the variants of Kripke transition systems [14] that these transition rules generate. We use these logics to express properties of these transition systems. Thirdly, in the case study we present a formal treatment of a distributed, rich Internet application and prove properties about the information it stores and disseminates. Our approach is a general one in the sense that we do not rely on a particular framework or programming paradigm. The properties we prove are ones that could be exhibited by any HTTP-based Internet program or protocol.

2 HTTP transactions and Internet application state

To the uninitiated an Internet application may appear to persist. A web page loads and then remains in the browser in much that same way that a desktop application remains in front of the user. The real situation is somewhat different, however. The presence of the web page may give the impression of permanence but in reality the instance of the server part of the application that produced it is likely to be destroyed the moment the web page is produced. By contrast, the client side part of a rich Internet application does persist, it can communicate with the server side part alongside the browser and it has access to and can amend shared information stored by the browser. To complicate the picture still further, the freedom that the browser admits means that user interactions cannot be guaranteed to occur sequentially in relation to the unfolding of the application state. The user may open new tabs, refresh the page, type URLs directly into the address bar or, most notoriously, use the forward and back buttons. The effects of these actions are complex to discern to say the least.

Rather than attempting to model this cat's cradle, we take a simplified approach. Our analysis begins with the working of the HTTP protocol and in particular the request-response pair, which we call an HTTP transaction. Although it is possible for the request-response pair to be broken, the connection may fail or the server may fault, for example, in our analysis it will be considered inviolable. Many transactions may happen, building up the complete picture, so we therefore take account of all possible transactions. Although if we make the assumption that no two transactions can occur at exactly the same time then they can be given a temporal ordering, such an ordering is of little practical use since it is extremely difficult if not impossible to establish any causal link between successive transactions. It is best therefore to think of the transactions to begin with as floating around in a kind of "soup", with no connections between them at all.

At this point we make two observations. The first is that HTTP transactions can happen as a result of requests from both the browser and the client side part of the application, and therefore this soup may contain both types. The second is that no account is taken of the cause of each transaction. Each may happen as a result of the user simply clicking a link; pressing the back button and then confirming that they would like the browser to resend form information; manually typing an address into the address bar; the client side part of the application making

a call via the XMLHTTPRequest object. By eschewing these causes we avoid the trap of failing to properly account for them. Furthermore, we do not have a model of browser architecture or attempt to discern the subtle interplay of user interactions with iframes, different tabs and the like. And if we lose something in taking this line what we gain is a complete model of what remains, namely the behaviour of the application as evinced by its transactions over HTTP.

This soup of transactions on its own is of little use. To begin with it takes no account of the state of the application, however this is defined. And some causal links must be established, some additional structure needs to be added. To do this we include the states of the application in our models, and associate them with transactions. More precisely, we say that if an application is in a certain state before an HTTP request, its state may change after the HTTP response. We call this process a transition, with that part of the transition visible over HTTP being precisely the transaction.

Our models therefore consist of the set of all possible application states together with sets of all possible transactions between any two states. We do not consider the causes of requests, and we assume that any request can happen at any time. This approach has precedents in the literature on the verification of network protocols, for example [16]. In such cases there is no user interface to be modelled, and verification consists of exhaustively "covering all the bases" with regard to all possible transactions and their effects on application state. And this approach also characterises some of the most stringent web security requirements in industry, including [2], where no assumptions are made about the types of requests that can be made in an "anything can happen" analysis. Our approach is a natural extension of these approaches.

3 A formal model and examples

In this section we define a formal model of HTTP transactions and Internet application state as well a transition rules, which describe the relation between the two. We then define two example applications based on this model. The first application uses cookies to enforce user flow through a site and demonstrates the persistence of application state in the browser. The second application maintains a server state in order to track visitors to a site.

The HTTP protocol is a request-response protocol that admits communication between a client, typically a web browser or a JavaScript program running in a browser, and a server, typically an instance of a script or program running on a web server. Both requests and responses consist of a number of headers together with a body. The request often contains a reference, called a uniform resource locator or URL for short, to a resource to be retrieved by the server. Scripts and programs on the server can also be invoked in this manner. The headers communicate cookies in both directions alongside other types information, such as the client's IP address. Cookies consist of name-value pairs of textual information and allow stateful communication between clients and servers. They are initially passed by the server to the client in response to a request. They are then stored in the browser and passed back from the client to the server in subsequent requests. Usually the instance of a script or program that creates any response is destroyed the moment the reponse is produced and therefore cookies allow subsequent instances to recover information specific to a particular client. Typically an identifier unique to the client is stored in a cookie, which allows information specific to that client to be retrieved from a database on the server. This leads to the common misconception that the identifier itself persists there. Aside from this, state not related to any particular client may be held on the server. Also a JavaScript program running in the browser may hold state apart from the browser's cookie store.

Cookies, which we model as atomic entities rather than name-value pairs, are taken from a set of cookies $\mathscr{C}$, with c, c' ranging over $\mathcal{C} = \mathscr{P}(\mathscr{C})$. Since the browser can effectively be instructed to both add and remove cookies from its store, we model them as being signed when passed to the browser, so we have a set of signed cookies $\mathscr{C}^{\pm} = \{+x | x \in \mathscr{C}\} \cup \{-x | x \in \mathscr{C}\}$, with $c^{\pm}$ ranging over $\mathcal{C}^{\pm} = \mathscr{P}(\mathscr{C}^{\pm})$. Global states are tuples containing browser's cookie store, which from now on we refer to as the browser's state, together with the client side and side server states:

$$(c, j, s) \in \mathcal{C} \times \mathcal{J} \times \mathcal{S}$$

Requests consist of a set of cookies $c \in \mathcal{C}$ together with a URL u taken from a set of URLs $\mathcal{U}$. Responses consist of a set of signed cookies $c^{\pm} \in \mathcal{C}^{\pm}$ together with a body b taken from a set of bodies B. We may refine requests and responses further as required, adding additional request headers, for example, or parameterising the response bodies. A transaction consists of a request and response in one tuple, which labels a transition between global states:

$$(c, j, s) \xrightarrow{(c, u, c^{\pm}, b)} (c', j', s')$$

Note that the browser's state c is echoed directly in the request, and therefore there is some redundancy in this formalism. We keep this, however, so that the transaction represents that part of the transition visible over HTTP in its entirety. To continue, when the browser receives signed cookies, it amends its state according to the following generic rule:

$$c' = c \cup \{x \in \mathscr{C} \mid +x \in c^{\pm}\} \setminus \{x \in \mathscr{C} \mid -x \in c^{\pm}\}$$

Transitions are governed by transition rules, which, given an initial state and request, determine both the response and final state:

$$(c, j, s) \xrightarrow[c^{\pm}, b]{c, u} (c', j', s')$$

Transition rules are typically obtained from static analysis. This process could be automated or might be an informal but nonetheless informed approximation, as in case study given later. In the case of the examples that follow, however, the transition rules are formally stated and effectively define the applications in question.

3.1 The 'agreement' example

When visiting a website, a user must first agree to the use of cookies, then agree to the terms and conditions, and only then are they given access to the welcome page. No assumptions can be made about the order in which the pages are requested, however. In order to enforce the correct flow, two nonce cookies are set when the user agrees to the relevant missives. An error page is shown if the user requests the pages out of order, say, or bookmarks a page in order to come back to it at a later stage when the cookies have been deleted. The user agrees both to the use of cookies and to the terms and conditions by way of forms presented on the first two pages. In both cases, only if the user ticks the checkbox and then submits the form are they allowed to continue. Upon successful submission of the forms nonce cookies are set, and it is the presence or otherwise of these cookies which determines which pages can be shown. No distinction is made between a page requested via the address bar and form submissions with the checkboxes left unticked, in which case the input is treated as invalid and the relevant page is shown again.

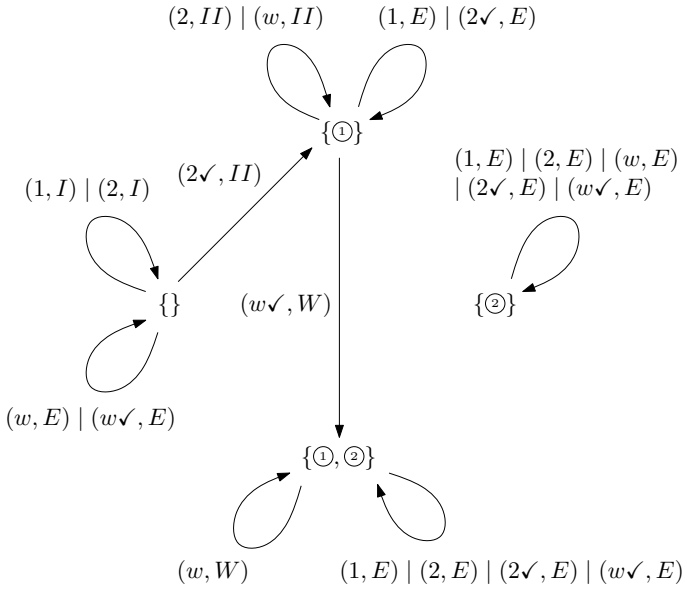

Figure 3.1: The 'agreement' application transition system

We model the two nonce cookies, plus the URLs that are used to request the three pages. To model valid form submissions, two additional URLs with ticks are added. We model the four page bodies returned, the first two containing the forms, then the welcome and error pages:

$$\mathcal{C} = \mathscr{P}(\{①,②\}) \quad \mathcal{U} = \{1, 2, w, 2✓, w✓\} \quad \mathcal{B} = \{I, II, W, E\}$$

The application's behaviour is defined by the following set of transition rules:

$$\{\} \xrightarrow[\{\}\ I]{\{\}\ 1 \mid 2} \{\} \qquad \{①\} \xrightarrow[\{\}\ II]{\{①\}\ 2 \mid w} \{①\} \qquad \{①,②\} \xrightarrow[\{\}\ W]{\{①,②\}\ w} \{①,②\}$$

$$\{\} \xrightarrow[\{+①\}\ II]{\{\}\ 2✓} \{①\} \qquad \{①\} \xrightarrow[\{+②\}\ W]{\{①\}\ w✓} \{①,②\}$$

$$\{\ldots\} \xrightarrow[\{\}\ E]{\{①\} \mid \{②\} \mid \{①,②\}\ 1} \{\ldots\} \qquad \{\ldots\} \xrightarrow[\{\}\ E]{\{②\} \mid \{①,②\}\ 2} \{\ldots\} \qquad \{\ldots\} \xrightarrow[\{\}\ E]{\{\} \mid \{②\}\ w} \{\ldots\}$$

$$\{\ldots\} \xrightarrow[\{\}\ E]{\{①\} \mid \{②\} \mid \{①,②\}\ 2✓} \{\ldots\} \qquad \{\ldots\} \xrightarrow[\{\}\ E]{\{\} \mid \{②\} \mid \{①,②\}\ w✓} \{\ldots\}$$

The first row represents the occasions when the user requests the correct page or submits a form in the correct order but without ticking the checkbox. The second row represents the occasions when the user correctly submits a form and a nonce cookie is set. The third and fourth rows represent the remaining occasions, on all of which error pages are shown. Note that both the client side and server side states are missing from these rules, since they are not present in this example. Note also that for brevity's sake, the browser states are sometimes replaced by ellipses with the understanding that they can be inferred from the headers. Recall that the browser's

state is echoed directly in the request. Finally, note that vertical bars allow several rules to be combined into one. Figure 3.1 shows the resulting transition system, which in this example is no more than a diagrammatic representation of the transition rules. The cookie headers are omitted, since they can be inferred from the initial and final states. Note that we also use the vertical bar rather than set notation, in keeping with the transition rules.

3.2 The 'visitors' example

A website maintains a counter on a landing page which is incremented every time a user requests that page for the first time. No cookies can be used and so in order to determine whether a request has originated from a new user, the request IP address is checked against the server state. If the request IP address cannot be found, the counter is incremented and the new IP address is added to the server state.

We model the lack of cookies, a single URL, a set of IP addresses and a countable set of page bodies. The server state is modelled as a subset of the set of the IP addresses paired with the states of the counter, which are precisely the natural numbers $\mathbb{N}$. In this case the application also has an initial state:

$$\mathcal{C} = \mathscr{P}(\{\}) \quad \mathcal{U} = \{U\} \quad \mathcal{A} = \{A_1, A_2, \cdots, A_N\} \quad \mathcal{B} = \{B(n) \mid n \in \mathbb{N}\}$$

$$(a, n) \in \mathcal{S} \subseteq \mathscr{P}(\mathcal{A}) \times \mathbb{N} \quad (\{\}, 0) \in \mathcal{S}$$

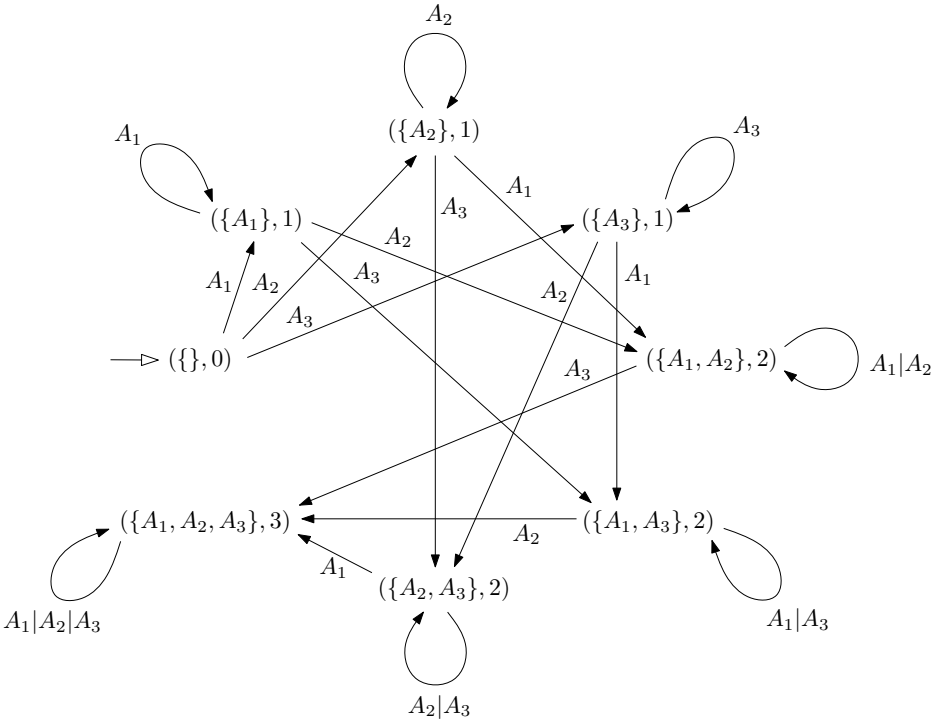

Figure 3.2: The 'visitors' application transition system

The application's behaviour is defined by the following two transition rules for $1 \leqslant i \leqslant N$:

$$(a,n) \xrightarrow[B(n)]{U,\,A_i} (a,n) \quad A_i \in a \qquad\qquad (a,n) \xrightarrow[B(n+1)]{U,\,A_i} (a \cup \{A_i\}, n+1) \quad A_i \notin a$$

Whereas the IP address is communicated alongside the URL in the request, on the other hand the counter is communicated as part of the page body. The returned page is essentially parameterised by the counter, therefore, reflecting the fact that the counter is embedded in the body of the page by some means. Note that the browser and client side states are missing from these rules, since neither is not present in this example. Note also that since there is no browser state, no cookies are communicated in the request and response parts of the rules. Strictly speaking these two transition rules are not single rules but rule schemas defining two countable sets of rules for each $(a,n) \in \mathscr{P}(\mathcal{A}) \times \mathbb{N}$. Figure 3.2 shows the resulting transition system in the case when $N = 3$. The initial state is highlighted on the left with a white filled arrow. Since there is only request URL and one returned page, these are omitted. It is also assumed that the correct value of the counter is echoed in the returned page and therefore this information is not included. Note again that we use the vertical bar rather than set notation for the transition labels.

4 A treatment of the examples using temporal logics

The two transition systems defined in the previous section are, with small differences, instances of Kripke transition systems, with information on both the states and transitions. In order to express properties of the applications in question we therefore use a variant of UCTL* [19], a state/event based logic whose semantics are defined over these types of transition systems. In what follows we give the standard definition of Kripke transition systems [14], together with the syntax and semantics of UCTL*, and then define variants of both for our purposes.

Definition 4.1. *A Kripke Transition System is a tuple* $(S, Act, \longrightarrow, AP, \mathcal{L})$ *where:*

- S *is a set of states ranged over by* s, s_0 *and* s_1*,*
- Act *is a set of actions, with* 2^{Act} *ranged over by* α*,*
- $\longrightarrow \subseteq S \times 2^{Act} \times S$ *is the transition relation with* $(s_0, \alpha, s_1) \in \longrightarrow$*,*
- AP *is a set of atomic propositions ranged over by* p*,*
- $\mathcal{L} : S \times AP \longrightarrow \{true, false\}$ *is an interpretation function that associates a value of* $true$ *or* $false$ *with each* $p \in AP$ *for each* $s \in S$*,*
- *For any two transitions,* $(s_0, \alpha_0, s_1), (s_0, \alpha_1, s_1) \in \longrightarrow \Rightarrow \alpha_0 = \alpha_1$*.*

Note that since transitions carry sets of actions and not just one, there is no silent action, with the empty set $\{\}$ being considered silent instead. Note also that we limit the number of transitions between any two states in any one direction to at most one.

Paths are sequences of transitions where the final state of one transition equals the initial state of the next, if there is one. They are ranged over by σ, σ' and σ''. Maximal paths are either infinite or their last state has no outgoing transitions. For the set of maximal paths starting at state s we write $\mu path(s)$. For the initial and final states of the first transition of a path σ we write $\mathsf{s}\sigma$ and $\sigma\mathsf{s}$, respectively, and we write $\sigma\mathsf{T}$ for the set of actions of the first transition of a path σ. A suffix σ' of a path σ is a path such that $\sigma = \sigma''\sigma'$ for some possibly zero length path σ''. A proper suffix σ' of a path σ is a path such that $\sigma = \sigma''\sigma'$ for some non-zero length path σ''. We write $\sigma \leqslant \sigma'$ when σ' is a suffix of σ and $\sigma < \sigma'$ when σ' is a proper suffix of σ.

Definition 4.2. *The syntax UCTL* is:*

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi' \mid \exists\pi \quad \pi ::= \phi \mid \neg\pi \mid \pi \wedge \pi' \mid X\pi \mid X_a\pi \mid \pi\, U\pi'$$

Here ϕ is a state formula and π a path formula.

Definition 4.3. *The semantics of UCTL* is:*

- $s \vDash p$ *iff* $\mathcal{L}(s,p) = true$
- $s \vDash \neg\phi$ *iff* $s \nvDash \phi$
- $s \vDash \phi \wedge \phi'$ *iff* $s \vDash \phi$ *and* $s \vDash \phi'$
- $s \vDash \exists\pi$ *iff* $\exists\sigma \in \mu path(s) : \sigma \vDash \pi$
- $\sigma \vDash \phi$ *iff* $_S\sigma \vDash \phi$
- $\sigma \vDash \neg\pi$ *iff* $\sigma \nvDash \pi$
- $\sigma \vDash \pi \wedge \pi'$ *iff* $\sigma \vDash \pi$ *and* $\sigma' \vDash \pi'$
- $\sigma \vDash X\pi$ *iff* $\exists(s,\alpha,s')\sigma'' : \sigma = (s,\alpha,s')\sigma'', \sigma'' \vDash \pi$
- $\sigma \vDash X_a\pi$ *iff* $\exists(s,\alpha,s')\sigma'' : a \in \alpha, \sigma = (s,\alpha,s')\sigma'', \sigma'' \vDash \pi$
- $\sigma \vDash \pi\, U\pi'$ *iff* $\exists\sigma' \geqslant \sigma : \sigma' \vDash \pi', \forall\sigma'' : \sigma \leqslant \sigma'' < \sigma' : \sigma'' \vDash \pi$

We now define a variant of UCTL* for our purposes, replacing atomic propositions with propositions about typed variables.

Definition 4.4. *Let x be a typed variable. The type of x, written τ_x, is the set of its permitted values. A set of typed variables is written X.*

Definition 4.5. *An interpretation of a set of typed variables X is a function I from X to $\bigcup\{\tau_x | x \in X\}$ such that $I(x) \in \tau_x$ for each $x \in X$. The set of all interpretations for a given set of typed variables X is written $\mathbb{I}$.*

Definition 4.6. *An (amended) Kripke Transition System a tuple $(S, Act, \longrightarrow, X, \mathcal{L})$ where the set of atomic propositions AP and interpretation function $\mathcal{L}$ replaced with the following:*

- *X is a set of typed variables,*
- *$\mathcal{L} : S \longrightarrow \mathbb{I}$ is an interpretation function that maps every state to an interpretation.*

Action formulae remain the same, however state formulae must be amended in order to accomodate propositions of the form $x = x'$ where x and x' are of the same type, or $x = v$ where $v \in \tau_x$. In the spirit of [5], we also introduce a set K of typed global variables, extending the syntax to include propositions of the form $x \leqslant k$ where $k \in K$, x and k are of the same type, the type is a numerical type that supports inequalities and the value of k is in τ_x.

Definition 4.7. *The (amended) syntax of UCTL* state formulae ϕ is:*

$$\phi ::= true \mid x = v \mid x = x' \mid x \leqslant k \mid \neg\phi \mid \phi \wedge \phi' \mid \exists\pi$$

Definition 4.8. *The (amended) semantics of UCTL* state formulae ϕ is:*

- $s \vDash x = v$ *iff* $\mathcal{L}(s)(x) = v$
- $s \vDash x = x'$ *iff* $\mathcal{L}(s)(x) = \mathcal{L}(s)(x')$
- $s \vDash x \leqslant k$ *iff* $\mathcal{L}(s)(x) \leqslant k$

The relation $s \vDash p$ is dropped with the other relations remaining.

If x has a numerical type, we also permit expressions of the form $x = v + 1$ with the obvious semantics. Unlike [5], which introduces typed constants, we might close formulae containing typed global variables under existential or universal quantification. If they are not closed in such a way, we consider them constant. We also quantify over sets of constants.

Finally, we derive the "eventually" F and "always" G operators [8] in the usual way and add a T operator, the latter being a convenient shorthand for combining path formulae satisfied immediately and "nexttime" on a path:

$$F\pi' = true_{true}U_{true}\pi' \qquad G\pi = \neg F\neg\pi \qquad \pi T_\chi \pi' = \pi \wedge X_\chi \pi'$$

4.1 The 'agreement' example

Here $X = \{c\}$ and $\tau_c = \{\{\}, \{①\}, \{②\}, \{①,②\}\}$. For the sake of readability, formulae $x = v$ are shortened to just v. For example, $c = \{①\}$ is written $\{①\}$. The ideal user journey is the following path:

$$\{\} \xrightarrow{(1,I)} \{\} \xrightarrow{(2✓,II)} \{①\} \xrightarrow{(w✓,W)} \{①,②\}$$

It is easy to produce a path with the same requests but the wrong responses, however:

$$\{①,②\} \xrightarrow{(1,E)} \{①,②\} \xrightarrow{(2✓,E)} \{①,②\} \xrightarrow{(w✓,E)} \{①,②\}$$

Lemma 4.1. *The following two formulae can be satisfied:*

$$\exists(\{\}T_{(1,I)}\{\}T_{(2✓,II)}\{①\}T_{(w✓,W)}\{①,②\}) \qquad \exists(X_{(1,E)}X_{(2✓,E)}X_{(w✓,E)}true)$$

Proof. These follow immediately from inspecting figure 3.1, which includes the above paths. □

4.2 The 'visitors' example

Here $X = \{a, n\}$, $\tau_a = \mathscr{P}(\mathcal{A})$, $\tau_n = \mathbb{N}$ and $K = \{N_1, N_2, N\}$ where $N_1, N_2, N \in \mathbb{N}$ and $N_1, N_2 \leqslant N$. We continue to use an ordered pair (a, n) for the state rather than a set.

Lemma 4.2. *The only state with $n = 0$ and $a = \{\}$ is the initial state.*

Proof. Only the second rule can be applied to the initial state, and there is no rule for subsequently decreasing n or reducing a. □

Lemma 4.3. *Apart from the initial state, all reachable states are of the form $(a \cup \{A_i\}, n+1)$, where $A_i \notin a$, and are reached by way of a transition from state (a, n).*

Proof. The first transition rule leaves states unchanged, therefore if a state is not the initial state it must be reached by the application of the second transition rule to some state, say (a, n), and therefore must be of the form $(a \cup \{A_i\}, n+1)$. □

Lemma 4.4. *For any reachable state (a, n), $|a| = n$.*

Proof. By induction on n. For $n = 0$ we know from lemma 4.2 that the only state with $n = 0$ is the initial state. Assume now that for all states (a, k) we have $|a| = k$. Consider some other, reachable state with $n = k+1$ which by lemma 4.3 we can write as $(a \cup \{A_i\}, k+1)$, reachable from the state (a, k). By the induction hypothesis $|a| = k$ and therefore $|a \cup \{A_i\}| = k+1$. □

Lemma 4.5. *For any reachable state (a, n), $n \leqslant N$. Formally, $\forall G(n \leqslant N)$ is valid.*

Proof. We simply observe that $n = |a| \leqslant |\mathcal{A}| = N$. □

We omit the proof that there is a path that leads to a state (a, n) for any $a \in \mathscr{P}(\mathcal{A})$ but note that this fact, in tandem with lemma 4.4, defines the set $\mathcal{S}$ of all reachable states.

Lemma 4.6. *The value of the counter returned to the user is no greater than N.*

Proof. For the first rule, the response is $B(n)$ with $n \leqslant N$. For the second rule, the response is $B(n+1)$ but $n = |a|$ by lemma 4.4, $a \subset \mathcal{A}$ and since $|\mathcal{A}| = N$, $n < N$ and therefore $n+1 \leqslant N$. □

Lemma 4.7. *No user can increment the counter twice. Formally, the following is not satisfiable:*

$$\exists A_i, N_1, N_2 : \exists(((n = N_1)T_{A_i}(n = N_1 + 1)) \wedge F((n = N_2)T_{A_i}(n = N_2 + 1)))$$

Proof. Suppose that the formula is satisfied. The path in question must be of the following form:

$$(a_1, N_1) \xrightarrow{A_i} (a_1 \cup \{A_i\}, N_1 + 1) \;\cdots\; (a_2, N_2) \xrightarrow{A_i} (a_2 \cup \{A_i\}, N_2 + 1)$$

We first note that $a_1 \cup \{A_i\} \subseteq a_2$, since all transitions are governed by two transition rules, one of which leaves the set a of any state (a, n) alone and the other of which adds an element. Then we note that in order for the second rule to be applied at the end of the path we must have $A_i \notin a_2$. However, $a_2 \supseteq a_1 \cup \{A_i\}$ and $A_i \in a_1 \cup \{A_i\}$ implies $A_i \in a_2$, a contradiction. □

5 The QuICDoc case study

QuICDoc [3] is a small, distributed, rich Internet application that uses the Concur algorithm [18] to merge concurrent changes to a shared document. A user simply types into a textarea and a client-side part of the application communicates the changes, what are called "diffs", to the server side part, which disseminates them to all the other client side parts after amending them in such as way as to ensure that they can be merged consistently. In [18] it is proved that this is possible but no account is taken of the effects of implementing the algorithm in an Internet setting. We bridge this gap in this section.

We consider just two clients, defining the global state as a tuple $(j_1, j_2, s) \in \mathcal{J}_1 \times \mathcal{J}_2 \times \mathcal{S}$ encompassing both client side states alongside the server side state. In order to relate the formal model to the implementation we define states as named tuples with values associated with the relevant heap variables, thus $j_i(\mathsf{gUid}, \mathsf{gWorkingDoc}, \mathsf{gTempDoc})$ and $s(\mathsf{gLastUid}, \mathsf{gDocument}, \mathsf{gDiffss})$ for $i = 1, 2$. The transition rules are broken down in to client side and server side rules, and are obtained from the source code [3] by manual static analysis:

$$j_i(\epsilon, \epsilon, \epsilon) \xrightarrow[uid, doc]{\mathsf{GET\_DOC}} j_i(uid, doc, \epsilon)$$

$$j_i(uid, doc, temp) \xrightarrow[diffs]{\mathsf{GET\_DIFFS}, uid} j_i(uid, \mathsf{applyDiffs}(doc, diffs), temp)$$

$$j_i(uid, doc, temp) \xrightarrow{\mathsf{PUT\_DIFFS}, uid, \mathsf{makeDiffs}(doc, temp)} j_i(uid, temp, temp)$$

$$s(luid, doc, diffss) \xrightarrow[luid+1, doc]{\mathsf{GET\_DOC}} s(luid + 1, doc, \mathsf{resetDiffs}(diffss, uid))$$

$$s(luid, doc, diffss) \xrightarrow[\mathsf{getDiffs}(diffss, uid)]{\mathsf{GET\_DIFFS}, uid} s(luid, doc, \mathsf{resetDiffs}(diffss, uid))$$

$$s(luid, doc, diffss) \xrightarrow{\mathsf{PUT\_DIFFS}, uid, diffs} s(luid, \mathsf{amendDoc}(doc, uid, diffs), \mathsf{amendDiffss}(diffss, uid, diffs))$$

Here ϵ represents an undefined value and a side condition for all the rules is that $uid \neq \epsilon$. We note in passing that breaking the transition rules in this way provides a means of partially specifying the behaviour of each part of the application, and hence taken together can be considered as a

partial specification of the application as a whole. To continue, the application has an initial global state, which is again broken down into client side and server side parts, thus $j_i(\epsilon,\epsilon,\epsilon)$ and $s(0,[[],[]],"")$ for $i=1,2$. Here $[...]$ represents an array.

To make use of these rules we recombine them. To do this we note that the client side part produces a request and consumes a response whilst the server side part consumes a request and produces a response. We therefore replace the elements of the request part of the server side rules with matching elements of the client side rules and vice versa. We implement this by collecting the elements of the requests and responses into tuples, with elements being either variables, constants or terms. When collecting elements we add the relevant subscripts, for example the uid variable in client side rules becomes uid_i, and so on. We apply a partial matching function recursively in a similar vein to [9], with a substitution being returned if a match can be found.

Definition 5.1. *The partial matching function for pairing transition rules is defined by structural induction by means of the following partial functions:*

$$\begin{array}{c}\mathsf{match}((e_1,e_2...),(e_1',e_2'...)) = \{\mathsf{match}(e_1,e_1')\}\cup\mathsf{match}((e_2...),(e_2'...))\\ \begin{array}{cc}\mathsf{match}(v,v') = v/v & \\ \mathsf{match}(v,c') = v/c' & \mathsf{match}(c,c) = \emptyset \\ \mathsf{match}(v,t') = v/t' & \mathsf{match}((),()) = \{\}\end{array}\end{array}$$

Here e_1,e_1', etc are arbitary elements, v,v' variables, c,c' constants and t' terms.

To match the requests, we make the server side tuple the first argument of the matching function since it is this tuple that contains the variables that must be replaced in the substitutions that follow. To match the responses, we make the client side tuple the first argument. By way of an example we show the matching of the requests and responses of the GET_DIFFS rules:

$$\begin{array}{l}\mathsf{match}((\mathsf{GET\_DIFFS},uid_s),(\mathsf{GET\_DIFFS},uid_i))\\ = \{\emptyset\}\cup\mathsf{match}((uid_s),(uid_i))\\ = \{\emptyset\}\cup\{\mathsf{match}(uid_s,uid_i)\}\\ = \{\emptyset\}\cup\{uid_s/uid_i\}\cup\mathsf{match}((),())\\ = \{\emptyset,uid_s/uid_i\}\end{array}\qquad\begin{array}{l}\mathsf{match}((diffs_i),(\mathsf{getDiffs}(diffss_s,uid_s)_s))\\ = \{\mathsf{match}(diffs_i,\mathsf{getDiffs}(diffss_s,uid_s)_s)\}\cup\mathsf{match}((),())\\ = \{diffs_i/\mathsf{getDiffs}(diffss_s,uid_s)_s\}\end{array}$$

Here the rules are paired as expected by matching the constants. If the requests and responses of two rules match, we apply the resulting substitutions to obtain a global rule. Here we show just the part of the GET_DIFFS rule corresponding to the transaction, omitting the global states:

$$\frac{\{diffs_i/\mathsf{getDiffs}(diffss_s,uid_s)_s\}.\{\emptyset,uid_s/uid_i\}.(\mathsf{GET\_DIFFS},uid_s)}{\{\emptyset,uid_s/uid_i\}.\{diffs_i/\mathsf{getDiffs}(diffss_s,uid_s)_s\}.(diffs_i)} = \frac{\mathsf{GET\_DIFFS},uid_i}{\mathsf{getDiffs}(diffss_s,uid_i)_s}$$

Note that the terms may contain elements that were not matched but are nevertheless subject to substitutions at this stage. The process is not a recursive one, however, since each set of substitutions only needs to be applied once.

5.1 A temporal logic with parameterised action formulae

Consider the proof of lemma 6.1 in [18]. To begin with we note that the assumption that the client cannot put diffs on the server before getting the document is intrinsic to the client side rules, since $uid \neq \epsilon$ and the third rule cannot therefore be employed before the first. Aside from this assumption, the proof relies on two facts. The first is that the document is communicated from server to client intact, which we express as a temporal formula.

Lemma 5.1. $X_{((\mathsf{GET\_DOC}),(uid,doc))}(doc_i = doc_s)$ *is satisfiable.*

Here the action formula becomes a pair of tuples similar to those used when combining transition rules. We borrow the subscript notation from the global rules, too. In fact in order to satisfy this formula we must match it against these global transition rules, since we do not have the transition system to hand. The global rule in question is the following:

$$(j_i(\epsilon,\epsilon,\epsilon), s(luid, doc, diffss)) \xrightarrow[(luid+1)_s, doc_s]{\mathsf{GET\_DOC}} (j_i((luid+1)_s, doc_s, \epsilon), s(luid+1, doc, diffss))$$

We again collect the elements of the requests and responses in the transition rule into tuples, applying a matching function recursively to both in order to match the formula against the rule.

Definition 5.2. *The partial matching function for matching formulae against transition rules is defined by structural induction by means of the following partial functions:*

$$\mathsf{match}((e_1, e_2...),(e_1', e_2'...)) = \{\mathsf{match}(e_1, e_1')\} \cup \mathsf{match}((e_2...),(e_2'...))$$

$$\begin{array}{rr} \mathsf{match}(v, v') = \emptyset & \mathsf{match}(\underline{v}, c') = v/c' \\ \mathsf{match}(v, c') = \emptyset & \mathsf{match}(\underline{v}, t') = v/t' \\ \mathsf{match}(v, t') = \emptyset & \mathsf{match}((),()) = \{\} \\ \mathsf{match}(c, c) = \emptyset & \end{array}$$

Here e_1, e_1', etc are arbitary elements, v, v' variables, c, c' constants, t' terms and $\underline{v}$ binding variables.

The aforementioned temporal formula and transition rule do indeed match, returning the substitutions $\{\emptyset\}$ and $\{\emptyset,\emptyset\}$, and since $j_i((luid+1)_s, doc_s, \epsilon) \vDash (doc_i = doc_s)$, the formula is satisfied.

The second fact is that when diffs are put on the server, they are suitably amended and then applied to the server's document. Transition rules allow the values of variables in the final state to be calculated from those in the initial state. In order to capture this mechanism in path formulae we must therefore save the value of variables in the initial state for use in the final path formula somehow. We do this by way of a universally quantified global variables, the values of which we fix in the initial proposition in the temporal formula.

Lemma 5.2. $\forall d : (doc_s = d) T_{((\mathsf{PUT\_DIFFS}, \underline{uid}, \underline{diffs}),())}(doc_s = \mathsf{amendDoc}(d, uid, diffs))$ *is satisfiable.*

In order to satisfy this formula we match it against the following global rule, which shows the relevant parts of the server side state only:

$$s(doc) \xrightarrow{\mathsf{PUT\_DIFFS}, uid_i, \mathsf{makeDiffs}(doc_i, temp_i)_i} s(\mathsf{amendDoc}(doc, uid_i, \mathsf{makeDiffs}(doc_i, temp_i)_i))$$

The match gives the substitution $\{uid/uid_i, diffs/\mathsf{makeDiffs}(doc_i, temp_i)_i\}$, which results in the final proposition:

$$(doc_s = \mathsf{amendDoc}(d, uid_i, \mathsf{makeDiffs}(doc_i, temp_i)_i))$$

A final state $s(\mathsf{amendDoc}(d, uid_i, \mathsf{makeDiffs}(doc_i, temp_i)_i))$ results from replacing the server side variable *doc* with the global variable d, and the formula is satisfied.
In order to make use of substitutions we have effectively replaced the $X_\chi \phi$ operator with an $X_\zeta \phi$ operator, where χ is a parameterised action formula.

Definition 5.3. *The semantics of the $X_\zeta \phi$ operator is:*

- $\sigma \vDash X_\zeta \phi$ *iff* $\sigma = \sigma'\sigma''$ *with* $\sigma'_{\mathsf{T}} \vDash \zeta \blacktriangleright \rho$ *and* $\rho.(\sigma'') \vDash \phi$

We now formally define the syntax and semantics of parameterised action formulae.

Definition 5.4. *The syntax of parameterised action formulae is:*

$$\begin{array}{rl} \zeta ::= & \underline{a} \mid \chi \\ \underline{a} ::= & (request, response) \\ request, response ::= & (\underline{e}...) \\ \underline{e} ::= & \underline{v} \mid v \mid c \mid t \\ t ::= & name(e...) \\ e ::= & v \mid c \end{array} \qquad \begin{array}{rl} \chi ::= & \tau \mid a \mid \neg\chi \mid \chi \wedge \chi' \\ a ::= & (request, response) \\ request, response ::= & (e...) \\ e ::= & v \mid c \end{array}$$

Definition 5.5. *The semantics of parameterised action formulae is:*

- $\alpha \vDash \underline{a} \blacktriangleright \rho$ *iff* $\exists a \in \alpha : \mathsf{match}(\underline{a}, a) = \rho$
- $\alpha \vDash \chi \blacktriangleright \emptyset$ *iff* $\alpha \vDash \chi$

The satisfaction relations for $\alpha \vDash \chi$ *otherwise remain the same.*

We conclude this section with two further temporal formulae, the first expresses the fact that when diffs are put on the server, all pending diffs are suitably amended, the second expresses the fact that when pending diffs are gotten by a client, they are applied to the client's document. The proofs are omitted to save space, but follow along the lines of the ones already given.

Lemma 5.3. *The following formulae are satisfiable:*

$$\forall d : (diffss_s = d) T_{((\mathsf{PUT\_DIFFS}, \underline{uid}, \underline{diffs}), ())} (diffss_s = \mathsf{amendDiffs}(d, uid, diffs)_s)$$

$$X_{((\mathsf{GET\_DIFFS}, uid_i), (\underline{diffs}))} (\mathsf{applyDiffs}(doc_i, diffs)_i = doc_s)$$

To satisfy the second of these formula we note that the previous results justify the equivalences $s * \delta * \delta'' = doc_i$, $\delta''.\Delta' = \mathsf{getDiffs}(uid_i)_s$ and $s' * \Delta'.\delta'' = doc_s$. Thus the path formula becomes $s' * \Delta'.\delta'' = s * \delta * \delta'' * \delta''.\Delta'$, which holds by lemma 6.2 in [18].

6 Related work

The approach taken in [4] formalises significant parts of the HTTP protocol as well as key web concepts such as the browser and the server. A concept of non-linear time is adopted in order to deal with the problem of the browser's back button, with the authors not being "...concerned with the actual temporal order between unrelated actions", an approach strongly espoused here.

In [12], significant emphasis is placed on user interactions. The opinion that "...any verfication tool for the web that does not account for user operations is not only incomplete, but potentially misleading" could be construed as being somewhat the opposite to our own. In [7], emphasis is placed on the structure of the web site in terms of its interelated links, with transitions being seen as taking place between web pages. In effect a temporal order is imposed on actions, with users chosing only from a set of inputs presribed by each web page, in marked contrast to the approach taken in [4] and our own. The variants of CTL and CTL* developed therein, employing free and bounds variables as well universal closure, bear some resemblance to the variants defined here, however.

By eschewing a model of user interactions, the approach here bears more resemblance to approaches used in the analysis of web services rather than programs [16]. Although our approach is not a model checking one, for example, our model results in the same number of actions from each state, an exhaustive treatment that typifies model checking. Our approach also owes much

to that of [9]. The variants of the logic UCTL* defined here borrow much from the logic SoCL, which is defined over similar transition systems to the ones defined here, there called doubly labelled transition systems [6]. In particular the concepts of binding variables and the matching of action formulae are taken directly and adopted here.

7 Conclusions and future work

The model of HTTP transactions and Internet application state we have defined has allowed us to prove properties of a rich, distributed Internet application. Because this model is founded only on the HTTP protocol and Internet application states, and not on any particular framework or programming paradigm, it could be used to prove properties of any Internet program or protocol, provided we can obtain the transition rules.

We coined the term application layer specification in the introduction but keep its precise meaning deliberately vague. It could refer to the transition systems we generate, which are useful visual aids and provide the semantics for the logics we employ. It could refer to the temporal formulae, which capture information about application states and transactions. Or it could refer to the transistion rules which, broken down into rules for each part of the application, provide a useful starting point for design and specification. All of these concepts provide insight into the behaviour of Internet applications.

The case study chosen was a relatively simple rich Internet application, with both server side and client side parts written in JavaScript. Our long term goal is more ambitious, however. In future work we plan to use application layer specifications to formalise properties of larger HTTP-based programs and protocols with, specifically, the Open ID Connect protocol in mind [1]. In future we plan to develop a more realistic model of HTTP transactions, too. In particular we need to break the rule that they are inviolable and label transitions with either requests or responses but not necessarily both at once.

The choice of the logic UCTL* we motivated by the need for formulae with nested next time operators but, rather than defining the $\phi T_\chi \phi'$ operator as shorthand for $\phi \wedge X_\chi \phi'$, we could define its semantics directly and otherwise do away with the need for a fully branching logic.

Finally, we make some comments on the use of terms in the transition rules in the case study, and the fact that temporal formulae are satisfied against these rules rather than the transition system that they generate. Both approaches are somewhat informal, however a formal treatment of the former can be found in [17], which includes transition systems generated by rules, which are gratifying called transition system specifcations. Moreover, semantics that encompases both the denotational meaning of terms and their operational meaning over transition systems are developed therein, indeed these two semantics are shown to be equivalent by means of the concept of mixed terms. For example, the specification $\mathcal{R}_P$ of a system is formally defined as:

$$\mathcal{R}_P = \mathcal{R} \cup \{p \xrightarrow{a} q \mid \langle a, q \rangle \in \psi(p)\}$$

Here R are the rules and $\psi : P \longrightarrow \mathscr{P}_f(A \times P)$ is a transistion system defined as a coalgebra, hence both the rules and all possible transitions are considered as axioms. We conclude therefore by looking forward to a full formal treatment based on this approach.

7.1 Acknowledgements

The author thanks the anonymous reviewers for their many helpful comments and corrections.

References

[1] *OpenID Connect*. Available at http://openid.net/connect/.

[2] *PCI SSC Data Security Standards Overview*. Available at https://www.pcisecuritystandards.org/security_standards/.

[3] *QuICDoc*. Available at http://www.doc.ic.ac.uk/~jes204/quicdoc.zip.

[4] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John C. Mitchell & Dawn Song (2010): *Towards a Formal Foundation of Web Security*. In: *CSF*, pp. 290–304. Available at http://doi.ieeecomputersociety.org/10.1109/CSF.2010.27.

[5] Jürgen Bohn, Werner Damm, Orna Grumberg, Hardi Hungar & Karen Laster (1998): *First-Order-CTL Model Checking*. In: *FSTTCS*, pp. 283–294. Available at http://dx.doi.org/10.1007/b71635.

[6] Rocco De Nicola & Frits W. Vaandrager (1990): *Three Logics for Branching Bisimulation (Extended Abstract)*. In: *LICS*, pp. 118–129.

[7] Alin Deutsch, Liying Sui & Victor Vianu (2007): *Specification and verification of data-driven Web applications*. *J. Comput. Syst. Sci.* 73(3), pp. 442–474. Available at http://dx.doi.org/10.1016/j.jcss.2006.10.006.

[8] E. Allen Emerson & Joseph Y. Halpern (1983): *"Sometimes" and "not never" revisited: on branching versus linear time (preliminary report)*. In: *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '83, ACM, New York, NY, USA, pp. 127–140, doi:10.1145/567067.567081. Available at http://doi.acm.org/10.1145/567067.567081.

[9] Alessandro Fantechi, Stefania Gnesi, Alessandro Lapadula, Franco Mazzanti, Rosario Pugliese & Francesco Tiezzi (2012): *A logical verification methodology for service-oriented computing*. *ACM Trans. Softw. Eng. Methodol.* 21(3), p. 16. Available at http://doi.acm.org/10.1145/2211616.2211619.

[10] Philippa Gardner, Sergio Maffeis & Gareth David Smith (2012): *Towards a program logic for JavaScript*. In: *POPL*, pp. 31–44. Available at http://doi.acm.org/10.1145/2103656.2103663.

[11] Zef Hemel, Danny M. Groenewegen, Lennart C. L. Kats & Eelco Visser (2011): *Static consistency checking of web applications with WebDSL*. *J. Symb. Comput.* 46(2), pp. 150–182. Available at http://dx.doi.org/10.1016/j.jsc.2010.08.006.

[12] Daniel R. Licata & Shriram Krishnamurthi (2004): *Verifying Interactive Web Programs*. In: *ASE*, pp. 164–173. Available at http://doi.ieeecomputersociety.org/10.1109/ASE.2004.10054.

[13] Michael C. Martin, V. Benjamin Livshits & Monica S. Lam (2005): *Finding application errors and security flaws using PQL: a program query language*. In: *OOPSLA*, pp. 365–383. Available at http://doi.acm.org/10.1145/1094811.1094840.

[14] Markus Müller-Olm, David A. Schmidt & Bernhard Steffen (1999): *Model-Checking: A Tutorial Introduction*. In: *SAS*, pp. 330–354. Available at http://dx.doi.org/10.1007/3-540-48294-6_22.

[15] Lawrence C. Paulson (1999): *Inductive Analysis of the Internet Protocol TLS*. *ACM Trans. Inf. Syst. Secur.* 2(3), pp. 332–351. Available at http://doi.acm.org/10.1145/322510.322530.

[16] Anders P. Ravn, Jirí Srba & Saleem Vighio (2010): *A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL*. In: *ISoLA (1)*, pp. 579–593. Available at http://dx.doi.org/10.1007/978-3-642-16558-0_47.

[17] Jan Rutten & Daniele Turi (1994): *Initial Algebra and Final Coalgebra Semantics for Concurrency*.

[18] James Smith (2013): *Concur - An Algorithm for Merging Concurrent Changes without Conflicts*. Available at http://arxiv.org/abs/1303.7462.

[19] James Smith (2013): *State-event based versus purely Action or State based Logics*. Available at http://arxiv.org/abs/1303.7459.

Template extraction based on menu information

Julian Alarte David Insa Josep Silva Salvador Tamarit
Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación
Camino de Vera s/n, E-46022, Valencia, Spain.
jalarte@dsic.upv.es dinsa@dsic.upv.es jsilva@dsic.upv.es stamarit@dsic.upv.es

Web templates are one of the main development resources for website engineers. Templates allow them to increase productivity by plugin content into already formatted and prepared pagelets. For the final user templates are also useful, because they provide uniformity and a common look and feel for all webpages. However, from the point of view of crawlers and indexers, templates are an important problem, because templates usually contain irrelevant information such as advertisements and banners. Processing and storing this information is likely to lead to a waste of resources (storage space, bandwidth, etc.). It has been measured that templates represent between 40% and 50% of data on the Web. Therefore, identifying templates is essential for indexing tasks. In this work we propose a novel method for automatic template extraction that is based on similarity analysis between the DOM trees of a collection of webpages that are detected using menus information. Our implementation and experiments demonstrate the usefulness of the technique.

1 Introduction

A web template is a prepared HTML page where formatting is already implemented and visual components are ready to insert content. Web templates are used as a basis for composing new webpages that share a common look and feel. This is good for web development because many tasks can be automated thanks to the reuse of components. In fact, many websites are maintained automatically by code generators, which generate webpages using templates. Web templates are also good for users, which can benefit from intuitive and uniform designs with a common vocabulary of colored and formatted visual elements.

Contrarily, web templates are an important problem for crawlers and indexers, because they judge the relevance of a webpage according to the frequency and distribution of terms and hyperlinks. Since templates contain a considerable number of common terms and hyperlinks that are replicated in a large number of webpages, relevance may turn out to be inaccurate, leading to incorrect results [1, 15, 17]. Moreover, in general, templates do not contain relevant content, they usually contain one or more pagelets [5, 1] (i.e., self-contained logical regions with a well defined topic or functionality) where the main content must be inserted. The main content of a webpage is often complementary to its template. Therefore, detecting templates can allow indexers to identify the main content that is usually inside a specific pagelet of the template.

Modern crawlers and indexers do not treat all terms in a webpage in the same way. Webpages are preprocessed to identify the template because template extraction allows them to identify those pagelets that only contain noisy information such as advertisements and banners. This content should not be indexed. Indexing the non-content part of templates not only affects accuracy, it also affects performance and is, in general, a waste of storage space, bandwidth and time.

Template extraction helps indexers to isolate the main content. This allows us to enhance indexers by assigning higher weights to the really relevant terms. Once templates have been extracted, they should

be processed for indexing. Links in templates allow indexers to discover the topology of a website (e.g., through navigational content such as menus), thus identifying the main webpages. They are also essential to compute pageranks.

Gibson et al. [7] determined that templates represent between 40% and 50% of data on the Web and that around 30% of the visible terms and hyperlinks appear in templates. This justifies the importance of template removal [17, 15] for web mining and search.

Our approach to template extraction is based on the DOM structures that represent webpages. Roughly, given a webpage in a website, we first identify a set of webpages that are likely to share a web template with it, and then, we analyze these webpages to identify the part of their DOM trees that is common. This slice of the DOM tree is returned as the template.

The technique exploits a new idea to automatically find a set of webpages that potentially share a web template. Roughly, we detect the template's menu and analyze the links of the menu to identify a set of mutually linked webpages. One of the main functions of a template is in aiding navigation, thus almost all web templates provide a large number of links, shared by all webpages implementing the template. Locating the menu allows us to identify in the topology of the website the main webpages of each category or section. These webpages very likely share the same template. This idea is simple but powerful and, contrarily to other approaches, it allows the technique to only analyze a reduced set of webpages to identify the web template.

The rest of the paper has been structured as follows: In Section 2 we discuss the state of the art and show some problems of current techniques that can be solved with our approach. In Section 3 we provide some preliminary definitions and useful notation. Then, in Section 4, we present our technique with examples and explain the algorithms used. In Section 5 we give some details about the implementation and show the results obtained from a collection of benchmarks. Finally, Section 6 concludes.

2 Related Work

Template detection and extraction are hot topics due to their direct application to web mining, searching, indexing, and web development. For this reason, there are many approaches that try to face this problem. Some of them have been presented in the CleanEval competition [2], which periodically proposes a collection of examples to be analyzed with a gold standard. The examples proposed are especially though for boilerplate removal, and content extraction.

Content Extraction is a discipline very close to template extraction. Content extraction tries to isolate the pagelet with the main content of the webpage. It is an instance of a more general discipline called *Block Detection* that tries to isolate every pagelet in a webpage. There are many works in these fields (see, e.g., [9, 16, 4]), and all of them are directly related to template extraction.

In the area of template extraction, there are three main different ways to solve the problem, namely, (i) using the textual information of the webpage (i.e., the HTML code), (ii) using the rendered image of the webpage in the browser, and (iii) using the DOM tree of the webpage.

The first approach is based on the idea that the main content of the webpage has more density of text, with less labels. For instance, the main content can be identified selecting the largest contiguous text area with the least amount of HTML tags [6]. This have been measured directly on the HTML code by counting the number of characters inside text, and characters inside labels. This measure produce a ratio called CETR [16] used to discriminate the main content. Other approaches exploit densitometric features based on the observation that some specific terms are more common in templates [12, 10].

The second approach assumes that the main content of a webpage use to be in the central part and

(at least partially) visible without scrolling [3]. This approach has been less studied because rendering webpages for classification is a computational expensive operation [11].

The third approach is where our technique falls. While some works try to identify pagelets analyzing the DOM tree with heuristics [1], others try to find common subtrees in the DOM trees of a collection of webpages in the website [17, 15]. Our technique is similar to these last two works.

Even though [17] uses a method for template extraction, its main goal is to remove redundant parts of a website. For this, they use the Site Style Tree (SST), a data structure that is constructed by analyzing a set of DOM trees and recording every node found, so that repeated nodes are identified by using counters in the SST nodes. Hence, an SST summarizes a set of DOM trees. After the SST is built, they have information about the repetition of nodes. The most repeated nodes are more likely to belong to a noisy part that is removed from the webpages.

In [15], the approach is based on discovering optimal mappings between DOM trees. This mapping relates nodes that are considered redundant. Their technique uses the RTDM-TD algorithm to compute a special kind of mapping called *restricted top-down mapping* [13]. Their objective, as ours, is template extraction, but there are two important differences. First, we compute another kind of mapping to identify redundant nodes. Our mapping is more restrictive because it forces all nodes that form pairs in the mapping to be equal. Second, in order to select the webpages of the website that should be mapped to identify the template, they pick random webpages until a threshold. In their experiments, they approximated this threshold as a few dozen of webpages. In our technique, we do not select the webpages randomly, we use a method to identify the webpages linked by the main menu of the website because they highly likely contain the template. We only need to explore a few webpages to identify them. Moreover, contrarily to us, they assume that all webpages in the website share the same template, and this is a strong restriction for many websites.

3 Preliminaries

In this paper webpages are represented with a DOM tree $T = (N, E)$ where nodes are labelled with their corresponding HTML tags (see Figure 1). $|T|$ denotes the number of nodes in T. $T[i]$ denotes the i node of T whose position in a preorder traversal of T is i. $root(T)$ denotes the root node of T. Given a node $T[i] \in N$, $parent(i)$ represents node $T[j]$ such that $(T[j], T[i]) \in E$. $subtree(T[i])$ denotes the subtree of T whose root is $T[i]$.

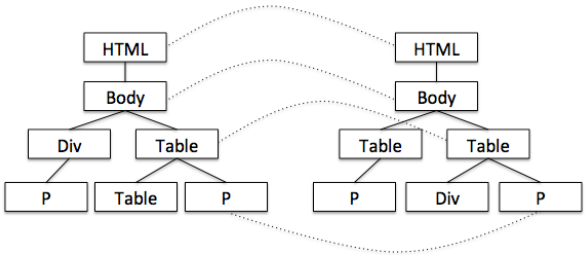

Figure 1: Top-down restricted mapping between DOM trees

In order to identify the part of the DOM tree that is common in a set of webpages, our technique uses an algorithm that is based on the notion of mapping. A mapping establishes a correspondence between the nodes of two trees.

Definition 3.1 *[14] A* mapping *from a tree T to a tree T' is any set M of pairs of integers satisfying:*

1. *For any pair (i, j) in M,*
   (a) $1 \leq i \leq |T|, 1 \leq j \leq |T'|$.
2. *For any two pairs (i_1, j_1) and (i_2, j_2) in M,*
   (a) $i_1 = i_2$ *iff* $j_1 = j_2$;
   (b) $i_1 < i_2$ *iff* $j_1 < j_2$;
   (c) $T[i_1]$ *is an ancestor (descendant) of* $T[i_2]$ *iff* $T'[j_1]$ *is an ancestor (descendant) of* $T'[j_2]$.

In order to identify web templates, we are interested in a very specific kind of mapping that we call *exact top-down mapping* (ETDM).

Definition 3.2 *A mapping M between two trees T_1 and T_2 is said to be* exact top-down *if and only if*

- *exact: for every pair $(i, j) \in M$, $T_1[i] = T_2[j]$.*
- *top-down: for every pair $(i, j) \in M$, with $T_1[i] \neq root(T_1)$ and $T_2[j] \neq root(T_2)$, there is also a pair $(parent(i), parent(j)) \in M$.*

This mapping is even more restrictive than other mappings such as, e.g., the *restricted top-down mapping* (RTDM) introduced in [13]. While RTDM permits the mapping of different nodes (e.g., a node labelled with *table* with a node labelled with *div*), ETDM only allows the mapping of nodes that are equal. Figure 1 shows an example of a ETDM. We can now give a definition of template using ETDM.

Definition 3.3 *Let $P = \{p_1 \dots p_n\}$ be a collection of webpages. A* template *of P is a tree $T = (N, E)$ where*

- *nodes: $N = \{p_j[m] \mid \forall p_j, p_k, 1 \leq j \neq k \leq n \ . \ (m, m') \in M_{j,k}\}$ where $M_{j,k}$ is a exact top-down mapping between p_j and p_k.*
- *edges: $E = \{(a, b) \in \bigcap_{1 \leq i \leq n} E_i \mid a, b \in N \ \wedge \ p_i = (N_i, E_i)\}$.*

We formalize a template of a set of webpages as a new webpage extracted from their intersection (computed by means of a ETDM between all the webpages).

4 Template extraction

Web templates are often composed of a set of pagelets. Two of the most important pagelets in a webpage are the menu and the main content. For instance, in Figure 2 we see two webpages that belong to the "News" portal of BBC. At the top of the webpages we see the main menu containing links to all BBC portals. We can also see a submenu under the big word "News". The left webpage belongs to the "Technology" section, while the right webpage belongs to the "Science & environment" section. Both share the same menu, submenu, and general structure. In addition to the main content, there is a common pagelet called "Top Stories" with the most relevant news, and another one called "Features and analysis".

In our technique we input a webpage (called key page) and we output its template. To infer the template, we analyze some webpages from the (usually huge) universe of related webpages. Therefore, we need to decide what related webpages should be analyzed. Our approach is very simple yet powerful:

Figure 2: Pagelets in the BBC web template

1. Starting from the key page, we identify a complete subdigraph in the website topology[1], and then
2. we extract the template by comparing the DOM tree of the key page with the DOM trees of the nodes in the complete subdigraph. The template is the intersection between the key page and all DOM trees in the subdigraph. This intersection is computed with a ETDM between the DOM trees.

4.1 Finding a complete subdigraph in a website topology

Given a website topology, a complete subdigraph (CS) is a collection of webpages that are pairwise mutually linked. It is a n-complete subdigraph (n-CS) if it is formed by n nodes. The interest in complete subdigraphs comes from the fact that the webpages linked by the items in a menu usually form a CS. This idea is original and is a way of identifying the webpages that contain the menu and, at the same time, they are the roots of the sections linked by the menu. The following example illustrates why menus provide very useful information about the interconnection of webpages in a given website.

Example 4.1 *Consider a website where all webpages share the same template, and this template has a main menu that is present in all webpages, and a submenu for each item in the main menu. This is the case of, e.g., the BBC webpages shown in Figure 2. The site map of this website may be represented with the topology shown in Figure 3.*

In this figure, each node represents a webpage and each edge represents a link between two webpages (we only draw some of the edges for clarity). Solid edges are bidirectional, and dashed and dotted edges are directed. Black nodes are the webpages pointed by the main menu. Because the main menu is present in all webpages, then all nodes are connected to all black nodes. Therefore all black nodes together form a complete graph (i.e., there is an edge between each pair of nodes). Grey nodes are the webpages pointed by the submenu, thus, all grey nodes together also form a complete graph. White nodes are webpages inside one of the categories of the submenu, therefore, all of them have a link to all black and all grey nodes.

Of course, not all webpages in a website implement the same template, and some of them only implement a subset of a template. For this reason, one of the main problems of template extraction is

[1] In our implementation, we restrict our search to webpages in the same domain as the key page. This is not necessary, but significantly increases the performance with a small (rarely appreciable) cost in the precision.

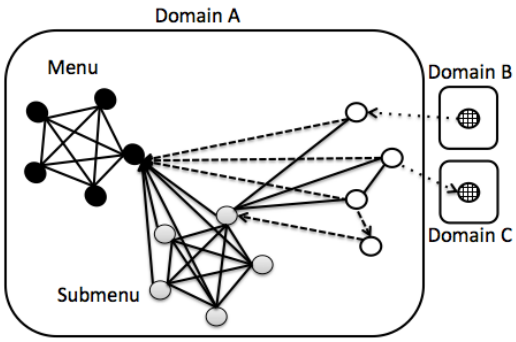

Figure 3: Site map of BBC

deciding what webpages should be analyzed. Minimizing the number of webpages analyzed is essential to reduce the load of web crawlers. In our technique, we assume that the template contains at least one menu, and we try to identify the webpages pointed out by the menu. Therefore, we only need to investigate the webpages linked by the key page, because they will for sure contain a CS that represents the menu.

In order to increase precision, we search for a CS that contains enough webpages that (hopefully) implement the template. This CS can be identified with Algorithm 1. The algorithm uses three trivial functions: $loadPage(link)$ that loads the webpage pointed by the input link, $getLinks(webpage)$ that returns the collection of links in the input webpage, and function *size* that computes set cardinality. Observe that the main loop iteratively explores the links of the *initialPage* (i.e., the key page) until it founds a n-CS. Note also that it only loads those pages needed to find the n-CS, and it stops when the n-CS has been found. We want to highlight the following mathematical expression:

$$CS = \{ls \in \mathscr{P}(links) \mid link \in ls \wedge \forall l, l' \in ls \,.\, (l \to l'), (l' \to l) \in connections\}$$

used to find the set of all CS that can be constructed with the current $link$.

Algorithm 1 Extract a n-CS from a website

Input: An *initialLink* that points to a webpage and an integer n
Output: A set of links to webpages that together form a n-CS.
If a n-CS cannot be formed, then they form the biggest m-CS with m < n.

begin
$initialPage =$ loadPage($initialLink$);
$reachableLinks =$ getLinks($initialPage$);
$links = \{initialLink\}$;
$connections = \{\}$;
$BCS = \{\}$;
foreach ($link$ **in** $reachableLinks$)
$page =$ loadPage($link$);
$existingLinks =$ getLinks($page$) $\cap$ $reachableLinks$;
$links = links \cup \{link\}$;
$connections = connections \cup \{(initialLink \to link)\}$;
$connections = connections \cup \{(link \to existingLink) \mid existingLink \in existingLinks\}$;
$CS = \{ls \in \mathscr{P}(links) \mid link \in ls \wedge \forall l, l' \in ls \,.\, (l \to l'), (l' \to l) \in connections\}$;
$MCS = cs \in CS$ such that $\forall cs' \in CS \,.\, \text{size}(cs) \geq \text{size}(cs')$;
if size(MCS) $= n$ **then return** MCS;
if size(MCS) $>$ size(BCS) **then** $BCS = MCS$;
return BCS;
end

4.2 Template extraction from a complete subdigraph

After we have found the set of webpages linked by the menu of the site (the complete subdigraph), we identify a ETDM between the key page and all webpages in the set. For this, initially, the template is considered to be the first webpage in the set. Then, we compute a ETDM between the template and the second webpage in the set. The result is the new refined template, that is further refined with another ETDM with the next webpage, and so on until all webpages have been processed. This process is formalized in Algorithm 2, that uses function *ETDM* to compute the biggest ETDM between two trees.

Algorithm 2 Extract a template from a set of webpages

Input: A set of n webpages $P = \{p_1 \dots p_n\}$
Output: A template

begin
 $template = p_1$;
 $i = 2$;
 while $(i <= n)$
  $template = ETDM(template, p_i)$;
  $i++$;
 return $template$;
end

function $ETDM$(tree $T_1 = (N_1, E_1)$, tree $T_2 = (N_2, E_2)$)
 $r_1 = root(T_1)$;
 $r_2 = root(T_2)$;
 if $(r_1 == r_2)$
 then
  nodes = $\{r_1\}$;
  edges = $\{\}$;
  for each $n_1 \in N_1, n_2 \in N_2 \,.\, n_1 == n_2, (r_1, n_1) \in E_1$ and $(r_2, n_2) \in E_2$ **do**
   $edges = edges \cup \{(r_1, n_1)\}$;
   $(nodes\_st, edges\_st) = ETDM(subtree(n_1), subtree(n_2))$;
   $nodes = nodes \cup nodes\_st$;
   $edges = edges \cup edges\_st$;
  return $(nodes, edges)$;
 else
  return $(\{\}, \{\})$;

5 Implementation

We implemented the technique presented in this paper as a toolbar Firefox's plugin. We can browse on the Internet as usual. Then, when we want to extract the template of a webpage, we only need to press the "Extract Template" button and the tool automatically (internally) loads the associated webpages to extract the template. The template is then displayed in the browser as any other webpage.

We conducted several experiments with real, online webpages to provide a measure of the average performance regarding recall, precision and the F1 measure (see, e.g., [8] for a discussion on these metrics). For the experiments, we selected a collection of domains with different layouts and page structures in order to study the performance of the technique in different contexts (e.g., company websites, news articles, forums, etc.). Then, we randomly selected the final evaluation set. We determined the template of each webpage (the gold standard) by:

1. Downloading the complete website of each benchmark (the key page together with all reachable webpages from it in the same domain).

2. Four different engineers did the following independently:
   (a) Manually exploring the key page and the webpages accessible from it to decide what part of the webpage is the template.
   (b) Printing the key page in paper and marking the template.
3. The four engineers met and together decided what was the template, and they marked it in a printed version of the key page.
4. Each element marked in the printed page was mapped to the DOM tree of the key page. All elements in the DOM tree that did not belong to the template were included in an HTML class *non-template* (i.e., we enriched the HTML code of the key page with a new class). This class was later used by an algorithm that we programmed to evaluate the results obtained by our tool.

Table 1 summarizes the results of the performed experiments, which were computed with a 4-CS. The first column contains the URLs of the evaluated webpages. For each benchmark, column `DOM nodes` shows the number of nodes of the whole DOM tree associated with this benchmark; column `Template` shows the number of nodes of the gold standard template; column `Retrieved` shows the number of nodes that were identified by the tool as the template; column `Recall` shows the number of template nodes correctly retrieved divided by the total number of actual template nodes; column `Precision` shows the number of template nodes correctly retrieved divided by the total number of retrieved nodes; finally, column `F1` shows the F1 metric that is computed as $(2*P*R)/(P+R)$ being P the precision and R the recall.

Benchmark	DOM nodes	Template	Retrieved	Recall	Precision	F1
www.felicity.co.uk	300 nodes	232 nodes	232 nodes	100 %	98,72 %	99,36 %
www.dsic.upv.es/~dinsa	241 nodes	74 nodes	74 nodes	100 %	90,24 %	94,87 %
www.engadget.com	1818 nodes	768 nodes	763 nodes	99,35 %	99,22 %	99,28 %
www.bbc.co.uk/news/	2991 nodes	604 nodes	552 nodes	91,39 %	67,73 %	77,80 %
www.vidaextra.com	2331 nodes	1137 nodes	18 nodes	1,58 %	100 %	3,12 %
www.ox.ac.uk/staff/	948 nodes	538 nodes	104 nodes	19,33 %	92,86 %	32,00 %
clinicaltrials.gov	543 nodes	389 nodes	378 nodes	97,17 %	96,92 %	97,05 %
en.citizendium.org	992 nodes	399 nodes	318 nodes	79,70 %	91,64 %	85,25 %
www.filmaffinity.com	1316 nodes	340 nodes	340 nodes	100 %	98,84 %	99,42 %
www.cnn.com	3860 nodes	192 nodes	148 nodes	77,08 %	98,67 %	86,55 %
www.lashorasperdidas.com	1822 nodes	553 nodes	252 nodes	45,57 %	100 %	62,61 %
labakeryshop.com	1368 nodes	403 nodes	175 nodes	43,42 %	96,15 %	59,83 %
www.dsic.upv.es/~jsilva/wwv2013	197 nodes	163 nodes	163 nodes	100 %	96,45 %	98,19 %
www.thelawyer.com	2708 nodes	949 nodes	742 nodes	78,19 %	76,50 %	77,33 %
www.us-nails.com	250 nodes	184 nodes	184 nodes	100 %	83,64 %	91,09 %
www.informatik.uni-trier.de	3083 nodes	117 nodes	8 nodes	6,84 %	100 %	12,8 %
www.wayfair.co.uk	1950 nodes	1507 nodes	697 nodes	46,25 %	99,57 %	63,16 %
catalog.atsfurniture.com	340 nodes	301 nodes	301 nodes	100 %	99,01 %	99,50 %
www.glassesusa.com	1952 nodes	1708 nodes	1659 nodes	97,13 %	99,70 %	98,40 %
www.mysmokingshop.co.uk	575 nodes	407 nodes	407 nodes	100 %	98,31 %	99,15 %
Average	1479 nodes	548 nodes	376 nodes	74,15 %	94,21 %	76,84 %

Table 1: Benchmark results

Experiments reveal an average precision of more than 94%, and an average recall of almost 75% even though two experiments produced a recall under 7%. These experiments are really difficult ones that produce the same problem in previous techniques such as [15]. The problem in these benchmarks is that some webpages pointed by the main menu do not use the template (i.e., some webpages feature the menu, or the necessary links in some form, but they do not use the template). Therefore, the intersection with these webpages produce an almost empty page, and this is the cause of the low recall.

6 Conclusions

Web templates are an important tool for website developers. By automatically inserting content into web templates, website developers and content providers of large web portals achieve high levels of productivity, and they produce webpages that are more usable thanks to their uniformity.

This work presents a new technique for template extraction. The technique is useful for website developers because they can automatically extract a clean HTML template of any webpage. This is particularly interesting to reuse components of other webpages. Moreover, the technique can be used by other systems and tools such as indexers or wrappers as a preliminary stage. Extracting the template allows them to identify the structure of the webpage, and the topology of the website by analyzing the navigational information of the template. In addition, the template is useful to identify menus, repeated advertisement panels, and what is particularly important, the main content.

Our technique uses the menus of a website to identify a set of webpages that share the same template with a high probability. Then, it uses the DOM structure of the webpages to identify the blocks that are common to all webpages. These blocks together form the template. To the best of our knowledge, the idea of using the menus to locate the webpages that share the template is new, and it allows us to quickly find a set of webpages from which we can extract the template. This is especially interesting for performance, because loading webpages to be analyzed is expensive, and this part of the process is minimized in our technique. Our implementation and experiments have shown the usefulness of the technique.

This technique could be also used for content extraction. Detecting the template of a webpage is very helpful to detect the main content. Firstly, the main content must be formed by DOM nodes that do not belong to the template. Secondly, the main content is usually inside one of the pagelets of the template that are more centered and visible, and with a higher concentration of text.

For future work, we plan to investigate a strategy to further reduce the amount of webpages loaded with our technique. The idea is to directly identify the menu in the key page by measuring the density of links in its DOM tree. The menu has probably one of the higher densities of links in a webpage. Therefore, our technique could benefit from measuring the links–DOM nodes ratio to directly find the menu in the key page, and thus, a complete subdigraph in the website topology.

7 Acknowledgements

This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación (Secretaría de Estado de Investigación)* under grant TIN2008-06622-C03-02 and by the *Generalitat Valenciana* under grant PROMETEO/2011/052. David Insa was partially supported by the Spanish Ministerio de Eduación under FPU grant AP2010-4415. Salvador Tamarit was partially supported by the Spanish MICINN under FPI grant BES-2009-015019.

References

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th international conference on World Wide Web*, WWW'02, pages 580–591, New York, NY, USA, 2002. ACM.

[2] M. Baroni, F. Chantree, A. Kilgarriff, and S. Sharoff. Cleaneval: a competition for cleaning web pages. In *LREC*. European Language Resources Association, 2008.

[3] R. Burget and I. Rudolfova. Web page element classification based on visual features. In *Proceedings of the 2009 First Asian Conference on Intelligent Information and Database Systems*, ACIIDS'09, pages 67–72, Washington, DC, USA, 2009. IEEE Computer Society.

[4] E. Cardoso, I. Jabour, E. Laber, R. Rodrigues, and P. Cardoso. An efficient language-independent method to extract content from news webpages. In *Proceedings of the 11th ACM symposium on Document engineering*, DocEng'11, pages 121–128, New York, NY, USA, 2011. ACM.

[5] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *Proceedings of the 10th international conference on World Wide Web*, WWW'01, pages 211–220, New York, NY, USA, 2001. ACM.

[6] A. Ferraresi, E. Zanchetta, M. Baroni, and S. Bernardini. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4)*, 2008.

[7] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW'05, pages 830–839, New York, NY, USA, 2005. ACM.

[8] T. Gottron. Evaluating content extraction on html documents. In *Proceedings of the 2nd International Conference on Internet Technologies and Applications*, ITA '07, pages 123–132, Wrexham, North Wales, 2007.

[9] T. Gottron. Content code blurring: A new approach to content extraction. In *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, DEXA'08, pages 29–33, Washington, DC, USA, 2008. IEEE Computer Society.

[10] C. Kohlschütter. A densitometric analysis of web template content. In *Proceedings of the 18th international conference on World wide web*, WWW'09, pages 1165–1166, New York, NY, USA, 2009. ACM.

[11] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM'10, pages 441–450, New York, NY, USA, 2010. ACM.

[12] C. Kohlschütter and W. Nejdl. A densitometric approach to web page segmentation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM'08, pages 1173–1182, New York, NY, USA, 2008. ACM.

[13] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, WWW'04, pages 502–511, New York, NY, USA, 2004. ACM.

[14] K. C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, July 1979.

[15] K. Vieira, A. S. da Silva, N. Pinto, E. S. de Moura, J. a. M. B. Cavalcanti, and J. Freire. A fast and robust method for web page template detection and removal. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM'06, pages 258–267, New York, NY, USA, 2006. ACM.

[16] T. Weninger, W. H. Hsu, and J. Han. CETR: content extraction via tag ratios. In *Proceedings of the 19th International Conference on World Wide Web, WWW'10, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 971–980. ACM, 2010.

[17] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD'03, pages 296–305, New York, NY, USA, 2003. ACM.

Model Checking GSM-Based Multi-Agent Systems

Pavel Gonzalez, Andreas Griesmayer, Alessio Lomuscio
Department of Computing, Imperial College London
{pavel.gonzalez09, a.griesmayer, a.lomuscio}@imperial.ac.uk

Artifact systems are a novel paradigm for implementing service oriented computing. Business artifacts include both data and process descriptions at interface level thereby providing more sophisticated and powerful service inter-operation capabilities. In this paper we put forward a technique for the practical verification of business artifacts in the context of multi-agent systems. We extend GSM, a modelling language for artifact systems, to multi-agent systems and map it into a variant of AC-MAS, a semantics for reasoning about artifact systems. We introduce a symbolic model checker for verifying GSM-based multi-agent systems. We evaluate the tool on a scenario from the service community.

1 Introduction

It has long been argued [19, 5] that agents are a fitting paradigm for service oriented computing (SOC). Indeed, agent-based research has contributed a wealth of techniques ranging from verification [16], protocols [20] and actual prototype implementations [1]. SOC is currently a fast moving research area with significant industrial involvement where highly scalable implementations play a key role. Agent-based solutions can shape developments in SOC if they remain anchored to emerging paradigms being put forward by the leading players in the area.

An increasingly popular paradigm being investigated in SOC is that of *business artifacts* [7]. In this approach *data*, not only processes, play a key part in the service description and implementations. While in traditional service composition, processes are advertised at interface level, in the artifact approach both processes and the data structures are given equal prominence.

Guard-Stage-Milestone (GSM) has recently been put forward [14, 13] as a language for implementing business artifacts. In line with the AI and agent-based tradition, GSM is a declarative language. GSM provides a description of *stages*, which are clusters of topical activity pertaining to some artifact data-structure. Stages are governed by *guards* controlling their activation and *milestones* determining whether or not the stage goals have been reached.

While business artifacts are an attractive methodology for developing business processes and GSM-based services are a rapidly evolving area of research, they lack fully-fledged automatic methodologies for verification, orchestration and choreography. In this paper we put forward a technique and an implementation for the practical verification of business artifacts from a multi-agent system perspective. Specifically, we give a MAS-based formal model to GSM systems and define the model checking problem on this model. We have built an implementation to verify automatically whether a GSM system, including a number of agents, satisfies given temporal-epistemic specifications which may include quantification over artifact instances. We test the technique against an application developed by IBM.

Several contributions have so far studied the verification problem from a theoretical perspective [8, 11, 2, 4]. The results obtained identify fragments of decidable settings either through restrictions on the specification language or the semantics. While these results are certainly valuable, they provide no constructive methodology for the practical verification of GSM-based systems.

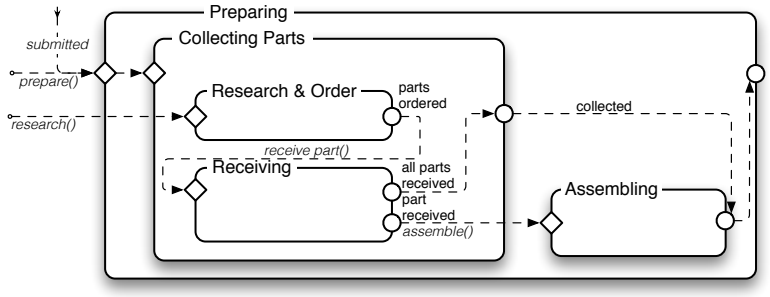

Figure 1: A lifecycle model.

Closer to the work here presented is [10] where we introduced `GSMC`, a model checker for GSM. However, the semantics of the underlying formalism is one of plain transition systems and no support for agents in the system is provided. With no agents being present, no support is offered for views and windows, two key concepts that we fully support here. Additionally, as our concern was focused purely on the artifact system, the specification language only supports temporal logic, thereby making impossible to verify the information-theoretic properties of agents throughout an exchange as we do here.

2 The Guard-Stage-Milestone Artifact Model

The Guard-Stage-Milestone (GSM) approach to artifact systems [7] is particularly suitable for large unstructured processes where users have the freedom to decide what actions they perform and in what order. GSM is substantially influencing the emerging Case Management Modelling Notation standard [17]. IBM Watson developed `Barcelona`, an application for modelling and execution of GSM-based artifact systems [12].

We define a GSM model Γ in line with [13]. *Artifact types*, which correspond to classes of key business entities, provide the core structure for the model. Each type has a *lifecycle model*, which describes the structure of the business process, and an *information model*, which gives a view of the data.

Definition 1 (Artifact Type) *An* artifact type *AT is a tuple $AT = \langle R, Att, Lcyc \rangle$, where R is the* name *of the artifact type; Att is the* information model *as set of* attributes*; and $Lcyc$ is the* lifecycle model.

GSM provides a declarative, hierarchical mechanism for specifying *lifecycle models* based on *milestones*, which correspond to operational objectives that an artifact aims to achieve, *stages*, which represent clusters of activity designed to achieve milestones, and *guards*, which trigger activities in a stage when a certain condition is fulfilled. Stages are organised hierarchically, where the roots are called *top-level stages* and the leaves are called *atomic stages*. Stages can run in parallel and own at least one milestone and one guard. A stage becomes *open* when one of its guards is fulfilled and *closed* when one of its milestones is *achieved*. A milestone can also be *invalidated* when certain conditions are met. Figure 1 illustrates a typical lifecycle model with several nested stages, where $\diamond$ represents guards and $\circ$ represents milestones.

The *information model* Att is partitioned into the set Att_{data} of *data attributes* to hold data and the set Att_{status} of *status attributes* to capture the state of the lifecycle model. Each stage (resp. milestone),

has a Boolean status attribute in Att_{status}, which is true iff the stage is *active* (resp. the milestone has been *achieved*). We write *Dom* for the domain of attributes in *Att* including the undefined value $\perp$. An *artifact instance* of an artifact type AT is a tuple $\iota = (AT, A_1 : c_1, \ldots, A_k : c_k)$, where $c_i \in Dom(A_i)$.

The artifact system interacts with the environment by sending and receiving messages with payloads, where a message with a specific payload is called a *typed external event*.

Definition 2 (Event Type) *An* event type ET *is a tuple* $ET = \langle E, AT, A_1, \ldots, A_l \rangle$*, where* E *is the* name *of the event type,* AT *is an artifact type, and* $A_i \in Att_{data}$*, where* Att_{data} *is the set of data attributes of* AT*.*

An *event instance* of an event type E is a tuple $e = (\iota, A_1 : c_1, \ldots, A_l : c_l)$, where ι is an artifact instance and $c_i \in Dom(A_i)$.The tuple $p = (A_1 : c_1, \ldots, A_l : c_l)$ is the *payload* of e. We now have all the ingredients to define a GSM model.

Definition 3 (GSM Model) *A* GSM model Γ *is a set of n artifact types* AT_i *for* $i \leq n$ *and m event types* ET_j *for* $j \leq m$*.*

Definition 4 (Snapshot of GSM Model) *A* pre-snapshot *of* Γ *is an assignment* Σ *that maps each attribute* $A \in Att_\iota$ *of each active artifact instance* ι *to an element of* $Dom(A)$*. A* snapshot *of* Γ *is a pre-snapshot that satisfies the following GSM invariants: all sub-stages of a closed stage are closed; all milestones of an open stage are false; at most one milestone of a stage can be achieved at any time.*

The operational semantics of GSM is based on the notion of a *business step* (B-step). This is an atomic unit that corresponds to the effect of processing one incoming event into the state of the artifact system. A B-step has the form of a tuple $(\Sigma, e, \Sigma', Gen)$, where Σ, Σ' are snapshots, e is an incoming external event, and *Gen* is a set of outgoing external events generated by opening atomic stages during the B-step.

The progress of the lifecycle is driven by incoming events, which are called *applicable* if the lifecycle is ready to consume them. When an event is consumed by the artifact system, its payload is copied to the information model and the lifecycle model is updated. The opening of an atomic stage activates a task associated with the stage. It either performs an *automated system task*, such as the creation of a new instance, or corresponds to an operation outside the artifact system, in which case a service call is sent to the environment. In both cases, the task completion is marked by an event.

Both milestones and guards are controlled declaratively through *sentries*. A sentry of an artifact instance ι is an expression $\chi(\iota)$, which has one of the following three forms: "*on* $\xi(\iota)$ *if* $\varphi(\iota)$"; "*on* $\xi(\iota)$"; "*if* $\varphi(\iota)$", where $\xi(\iota)$ is an event expression for the *triggering event* e of ι and $\varphi(\iota)$ is a well-formed formula over the instances occurring in the system. Milestones have one or more *achieving sentries* and optionally one or more *invalidating sentries*.

B-steps are computed according to the *incremental semantics* defined in [13] using a sequence of *Prerequisite-Antecedent-Consequent* (PAC) rules, which govern opening of stages and achieving of milestones. A PAC rule for a GSM model Γ is a tuple $\rho = (\pi, \alpha, \gamma)$, where: the prerequisite π is a formula on the attributes in Att_{status}; the antecedent α is a sentry based on attributes in Att, the internal events over Att_{status} and external event types ET; the consequent γ is an internal event $\odot\sigma$, where $\odot \in \{+, -\}$ is the update of a status attribute $\sigma \in Att_{status}$. In response to a single incoming external event, the artifact system fires all *applicable* rules until no rule can be fired.

To ensure the semantics is well-founded, an acyclicity condition is imposed. This is defined by using the *polarised dependency graph*, denoted $PDG(\Gamma)$. A GSM model Γ is *well-formed* if $PDG(\Gamma)$ is acyclic, i.e., there is no cyclic interdependence between the PAC rules of Γ.

3 Artifact-Centric Multi-Agent Systems

To analyse interactions within a GSM-based artifact system, we use artifact-centric multi-agent systems (AC-MAS) [11, 3], a semantics based on interpreted systems [18, 9], and a temporal-epistemic specification language with quantification over artifact instances.

3.1 Formal Model

In an AC-MAS a set of agents $\mathscr{A}$ share an environment E constituted by the artifact system, i.e., the underlying elements of the environment are evolving artifacts of type R. The environment and an agent $i \in A$ have a local state (L_E and L_i respectively), where the agent can observe parts of the environment (i.e., some of the artifact instances in it). The local state of an agent thus comprises private data and observable aspects of the artifact system. We write $l_E(s)$ to represent the local state of the environment in the global state s, and $l_i(s)$ to represent the local state of agent i.

An agent i and the environment E communicate by synchronisation on actions, where Act_E corresponds to events enabled by the artifact system, and $Act_i \subseteq Act_E \cup \{skip\}$ is the set of *local actions* corresponding to events that can be executed by the agent and the idle action *skip*. As in plain interpreted systems, protocols are used to select the actions performed in a given state, where $P_E : L_E \to 2^{Act_E}$ is the environment's *protocol function*, which enables executable events depending on the local state of the artifact system and $P_i : L_i \to 2^{Act_i}$ is the *protocol function* of agent i.

Following the terminology of [3] we define an AC-MAS as the composition of the environment and a number of agents as follows:

Definition 5 (AC-MAS) *Given an environment E and a set of agents $\mathscr{A}$, an* artifact-centric multi-agent system *is a tuple $\mathscr{P} = \langle S, \mathscr{I}, \tau \rangle$, where $S \subseteq L_E \times L_1 \times \cdots \times L_n$ is the set of* reachable global states*, $\mathscr{I}$ is the* initial state*, and $\tau : S \times Act \to 2^S$ with $Act = Act_E \times Act_1 \times \cdots \times Act_n$ is the* global transition function*. The transition $\tau(s, \alpha)$ is defined for $\alpha = (a_E, a_1, \ldots, a_n)$ iff $a_E \in P_E(l_E(s))$, and $\exists_{0 \leq i < n} : a_i \in P_i(l_i(s))$, $a_E = a_i \wedge \forall_{j \neq i} : a_j = skip$.*

Intuitively, the conditions on the transition relation limit the communication between agents and environment such that environment and agent agree on the action. The environment enables actions when the artifact system is ready to consume them, while the agent i decides on the actions to execute depending on a local strategy encoded in P_i. Only one agent can interact with the environment at a time while the others are idle.

We write $s \to s'$ iff there exists an α, such that $s' \in \tau(s, \alpha)$, and call s' the *successor* of s. A *run* r from s is an infinite sequence $s^0 \to s^1 \to \ldots$ with $s^0 = s$. We write $r[i]$ for the i-th state in the run and r_s for the set of all runs starting from s. A state s' is *reachable* from s if there is a run from s that contains s'. In line with the semantics of epistemic logic [9], we say that the states s and s' are *epistemically indistinguishable* for agent i, or $\sim_i$, iff $l_i(s) = l_i(s')$.

3.2 The Logic IQ-CTLK

We are interested in specifying temporal-epistemic properties of agents interacting with the artifact system, as well as the system itself. Since GSM supports the dynamic creation of unnamed artifacts, the properties need to be independent of the actual number or possible IDs of artifact instances in the system. We therefore define a temporal-epistemic logic that supports quantification over the artifact instances. We call the logic IQ-CTLK, for *Instance Quantified CTLK*, where CTLK is the usual epistemic logic on

branching time. It is a subset of FO-CTLK where quantification can only be over artifact instances but not data. The syntax is defined in BNF notation as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E(\varphi U \varphi) \\ \mid K_i\varphi \mid \forall x : R\ \varphi \mid \exists x : R\ \varphi$$

where R is the name of an artifact type and p is an atomic proposition over the agents' private data and the attributes of active instances that are specified in terms of *instance variables* bound by the quantification operators. The quantified instance variables range over the active instances of a given artifact type R in the state where the quantification is evaluated and must be bound. We write $R(s)$ for the set of instances of type R in s. The remaining CTL operators can be constructed by combination of the ones given above. For example, $AG\ \forall x : OrderAF\ K_i x.sent$ encodes the property expressing that in any reachable state, agent i will eventually know that the attribute *sent* is set to true for every active instance of type *Order*.

We inductively define the semantics of IQ-CTLK over an AC-MAS $\mathscr{P}$ as follows. A formula φ is true in a state s of $\mathscr{P}$, written $(\mathscr{P},s) \models \varphi$, if:

$$\begin{array}{ll}
(\mathscr{P},s) \models p & \textit{iff } p \in s \\
(\mathscr{P},s) \models \neg\varphi & \textit{iff } (\mathscr{P},s) \not\models \varphi \\
(\mathscr{P},s) \models \varphi_1 \vee \varphi_2 & \textit{iff } (\mathscr{P},s) \models \varphi_1 \vee (\mathscr{P},s) \models \varphi_2 \\
(\mathscr{P},s) \models EX\varphi & \textit{iff } \exists_{s'} : s \to s' \wedge (\mathscr{P},s') \models \varphi \\
(\mathscr{P},s) \models EG\varphi & \textit{iff } \exists_{r \in r_s} : \forall_{i \geq 0} : (\mathscr{P}, r[i]) \models \varphi \\
(\mathscr{P},s) \models E(\varphi U \psi) & \textit{iff } \exists_{r \in r_s} : \exists_{k \geq 0} : (\mathscr{P}, r[k]) \models \psi \wedge \\
 & \quad \forall_{j<k} (\mathscr{P}, r(j)) \models \varphi \\
(\mathscr{P},s) \models K_i\varphi & \textit{iff } \forall s' \in S : s \sim_i s' \Rightarrow (\mathscr{P},s') \models \varphi \\
(\mathscr{P},s) \models \forall x : R\ \varphi & \textit{iff } \forall u \in R(s) : (\mathscr{P},s) \models \varphi[u/x] \\
(\mathscr{P},s) \models \exists x : R\ \varphi & \textit{iff } \exists u \in R(s) : (\mathscr{P},s) \models \varphi[u/x]
\end{array}$$

Given an AC-MAS model $\mathscr{P}$ and an IQ-CTLK specification φ, the model checking problem concerns the decision as to whether the formula φ holds at the initial state of $\mathscr{P}$.

4 Agent-Based GSM

Naturally, a GSM program only deals with the machinery related to the artifact system but does not provide a description of the agents interacting with it. To conduct the verification of agent-based GSM systems via model checking, we define A-GSM as an extension of GSM with a set of external agents, and give a formal mapping $f : A\text{-}GSM \to AC\text{-}MAS$, such that f preserves satisfaction of formulas in the specification language IQ-CTLK.

4.1 Agent Description

Here we outline how the agents are specified and interact with GSM to define an *A-GSM instance*. The behaviour of an agent is determined by the permitted access to the artifact system AS and by local decisions regarding events to send. The former is determined by an agent's *role*, while the latter are defined for each agent individually.

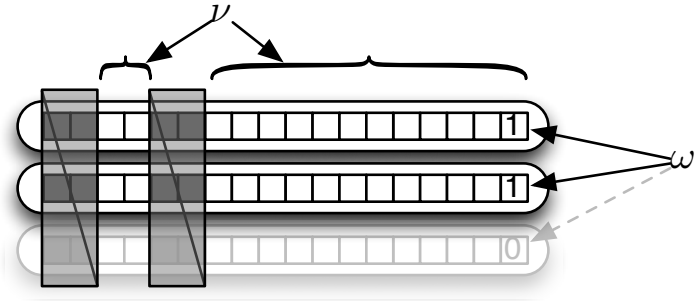

Figure 2: Static and Dynamic visibility in A-GSM.

The *role* is defined using the *view* ν for the visible attributes, the *window* ω to select the visible instances, and the *set of events* ε that are accepted by AS. While ν and ε are simple lists, $\omega_i(\iota)$ is a formula that is evaluated for a specific artifact instance ι and an agent i. The instance is exposed to the agent only if $\omega_i(\iota)$ evaluates to *true*. In addition to the role, the description of an agent also contains a *protocol* $\wp$ to determine its behaviour depending on the visible state of the AS, the agent's unique ID, and its private variables.

The concepts of ν, ω and ε are powerful tools to define the aspects agents can see and the ways they can interact with an artifact system. In Figure 2 the lines correspond to artifact instances that were created during run-time and the columns correspond to data attributes. ν defines a *static view* of the system, as it hides for each agent a fixed set of attributes depending on his role. For example, a *Customer* can only see that the state of an order moved from assembling to shipping, while a *Manufacturer* sees more detail, e.g., on suppliers. In contrast, ω gives a *dynamic* selection of the parts of the AS an agent can access in terms of the state of artifact instances as it hides complete instances depending on the current state. For instance, a *Manufacturer* may only see instances that represent unfinished orders while the window of a *Customer* can use the ID to restrict access to its own orders only.

Figure 3 outlines the description format. Visible data attributes are listed in the `view` field. The `window` field contains the formula for $\omega_i(\iota)$, where `$$` is a placeholder for the agent's ID. The field `instantiation` lists all artifact types that agents of this role may instantiate; the corresponding instantiation events are added to ε. To specify the status attributes and events that are added to ν and ε, the field `transformation` holds a set of GSM operators that allow to hide parts of the GSM model Γ. Valid commands are `hide_stage_status("S")` and `hide_milestone("m")` to hide the status attributes of stage S and milestone m respectively, and `delegate_sentry("s")` to remove events from ε_i if they are only used in sentry s. For convenience, the macro operators `condense_stage("S")` and `eliminate_stage("S")` hide all sub-stages or all information including guards and milestones respectively.

The private variables of an agent are defined in a list `var` of variable names $\bar{x}$ with their type and initial value. The `protocol` lists entries of the form `e :` γ `->` μ for all events e the agent can send to the AS. If multiple entries for the same event are given, they are treated as disjunction. The condition γ is given in terms of data attributes of the instance ι, the payload, and the private variables. It defines the protocol function $\wp_i(\iota,\bar{x})$, which gives the set of events e with their respective payloads that can

```
role Customer {
 view: CustomerId, ManufacturerId;
 window: CustomerId == $$;
 instantiation: CO;
 transformation:
   condense_stage(CO, Preparing);
};
agent Diogenes {
 role:  Customer;
 vars:  bool cancelled = false;
 protocol:
  Create_CO: CustomerId == "Diogenes"
             -> cancelled = cancelled,
  OnCancel: true -> cancelled = true;
};
```

Figure 3: An agent definition file.

be sent in the current state. The protocol also gives an update function $\mu_i(e,\bar{x})$, which computes new assignments for the local variables depending on the selected event and the local state of the agent. Note that by imposing conditions on the *payload* of an event e, $\wp$ also allows the agent to assign a specific value to its parameters, e.g., `CustomerId` is a parameter of `Create_CO`.

To handle *automated tasks*, we define an *AutoAgent*, which handles service calls and computations in the GSM model Γ and returns the result to the artifact system in form of an event. The *AutoAgent* holds pending tasks in a buffer t, has full access to Γ, and can send the return messages at any time, but is otherwise handled like any other agent.

4.2 Mapping to AC-MAS

We now establish the formal mapping $f : A\text{-}GSM \rightarrow AC\text{-}MAS$. Note that the semantics for the local states and protocols of agents in A-GSM are given in terms of AC-MAS. We define the map by constructing the environment $\langle L_E, Act_E, P_E\rangle$ from the GSM model Γ of a given artifact system and creating an agent $\langle L_0, Act_0, P_0\rangle$ for the *AutoAgent*, and $\langle L_i, Act_i, P_i\rangle$ with $1 \leq i \leq n$ for each external A-GSM agent. We identify a GSM event e with an AC-MAS action a and will omit the conversion in the following for ease of presentation. The sets of actions Act_E, Act_0, and Act_i are thus directly defined by the events the AS provides and the permissions of the agents.

Global state: To construct a global AC-MAS state $(l_E, l_0, \ldots, l_n) \in S$ from an snapshot Σ, an *AutoAgent* buffer t and the local agent states x_i, we identify l_E with Σ and l_0 with t. The local states $l_1, \ldots, l_n$ of the external agent comprise the state of the private variables x_i and the *projections* $\Sigma_{|i}$ of the environment snapshot such that:

$$\Sigma_{|i} = \{\iota \mid \exists_{\iota' \in \Sigma} : \omega_i(\iota') \wedge \iota = \iota'_{|_{V_i}}\}$$

where $\iota'_{|_{V_i}}$ is the restriction of the artifact instance ι' to the variables in V_i (variables not in V_i are replaced by $\perp$).

The initial state $\mathscr{I}$ is the empty state without any artifact instances in Σ or pending tasks in l_0. Private variables are initialised to their initial value.

Protocol: By construction, GSM executes only *applicable* events and blocks all others. Artifact instantiation events are always permitted. This is reflected in the environment protocol P_E:

$$P_E(\Sigma) = \{a | \exists_{\iota \in \Sigma} : (\chi \in X(\Gamma) \wedge \chi(\iota, a)) \vee a \in inst\}$$

where $X(\Gamma)$ is the set of all sentries in Γ and $\chi(\iota, a)$ is the evaluation of a sentry χ with respect to the action a and status attributes $Att_{status} \in \iota$. We write *inst* for the set of artifact instantiation events. The *AutoAgent* stores the set of pending tasks in its buffer t and sends them at a later point to Γ. Thus, the protocol simply selects any pending task from its buffer by using the expression $P_0(t) = \{a | a \in t\}$. The protocol of an agent i gives the set of actions that are available in visible instances of its local state and satisfy its local protocol:

$$P_i(l_i) = \{a | \exists_{\iota \in l_i} : a \in \varepsilon_i(\iota) \cap \wp_i(\iota, x_i)\}$$

These components suffice to instantiate a full AC-MAS from Definition 5. With these details in place we conclude the formal map from *A-GSM* to *AC-MAS*. In the remainder of the paper we present an implementation of a model checker for *IQ-CTLK* on *AC-MAS*.

5 Implementation

To perform AC-MAS model checking, we have implemented a model checking tool built on top of `GSMC` [10]. The checker, called `Madrid`[1] is written in C++ and uses the CUDD library [21] for the back-end symbolic computations. `Madrid` builds the model and the transition relation and performs a symbolic state space exploration based on BDDs. The GSM model and the specification of the *AutoAgent* are directly loaded from the `Barcelona` XML input file; agent definitions are given in form of a configuration file as shown in Figure 3.

To obtain finite state models, we introduce a *bound* on the number of instances that can be generated and use abstraction to create finite data. We allocate BDD variables for the states of the agents and the maximum number of artifact instances present in a run. In addition, we introduce an *Overflow* flag that indicates if the number of instances or data values were exceeded in a run. We pay special attention to this case because some of the results of the check may be unsound and require a re-check with higher bounds. We furthermore capture the *Event ID* and *Payload* of the next action a that is to be executed. A special flag *Created* in each artifact instance indicates whether it was instantiated in the corresponding run.

Rather then exploring all possible runs of the system one by one, the tool uses a symbolic representation to directly operate on sets of states. Protocols, states, temporal and epistemic relations are thus encoded as Boolean formulas using BDDs (denoted as $\overline{\wp}$, $\overline{\omega}$, ...). Differently from the current state of the art [15], here we have to consider windows to correctly encode the epistemic relations. We define the Boolean formula $\bar{l}_i(\Sigma)$ to encode the set of local states l of agent i at GSM snapshot Σ as follows:

$$\bar{l}_i(\Sigma) = \bigvee_{\iota \in \Gamma} ((\exists_{x \notin \nu_i(\iota)} : \Sigma \wedge \overline{\omega}_i(\iota)) \vee \neg \overline{\omega}_i(\iota))$$

[1] The pre-compiled binaries of the tool can be downloaded from `https://www.dropbox.com/s/tux0vl0mg3c1nk4/madrid.tar.gz`

For constructing the state space, the protocol is taken into account to ensure all transitions are the result of the execution of actions that are enabled at their respective states. We also ensure that no action operates on instances outside the agent's window. We encode the protocol for agent i as:

$$\overline{P}_i(\Sigma) = \bigvee_{\iota \in \Gamma} (\exists_{x \notin (v_i \cup \overline{a})} : \overline{\wp}_i(\iota) \wedge \bar{l}_i(\Sigma) \wedge \overline{\omega}_i(\iota))$$

where $\overline{a}$ denotes the variables used for encoding the action.

Any IQ-CTLK formula φ to be verified is first rewritten by replacing the quantification operators with formulas that range over the actual instances. However, because artifact instances are created dynamically at run-time, the number of *active* instances is not known *a priori* and needs to be considered in the formula. We use the expression $created(\iota)$ to check if an instance was created (the *Created* flag is set) and rewrite the quantified formulas as follows:

$$\forall x : \varphi \Rightarrow \bigwedge_{\iota \in \Gamma} : created(\iota) \rightarrow \varphi$$
$$\exists x : \varphi \Rightarrow \bigvee_{\iota \in \Gamma} : created(\iota) \wedge \varphi$$

Note that, for any existential formula to be valid, at least one of the artifact instances needs to be active; this is not the case in the initial state because no artifact instance has been created yet. Quantifiers can be arbitrarily nested and are resolved recursively. Once the details above are considered, `Madrid` follows existing methodologies to perform the verification of temporal-epistemic formulas [15].

5.1 Limitations

Note that the bound in the number of instances restricts the possible behaviour of the system, while data abstraction leads to an over approximation. This may lead to loss of soundness or completeness when the limits of the artifact instances are reached. The exact outcome depends on the type of the property considered. A violation of a universal property, for instance, does denote a violation on the full unbounded model even if the bound was exceeded during the computation. To the contrary, if an existential property is not satisfied, no conclusion can be drawn regarding the full model in general. These are limitations in the technique at present but, as we show in the next section, interesting scenarios can still be analysed.

6 Experimental Results

We evaluated `Madrid` on the Order-to-Cash scenario, a simplified version of the IBM back-end order management application supplied by IBM Research [13]. In this scenario a manufacturer schedules the assembly of a product based on a confirmed purchase order from a customer. Typically, a product requires several components that are sourced from different suppliers. After all components have been delivered the product is assembled and shipped to the customer.

The GSM program is specified in the form of a single-artifact `Barcelona` schema consisting of 9 stages and 11 milestones. To verify the model we performed small modifications to abstract from concrete products and created three agent roles for the above scenario: 1) a *Customer* who creates an artifact instance that represents the order and can only see instances they created; 2) a *Manufacturer* who fulfils the order and can see only uncompleted instances of orders sent to him by a customer; and 3) a

Table 1: Properties of the Order-To-Cash case study.

$$AG\ \forall x : CO((x.BId = Dio \wedge \neg x.Cancelled) \rightarrow K_{Dio}\ EF\ x.Received) \quad (1)$$

$$EF\ \exists x : CO(x.BId \neq Dio \wedge K_{Dio}\ x.Received) \quad (2)$$

$$AG\ \forall x : CO((x.BId = Dio \wedge x.Ready) \rightarrow K_{Dio} x.Parts = 3) \quad (3)$$

$$EF\ \exists x : CO(x.BId = Dio \wedge x.Cancelled \wedge \neg Dio.cancelled) \quad (4)$$

Carrier who ships the finished product to the customer, and who can see only instances of orders that are to be shipped via them.

Figure 1 gives the lifecycle of the *Preparing* stage. It is controlled solely by the manufacturer, who, upon receiving the order, launches a research process to identify suitable suppliers and orders the required components. The assembling process can begin when the first component is received and remains active until all the components are collected. This is modelled by introducing a counter; the process is considered complete when 3 components have arrived.

Table 1 gives the properties we checked for different numbers of agents and artifact instances, where *Dio* is a customer agent (*Diogenes*) and *CO* stands for the *CustomerOrder* artifact type. Property (1) represents that *Diogenes* knows that the product can always be received in all of his orders as long as they are not cancelled (i.e., that there is no deadlock in processing the order). To check that the order is private to the customer, property (2) expresses that *Diogenes* may know a product is received for an order with different owner. Property (3) encodes the ability of an agent to deduce information it can not directly observe by checking if *Diogenes* always knows there are 3 *Parts* collected in all of his orders when the milestone *Ready* is achieved. Property (4) implies that an agent other than *Diogenes* can cancel an order that belongs to *Diogenes*. This is done by using a private variable, which is set true only if *Diogenes* executed the *Cancelled* event.

We ran the tests on a 64-bit Fedora 17 Linux machine with a 2.10GHz Intel Core i7 processor and 4GB RAM and measured the number of reachable states, memory used, and CPU time required. The model checker evaluated the properties (1) and (3) to be true and the properties (2) and (4) to be false in the model. This is in line with our intuition of the model and shows that the GSM program of Order-to-Cash application is indeed correct with respect to the requirements.

Table 2 reports the performance for 3 agents (one for each role) and 15 agents respectively (6 customers, 5 manufacturers, and 4 carriers). We see that the run-time grows exponentially with the number of artifact instances, while the number of agents influences the resource usage only moderately. This is because additional agents add fewer states than additional artifact instances. The results show that the tool has the ability to effectively handle large state spaces, which is required to model realistic artifact systems with complex agent interactions.

Table 2: Reachable states, memory and time usage for different numbers of artifact instances ι and agents.

	3 agents			15 agents		
#ι	#states	MB	s	#states	MB	s
1	1.17 e2	27	0.1	2.92 e3	31	0.2
2	3.71 e3	52	0.7	4.16 e6	70	4.9
3	1.16 e5	64	5.9	5.82 e9	84	65.5
4	3.67 e6	96	42.1	8.01 e12	222	360.2
5	1.18 e8	195	176.7	1.09 e16	539	1419.6

7 Conclusions

In this paper we put forward a technique for the practical verification of GSM-based MAS. The approach consists of defining a formal map from the declarative, executable language GSM to an extension of previously studied artifact-centric MAS, a semantics for reasoning about MAS in a quantified setting of the artifact system environment. We reported on a fully-fledged model checker that implements this formal map and supports temporal-epistemic specifications in which quantification is allowed over artifact instances. The experimental results obtained against the Order-to-Cash application led us to conclude that the practical verification of reasonably sophisticated GSM-based MAS is feasible and scalable in many valuable scenarios in business processes and services.

We plan to extend the work reported here in a number of ways, including the support of limited forms of quantification over the data. Theoretical studies [11, 3] point to high-undecidability in settings where unbounded data is present. For this reason we will work on existential abstraction and data abstraction to achieve a transfer of the verification outcome from abstract to concrete models. At a later stage we wish to combine these with counter-example guided refinement procedures [6].

References

[1] M. Baldoni, C. Baroglio & V. Mascardi (2010): *Special Issue: Agents, Web Services and Ontologies: Integrated Methodologies*. Journal of Multiagent and Grid Systems 6(2), pp. 103–104. Available at http://dx.doi.org/10.3233/MGS-2010-0143.

[2] F. Belardinelli, A. Lomuscio & F. Patrizi (2011): *Verification of Deployed Artifact Systems via Data Abstraction*. In: Proceedings of International Conference on Service oriented Computing (ICSOC'11), LNCS 7084, pp. 142–156. Available at http://dx.doi.org/10.1007/978-3-642-25535-9_10.

[3] F. Belardinelli, A. Lomuscio & F. Patrizi (2012): *An Abstraction Technique for the Verification of Artifact-Centric Systems*. In: Proceedings of Principles of Knowledge Representation and Reasoning (KR'12), pp. 319–328.

[4] F. Belardinelli, A. Lomuscio & F. Patrizi (2012): *Verification of GSM-based artifact-centric systems through finite abstraction*. In: Proceedings of International Conference on Service oriented Computing (ICSOC'12), LNCS 7636, pp. 17–31.

[5] T. Bultan, J. Su & X. Fu (2006): *Analyzing Conversations of Web Services*. IEEE Internet Computing 10(1), pp. 18–25. Available at http://doi.ieeecomputersociety.org/10.1109/MIC.2006.1.

[6] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu & H. Veith (2003): *Counterexample-guided abstraction refinement for symbolic model checking*. Journal of the ACM 50(5), pp. 752–794. Available at `http://doi.acm.org/10.1145/876638.876643`.

[7] D. Cohn & R. Hull (2009): *Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes*. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 32(3), pp. 3–9.

[8] A. Deutsch, L. Sui & V. Vianu (2007): *Specification and Verification of Data-Driven Web Applications*. Journal of Computer and System Sciences 73(3), pp. 442–474.

[9] R. Fagin, J. Y. Halpern, Y. Moses & M. Y. Vardi (1995): *Reasoning About Knowledge*. The MIT Press.

[10] P. Gonzalez, A. Griesmayer & A. Lomuscio (2012): *Verifying GSM-based Business Artifacts*. In: Proceedings of the IEEE International Conference on Web Services (ICWS'12), pp. 25–32.

[11] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch & M. Montali (2012): *Verification of Relational Data-Centric Dynamic Systems with External Services*. Technical Report, CoRR Technical Report, abs/1203.0024, arXiv.org.

[12] F. T. Heath, R. Hull & R. Vaculín (2011): *Barcelona: A design and runtime environment for modeling and execution of artifact-centric business processes (demo)*. In: Proceedings of the International Conference on Business Process Management (BPM'11).

[13] R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya & R. Vaculín (2011): *Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events*. In: Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'11), pp. 51–62.

[14] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya & R. Vaculin (2011): *Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles*. In: Proceedings of the Internationsn Workshop on Web Services and Formal Methods (WS-FM'10), LNCS 6551.

[15] A. Lomuscio, H. Qu & F. Raimondi (2009): *MCMAS: A Model Checker for the Verification of Multi-Agent Systems*. In: Proceedings of Computer Aided Verification (CAV'09), LNCS 5643.

[16] A. Lomuscio, H. Qu & M. Solanki (2012): *Towards Verifying Contract Regulated Service Composition*. Journal of Autonomous Agents and Multi-Agent Systems 24(3), pp. 345–373.

[17] Object Management Group (2012): *Proposal for: Case Management Modeling and Notation (CMMN) Specification 1.0*. Document bmi/12-02-09.

[18] R. Parikh & R. Ramanujam (1985): *Distributed Processes and the Logic of Knowledge*. In: Logic of Programs, LNCS 193, pp. 256–268.

[19] M. Singh, A. S. Rao & M. Georgeff (1999): *Formal Methods in DAI: Logic-Based Representation and Reasoning*. In Gerhard Weiß, editor: Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence, MIT Press, pp. 331–376.

[20] M. P. Singh & M. N. Huhns (2005): *Service-oriented computing - semantics, processes, agents*. Wiley.

[21] F. Somenzi (2012): *CUDD: CU Decision Diagram Package - Release 2.5.0*. `http://vlsi.colorado.edu/~fabio/CUDD/`. January 2013.