on2ts - Typescript generation from OWL ontologies and SHACL

Jesse Wright[1] [0000−0002−5771−988X], Sergio J. Rodríguez Méndez[1] [0000−0001−7203−8399], Armin Haller[1] [0000−0003−3425−0780], Kerry Taylor[1] [0000−0003−2447−1088], and Pouya G. Omran[1] [0000−0002−4473−3877]

Australian National University, Canberra ACT 2601, AU
{`firstname.lastname`}`@anu.edu.au`
`https://cecs.anu.edu.au/`

Abstract. Ontologies expressed in OWL and their associated SHACL [4] constraints contain detailed metadata and assumptions on Knowledge Graphs (KGs). With this information comes the possibility of developing powerful front-end applications that interact with these KGs. These opportunities are often unrealised as Web developers are required to interpret OWL and SHACL statements in order to write front-end code that conforms with the back-end data model. This paper introduces *on2ts*, a developer tool that automatically converts OWL definitions and SHACL constraints into Typescript [1] classes and interfaces. This enables developers to import these definitions directly into their application. The authors have developed this tool with the goal of reducing development time of linked-data applications, improving type-safety of linked-data applications and providing an appropriate level of abstraction for front-end development environments. In addition, we note that *on2ts* promotes consistent types and programming patterns across libraries designed for similar OWL ontologies. This is in the best interest of producing a scalable network of Semantic Web technologies.

Keywords: Typescript · Javascript · SHACL · OWL · Ontology · Code Generation · linked-data · Developer Tool

1 Introduction

Despite rapid growth in the development of back-end technologies for the Semantic Web, front-end tools remain limited. *on2ts* aims to address several issues hindering development of *scalable* linked-data applications and libraries:

1. Developers have been slow to adopt Semantic Web technologies, and historically, the developer experience in linked-data applications has been poor.
2. Applications and libraries which are designed for the same ontology are incompatible due to different internal methods for handling data structures.
3. There are few linked-data libraries which make use of Typescript (or similar typed languages that are used by developers of large scale web applications).

4. Open world assumptions in linked-data environments force developers to handle cases that are not encountered in traditional databases.

The first issue has recently come to light in Semantic web communities focused on front-end development. The authors of *LDflex* note that: "use cases indicate that designing a linked-data developer experience—analogous to a user experience—is crucial for adoption" [11] and have begun to address this issue with APIs which abstract away complexities of RDF and SPARQL. Whilst such tooling is sufficient for small scale applications, we believe that these libraries require the support of ontology-based objects if they are to be used in scalable applications. In addition, we are not aware of any tooling that specifically aids Typescript/-Javascript developers in handling the complexities of open world assumptions arising from the RDF data model. In this paper, we discuss how *on2ts* can be used to resolve these issues.

The authors are aware of similar code generators which have been developed for *other* languages such as Protégé's code generator [10], the OWL-API [2] for Java and owl2perl [3] for Perl. Additionally there some Javascript libraries [5] that extract some low level information such as the IRIs and TermTypes from ontologies. Crucially, we note that none of these frameworks make use of SHACL to provide additional or customised validation behavior.

The Shapes Constraint Language (SHACL) [4] is a W3C recommendation developed to express conditions, as shape graphs, for validating RDF-based KGs. Thus domain-relevant structure can be enforced. Whilst similar languages such as ShEx [7] exist, the authors have chosen to use SHACL as it is a W3C recommendation and appears to have greater usage within industry. We additionally note that tools including RDFShape [8] enable the conversion from ShEx to SHACL.

Typescript is an open-source language developed by Microsoft which forms a superset of Javascript. It introduces static type definitions that "describe the shape of an object, providing better documentation, and allowing Typescript to validate that your code is working correctly"[1]. Reasons for choosing to develop *on2ts* in Typescript include; the language's increasing popularity[2] and the enhanced developer experience that Typescript provides when working in most IDEs[3]. Furthermore, static typing can help developers use patterns that accurately match definitions of OWL ontologies and adhere to the constraints of associated SHACL shapes. Javascript applications may also import libraries written in Typescript and benefit from tooltips derived from Typescript definitions.

[1] `https://www.typescriptlang.org/`

[2] A recent survey of 65,000 developers by StackOverflow indicates that $\sim 67\%$ of web developers are using Typescript in their projects [6].

[3] `https://code.visualstudio.com/docs/languages/typescript`

2 From ontologies and SHACL to Typescript

on2ts uses *LDflex* [11] to read OWL axioms and SHACL constraints. *ts-morph* [9] is then used to generate the subsequent Typescript files. In this process, the ontology file is used to generate the base structure of classes, which includes encoding *domain*, *range*, *subclass* and *type* declarations. In contrast the SHACL files are used to generate validators that are applied when instantiating or loading an instance of a given class.

on2ts makes use of exotic objects and proxies to allow Javascript/Typescript developers to interact with ontologies using familiar patterns of Object Oriented Programming. For instance the `Symbol.hasInstance` method is overwritten on the classes generated by *on2ts*. This enables the `instanceof` keyword to have the same behavior as `a/rdf:type` in a SPARQL query (including inferencing using *subclass*/*subproperty* relations). This means that if we have `const myCl = new rdfs.Class('ex:myCl')` then `myCl instanceof rdfs.Class`, `myCl instanceof rdfs.Resource` and `rdfs.Resource instanceof rdfs.Class` all return `true`. A simplified implementation of this is given as follows:

```
static [Symbol.hasInstance](obj: any): boolean {
    return obj.constructor === this // myCl instanceof Class
    || obj.rdfType === this // Class ins...of rdfs.Class|rdfs.Resource
    || obj.constructor.__proto__ === this // myCl instanceof Base
    || obj.constructor.getExtends?.() // myCl instanceof rdfs.Resource
        .some((extend: typeof Base) => extend instanceof this)}
```

similarly we have overwritten the *in* operator to check if an *instance* has a given *property* within the data model. For instance as below

```
function handlePerson(person: Person): void {
    if (ex.SupervisorOf in person) {...}
    else {...}}
```

on2ts similarly encodes and makes other information available. This includes information related to *domain*/*range* and *datatype* properties. Importantly, note that as the ontology definitions are encoded within the Typescript files, all information is self contained in the program and hence these operations require no external API calls beyond the inital loading of `ex:myCl`.

In some cases we need to add Javascript/Typescript methods to supplement the ontology/SHACL definitions. For instance, we need to add custom `validator` methods to each class of `SHACL` validators. These methods are integrated into the objects using *ts-morph* during the code generation process.

Data from SHACL -

Traditional applications that interact with SQL databases are programmed against a strict set of assumptions that are encoded to the model of the database.

The open world assumption of RDF data models forces developers to work with many cases that would traditionally not have to be considered. SHACL constraints allow developers or information architects to define a set of conditions that data must satisfy in order for applications to reasonably perform. By encoding this information into our applications data model, validation of data entering the application can be automated and programmers can work under assumptions that they are traditionally used to.

Consider a linked-data application for playing a game of chess. A reasonable assumption is that each piece in play lies in a valid position on the 8×8 board. This constraint is given as follows:

```
ex:inPlayShape a sh:nodeShape ;
  sh:property [
    sh:path ex:position ;
    sh:datatype xsd:string ;
    sh:pattern "^[A-H][1-8]\$" ;];
  sh:property [
    sh:path ex:inPlay ;
    sh:value true ;] .
```

If this SHACL constraint is applied (at code generation time) to the class `ex:inPositionPiece` it becomes safe to assume that if the code `const piece = new inPositionPiece(loaded_piece)` runs without error, then the piece will be in a valid position on the playing board.

We could also consider the case of designing an application which includes payments. The application requires that users have submitted their payment details prior to making a purchase. To achieve this, developers can design a `payingUser` shape and a `makePayment` function as follows

```
sh:payingUser a sh:nodeShape ;
    sh:or (sh:and ([
            sh:path ex:preferredMethod ;
            sh:hasValue ex:payPal ;]
        ex:payPalShape) sh:and ([
            sh:path ex:preferredMethod ;
            sh:hasValue ex:creditCard ;]
        ex:creditCardShape)
```

```
import {Person, PayingUser, creditCard, payPal} from 'on2ts/webUserOntology'
function async makePayment(user: Person, amt: number) {
    const payingUser = await schimatosLoader(new PayingUser(user))
    switch ('${await payingUser.preferredMethod}') {
        case '${card}': return card(await Person.card, amt)
        case '${payPal}': return payPal(await Person.payPal, amt)
        default: return bitCoin(await Person.bitCoin, amt)}}
```

In the *schimatosLoader*, we are using *on2ts* with *Shímatos* to catch data that does not pass validation. The `await` statement does not resolve until the user has submitted valid payment details within a popup produced by the *schimatosLoader*. Developers of the payment application can thus assume that `payingUser` will satisfy the requirements of `sh:payingUser` once it has resolved.

3 Availability

This tool is under active development, at `https://github.com/jeswr/on2ts` along with sample Typescript files generated from several ontologies. In the repository we are also building the plugin required to use *LDflex* [11] in conjunction with these files. Demonstration videos and sample use cases will be added during development. Some *on2ts* features discussed in this paper have not been implemented at the time of writing.

References

1. Bierman, G., Abadi, M., Torgersen, M.: Understanding typescript. In: Jones, R. (ed.) ECOOP 2014 – Object-Oriented Programming. pp. 257–281. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
2. Horridge, M., Bechhofer, S.: The owl api: A java api for owl ontologies. Semantic Web **2**(1), 11–21 (2011). https://doi.org/10.3233/SW-2011-0025
3. Kawas, E., Wilkinson, M.D.: OWL2Perl: creating Perl modules from OWL class definitions. Bioinformatics **26**(18), 2357–2358 (07 2010). https://doi.org/10.1093/bioinformatics/btq416, `https://doi.org/10.1093/bioinformatics/btq416`
4. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Recommendation, w3c (July 2017), `https://www.w3.org/TR/shacl`
5. ontola: ontola/ontologies: v2.0.1 (Aug 2020), `https://github.com/ontola/ontologies`
6. Overflow", S.: Stack overflow developer survey 2020 (2020), `https://insights.stackoverflow.com/survey/2020`
7. Prud'hommeaux, E., Boneva, I., Gayo, J.E.L., Kellogg, G.: Shape expressions language 2.1. Final community group report 8 october 2019, W3C Community Group (2019), `http://shex.io/shex-semantics/`
8. RDFShape: Parse and convert schema, `http://rdfshape.herokuapp.com/schemaConversions`
9. Sherret, D.: dsherret/ts-morph: v7.3.0 (Aug 2020), `https://github.com/dsherret/ts-morph`
10. University, S.: protegeproject/code-generator: v2.0.0 (Aug 2020), `https://github.com/protegeproject/code-generator`
11. Verborgh, R., Taelman, R., Herwegen, J.V., Lemée, J.B., Willems, J., Shurmer, J., de Jong, M.: Ldflex/ldflex: v2.11.1 (Jun 2020). https://doi.org/10.5281/zenodo.3894538, `https://doi.org/10.5281/zenodo.3894538`