Overcoming ReqIF Exchange Challenges through Deep File Analysis and Dedicated Tooling

PEUGEOT Thomas
Moss S.A.S.
Issy-Les-Moulineaux, France
thomas.peugeot@moss.fr

MERBOUCHE Nicolas
Moss S.A.S.
Issy-Les-Moulineaux, France
nicolas.merbouche@moss.fr

GILLET Vincent
DGA MI
Bruz, France
vincent-j.gillet@intradef.gouv.fr

Effective requirement exchange between stakeholders is critical for streamlined systems development. The Requirement Interchange Format (ReqIF) 1.2 serves as the de facto standard for interoperability between requirements management tools, enabling the transfer of documents, images, and traceability links. This paper presents a return of experience (Retex) on the use of ReqIF for data exchange between IBM DOORS and Siemens Polarion, as well as between Polarion instances. While standards promise seamless integration, our practical application revealed significant challenges that necessitated a deep-dive analysis of the ReqIF file structure to resolve.

We present an in-depth description of the ReqIF structure derived from this investigation. To support this analysis, we introduce *gongreqif*, an open-source tool developed to navigate and comprehend ReqIF file content. Based on our findings, we offer concrete recommendations for practitioners implementing ReqIF-based exchanges and for vendors seeking to improve the usability of their import/export functions. Finally, this experience with ReqIF provides valuable insights for the upcoming SysML v2 standard, highlighting the crucial gap between advertised compliance and true, practical interoperability for tool vendors.

Keywords—ReqIF, requirements, tooling, DOORS, POLARION, SysML V2

I. Introduction

This paper is a return of experience of a group of industry practitioners.

A. System of interest

We present our findings with the help of a mockup system. This mockup system is an autonomous, wheeled drone for collecting tennis balls.

The model contains 20 stakeholder requirements, 60 system requirements, 30 traceability links and 3 specifications. The system is expected to incorporate new features every year.

Our findings can apply to a system with many times this number of requirements.

B. Stakeholders

They are 4 stakeholders.

- A client responsible for the stakeholder requirement specification
- 3 private contractors: contractor A is responsible for the consistency of the system level specification and two contractors, B and C, share ownership of the system level requirements, and that are responsible for the delivery of the system.

C. Requirement engineering process

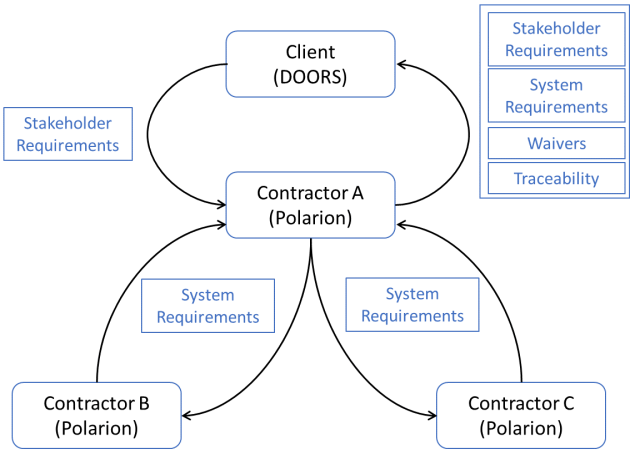

Fig. 1. Requirement engineering process

Client authors stakeholder requirements. Those requirements are handed over to contractor A.

Contractor A drafts system requirements and negotiates with all stakeholders that the system level requirements are consistent with:

- stakeholder requirements,
- user needs that were somehow ambiguous during stakeholder requirements capture,
- the ability of the contractors B and C to deliver a system to meet those requirements.

If a stakeholder requirement happens to be irrelevant at design time or impossible to meet, a waiver is authored and negotiated with stakeholders.

Contractors A, B and C are responsible for the delivery of the traceability matrix between stakeholder requirements and system requirements and waivers.

The requirement engineering process follows system engineering practices, but the tooling is more specific to this project. Stakeholder's networks are isolated and system engineering tools can only exchange files.

Stakeholders use different requirements authoring tools:

- client uses Doors
- contractors use Polarion

We use ReqIF for exchanging files ([1][2][3]). There is scarce literature on the subject, and documentation of both POLARION and DOORS lack specifics on conversion techniques from/to the ReqIF standard.

D. *Configuration management process*

For this system of interest, stakeholders have an incentive to have a smooth requirement exchange because the system is expected to routinely incorporate new features.

Therefore, it is mandatory that stakeholder tools manage *first imports* but also *updates* of ReqIF files in output as well as in input. Usually, first imports are well handled by tools. Updates are more challenging and have driven the present effort.

More specifically, if a Change Request (CR) is accepted by the Change Control Board (CCB), then the system requirement specification is updated. We want all imported specification to be automatically updated in each stakeholder tool in a matter of minutes.

II. WHAT IS A REQIF FILE ?

A. *M0, M1, M2 and M3*

The ReqIF standard is authored by the Object Modeling Group (OMG). From a modeling perspective, a ReqIF file can be seen as four layers:

- The "objects" (or MO) that describes requirements ("spec objects"), relations between requirements ("spec relations"), group of relation ("spec relation group") and documents containing both ("specification")
- The "spec types" (model/M1 or onthology) that describes the models of requirements ("spec object type"), models of requirements relations ("spec relation type"), models of group of requirement relation and models of specifications ("specification type"). Every object follows a "spec type".
- The "datatypes" (meta model/M2) that describes what kind of attributes the model can use. This includes the definition of enumeration.
- The ReqIF (meta meta model/M3) which is the standard itself and that is defined in a XSD document. For example, it states that attributes basic types are REAL, INTEGER, STRING, XHTML, DATE and ENUMERATION.

A ReqIF file is an XML file that is defined by a ReqIF XSD (made of 84 elements with 368 fields).

B. *Content of the ReqIF files of the mockup system*

The presented ReqIF file contains:

- the stakeholder requirement, the system requirements and the waivers.
- Traceability (relation between stakeholder requirements and system requirements)
- 3 documents (specification): specification with the stakeholder requirements, specification with the system requirements and a document for the traceability

The following figures are screen captures of the *gongreqif* tool. This tool shows the number of instances per element of the ReqIF file (on the top right of each box).

The gongreqif tool and the mock ReqIF file are available on github ([4]).

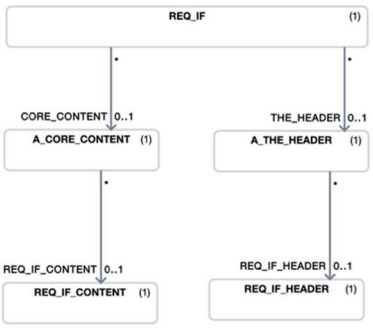

Fig. 2. ReqIF Top Level Content

At the top level of the XML file, a ReqIF file has a header and a content.

One part of the content is the datatypes definition as in the following figure. Each datatype is used in the mockup ReqIF file.

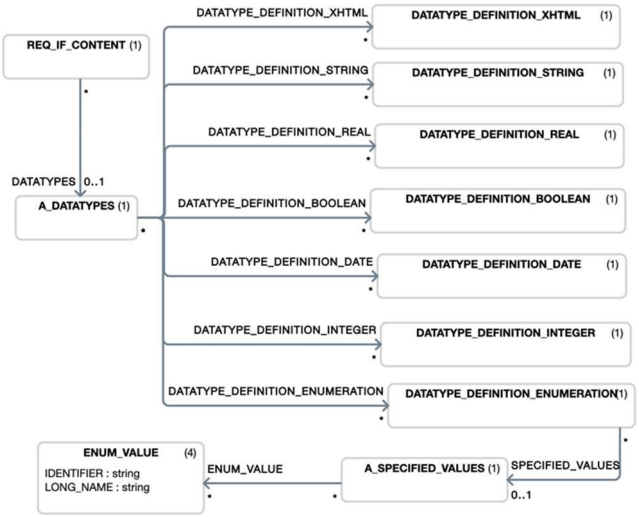

Fig. 3. The Datatype definitions in a ReqIF XML

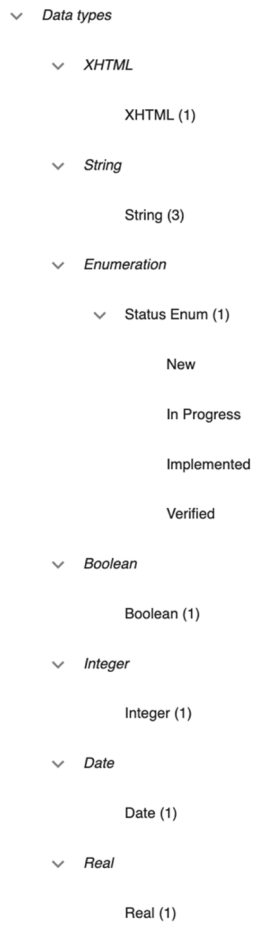

Fig. 4. Datatypes of the ReqIF file

Then, the ReqIF file defines 7 spec types.

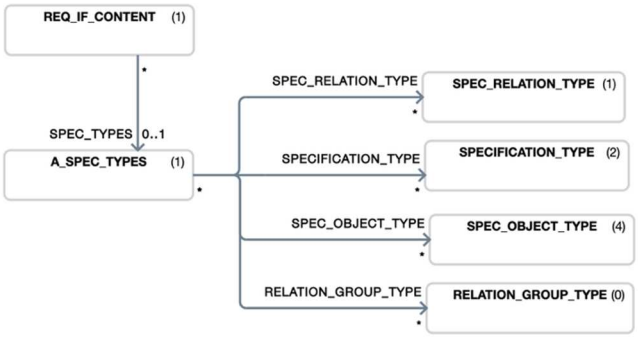

Fig. 5. The Spect Types definition

The 7 types of spec type of the mockup file are shown in the following screen capture of *gongreqif*. For each field of each spec object type, the user can choose how to render the field. For instance, the user can specify that a "Heading" ID and LONG-NAME shall be rendered as a title in the rendered document.

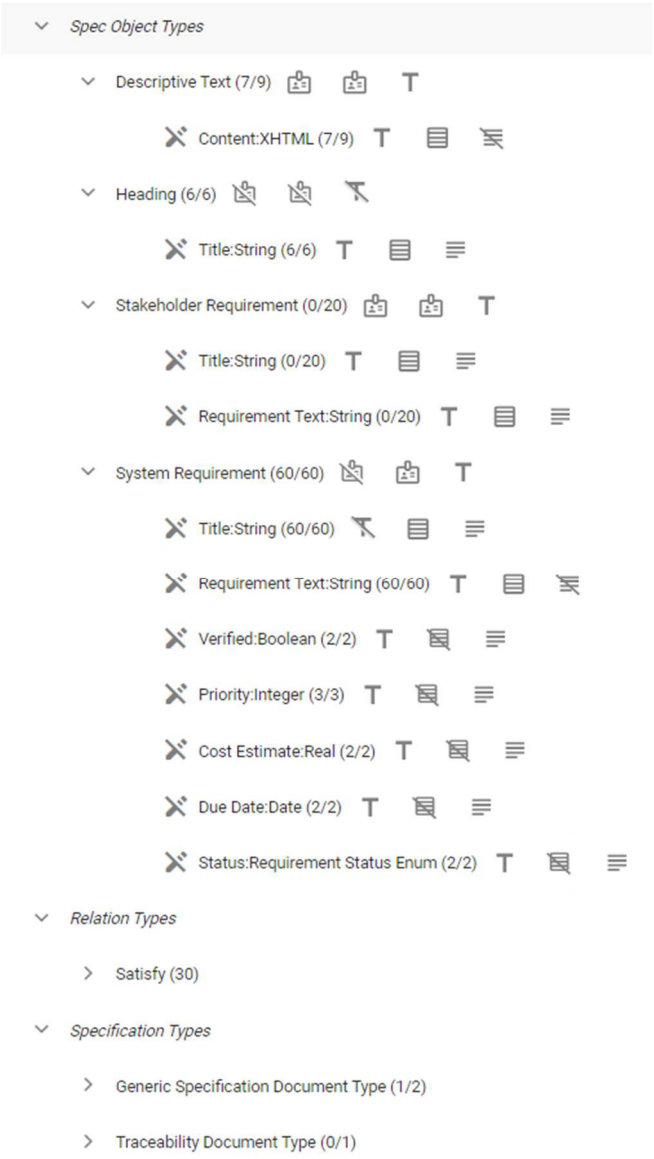

Fig. 6. The 4 object types, the relation type and the specification types

Within the object type, the stakeholder requirement type and the system requirement type are self-explanatory. The Heading and Descriptive Text are part of "specification" which is a document containing requirements (see later).

Then, the ReqIF file describes spec objects. They are defined by their types and the values of their attributes (see following figure).

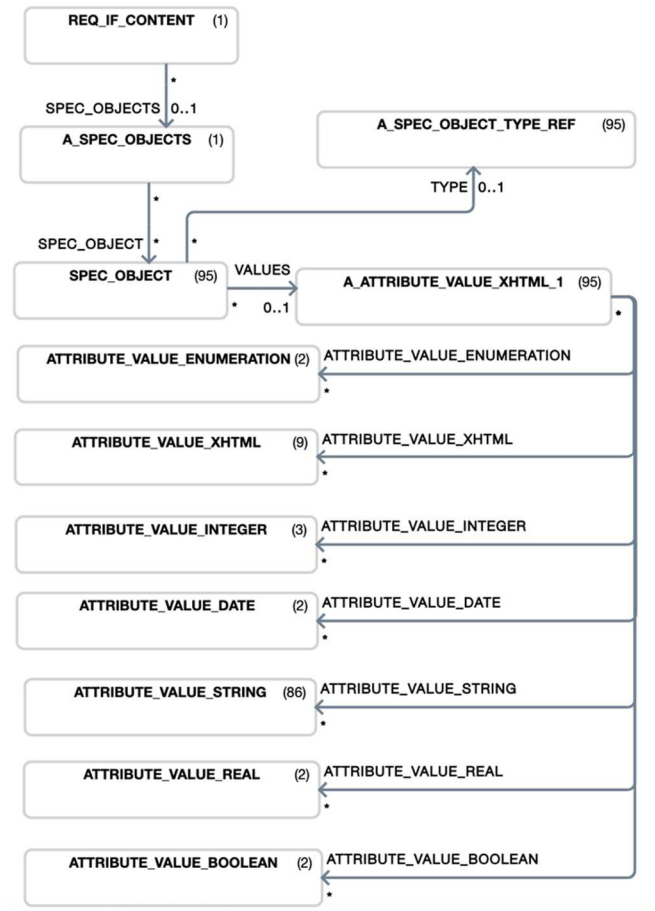

Fig. 7. Attributes of a spec object

In another part, the ReqIF file describes the specifications.

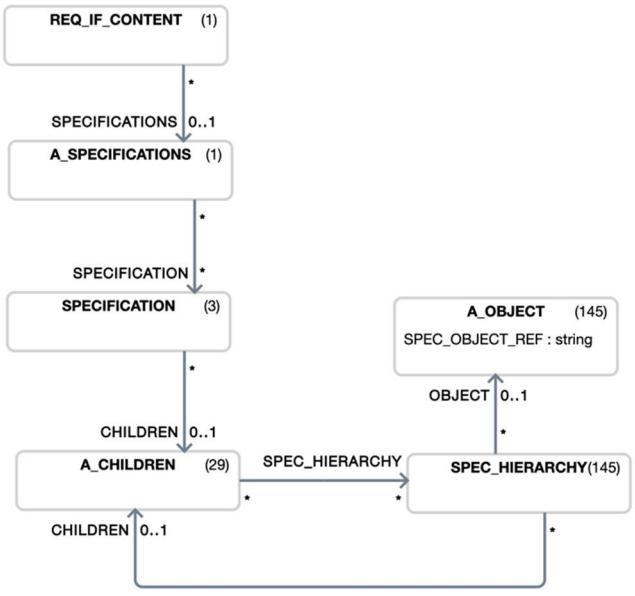

Fig. 8. Structure of a specification

A specification is a hierarchical structure with leaves being "objects". An object can be any spec object (a spec object or a relation). The following figure is the rendering of the hierarchy of the beginning of the system requirement specification.

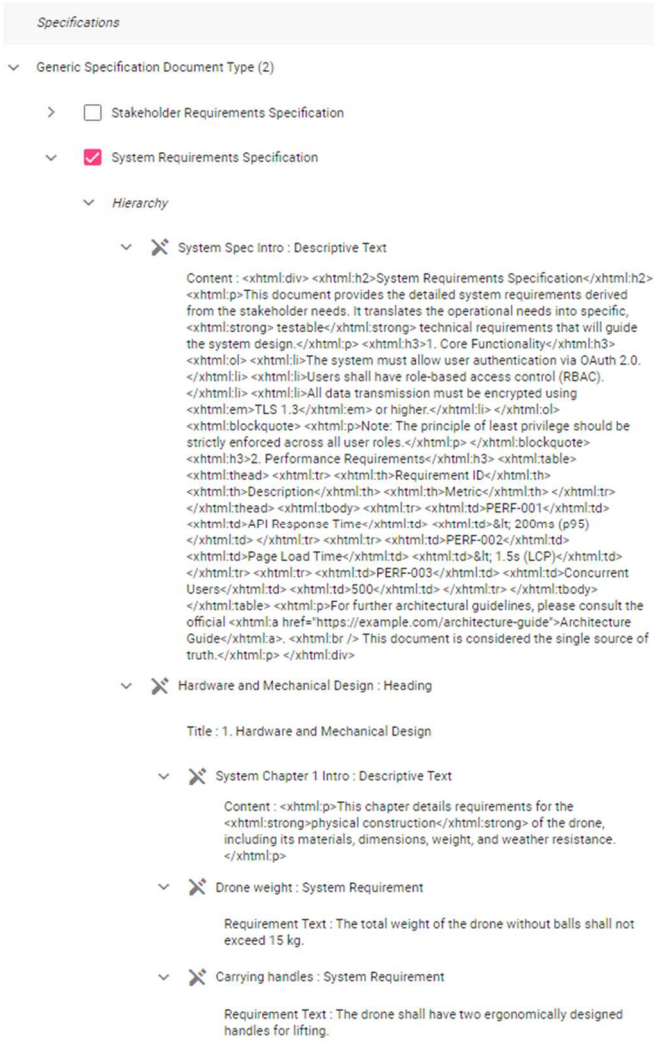

Fig. 9. Specification hierarchy

C. *Rendering of the system specification*

The following figure renders this hierarchy as a document. One can see the importance of Heading and Descriptive text spec objects to organize a specification. There are not requirements, but they structure the specification.

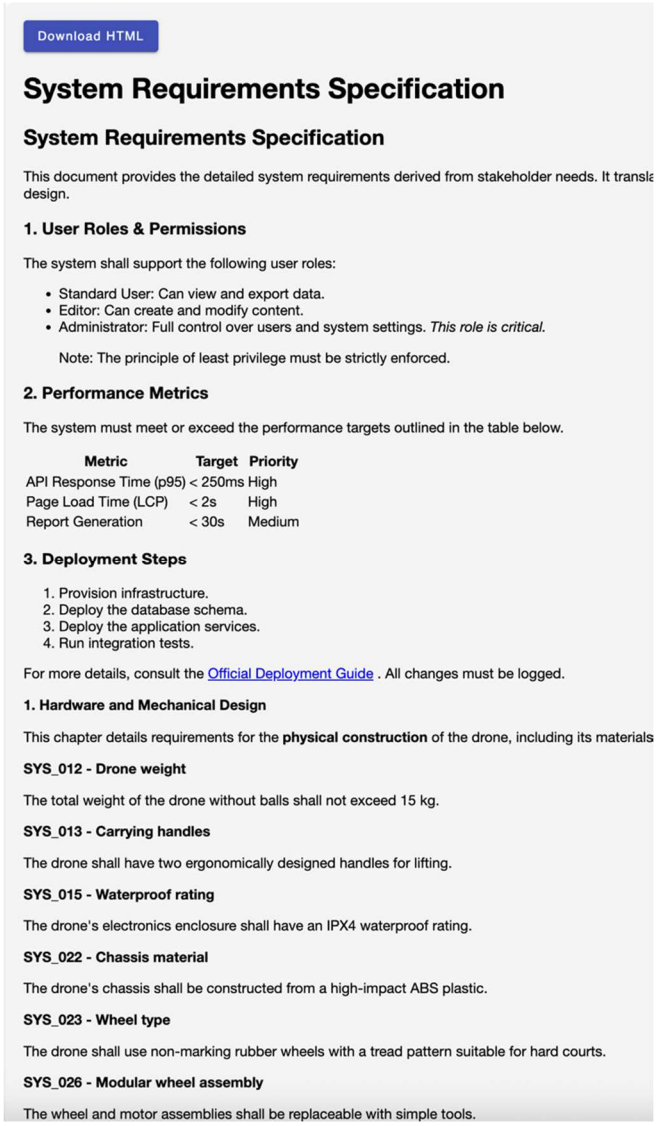

Fig. 10. Rendering of a specification

With Polarion, a paragraph can be pure text or a “Work Item”. It is our recommendation that everything is a Work Item. For instance, a “Description Text” type shall be defined and used for paragraphs.

D. *The Power of ReqIF*

The ReqIF standard is a powerful tool. The model presented above is not limited to requirements. ReqIF can be used to exchange requirements but also design documents.

III. DOORS AND POLARION EXPERIENCE

We present some challenges that were met. We experienced with DOORS/POLARION interface and POLARION/POLARION interface.

A. *ReqIF file content is key to analyse issues*

To debug behaviors between tools, you sometimes cannot know if the problem is with the source tool or the target tool. One solution is to deep dive into the ReqIF files.

Unfortunately, a ReqIF XML file is very difficult to read for a human. For this reason, it is recommended to have a tool for analyzing ReqIF files. We present an open source tool, developed for this need.

B. *ReqIF updates is the real stuff*

A tool that imports a ReqIF file sees a specification that was authored in another tool as a foreign object. The challenge for the tool is to import an update of the ReqIF file and to update the foreign object accordingly (via a matching mechanism).

POLARION version 3.10.24 can manage updates. To our knowledge, DOORS matching mechanism requires a scripting tool to be developed.

C. *With DOORS v9, export the type of each object*

When DOORS v9 exports a ReqIF, all objects are of the same SPEC-OBJECT-TYPE in the ReqIF File. By default, POLARION lack information for mapping DOORS objects to POLARION work item. For instance, what objects are paragraphs and what objects are requirements?

There is a solution, if you export the type of the object via an enumerate attribute that provide the target type, it is possible for POLARION to map correctly the object to its target type. During the import, you need to map the enumerate value to the “Type” standard attribute in POLARION. The following figure shows how a “DescriptionObject” attribute is used to specify in what kind of POLARION Work Item a DOORS object must be translated.

Fig. 11. Specification of the mapping in POLARION of objects exported by DOORS

D. *To exchange images, use the reqifz format*

A ReqIF can reference images within xhtml typed elements. If the ReqIF file is bundled with the referenced images files within a ZIP file (with the extension “reqifz”), POLARION can import the images correctly.

E. *An ICD shall be created to document how tools translate to and from ReqIF files*

Exchanging requirements between tools is a system engineering effort where the system is composed of requirement management tools (4 in our cases). As in all interfacing effort, a document shall control how the interface works. This Interface Control Document (ICD) shall describe:

- The model of requirements and traceability and their attributes within each tool
- The model of requirements and traceability in the exchanged ReqIF files
- The mapping in export to ReqIF and import from ReqIF (first import and update) of the tools model

A tool like gongreqif allows stakeholders to verify that their tool behavior meet the ICD.

IV. WHAT IS EXPECTED FROM TOOL VENDORS

The ReqIF standard is powerful but to benefit from the expressiveness, tools vendors should document:

- How are datatypes exported/imported?

- How can the datatype mapping be configured?
- How are spec types (spec objects, spec relation, specifications) exported/imported?
- How can the spec types mapping be configured?
- How are updates managed? For instance, if a specification in a source tool updates a datatype, is this update considered in the target tool?
- How are specifications exported/imported?

More generally, a tool user shall be able to precisely manage how the tool content is translated in a ReqIF file and how the ReqIF file is translated into the tool content. This tool configuration shall cover exports, first import and updates.

V. The Gonreqif tool

Gongreqif is a command-line tool that was developed for analyzing and inspecting ReqIF files. The user is expected to browse or drop his ".reqif" or ".reqifz" files to get a structured view of the requirements data.

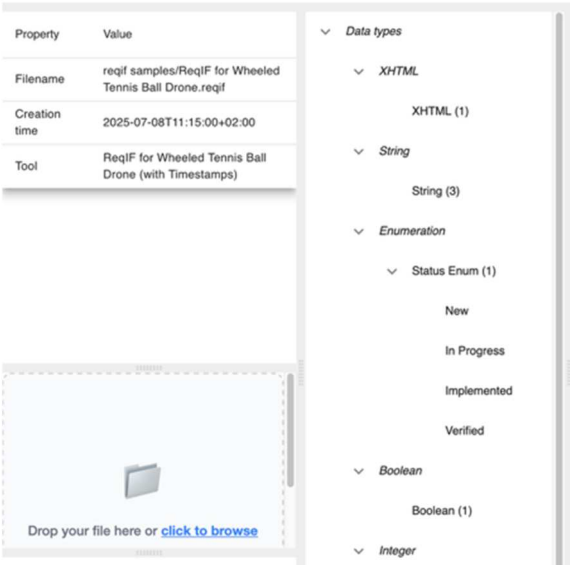

Fig. 12. Gongreqif welcome interface

The tool is developed in go ([5][6]).

Gongreqif was developed with less than 2500 lines of go code.

For ReqIF, the 913 lines of the ReqIF xsd specification file were automatically translated into 1459 lines of go code to describe the ReqIF ontology with the help of gongxsd ([9]).

The tool was then developed with the gong framework. Gong is dedicated to the development of ontology-based system engineering tools with minimal effort ([7][8][10]).

VI. Conclusion

System engineering tools interoperability is critical on projects with a lot of stakeholders. This paper reports lessons learned to interoperate DOORS with POLARION and POLARION with POLARION. It is hoped that SysML v2 ([11]) will lessen such interoperability efforts.

VII. Acknowledgements

The authors want to thanks the support of Antoine Le-Roy of France DGA and Philippe Marjanovic and Guillaume Deparois of Thales.

[1] C. Ebert and M. Jastram, "ReqIF: Seamless Requirements Interchange Format between Business Partners," in *IEEE Software*, vol. 29, no. 5, pp. 82-87, Sept.-Oct. 2012

[2] prostep ivip. (2023). *Recommendation for the use of ReqIF 1.2 in the automotive industry* (Version 2.1). Retrieved from https://www.ps-ent-2023.de/fileadmin/prod-download/Recommendation_ReqIF_V2.1_final.pdf

[3] Object Management Group. Requirements Interchange Format (ReqIF), version 1.2. http://www.omg.org/spec/ReqIF/1.2

[4] gongreqif, ReqIF File Analyzer, https://github.com/fullstack-lang/gongreqif

[5] Russ Cox, Robert Griesemer, Rob Pike, Ian Lance Taylor, and Ken Thompson. 2022. The Go programming language and environment. Commun. *ACM* 65, 5 (May 2022), 70–78

[6] Build simple, secure, scalable systems with Go, https://go.dev/

[7] D. Ernadote, "An ontology mindset for system engineering," *2015 IEEE International Symposium on Systems Engineering (ISSE)*

[8] T. Peugeot, "GONG: an open source Ontology Based System Engineering toolset," *2022 IEEE International Symposium on Systems Engineering (ISSE)*, Vienna, Austria, 2022, pp. 1-4

[9] gongxsd, https://github.com/fullstack-lang/gongxsd

[10] gong, https://github.com/fullstack-lang/gong

[11] Bajaj, M., Friedenthal, S. and Seidewitz, E. (2022), Systems Modeling Language (SysML v2) Support for Digital Engineering. INSIGHT, 25: 1