## Introduction

How to validate data against expectations? Major options are visual inspection, programatic checking and validation against a schema document (e.g. XSD, RelaxNG, Schematron, JSON Schema) or a schema graph (e.g. SHACL). Schema validation is in many scenarios the superior approach, as it is automated and declarative. But there are also limitations worth considering, when thinking about validation in general.

First, schema languages describe instances of a particular format or mediatype only (e.g. XML, JSON, RDF), whereas typical projects involve a mixture of mediatypes. Therefore schema validation tends to describe the state of something which is a piece from a jigsaw puzzle, and the question arises how to integrate the results into a coherent whole.

Second, several schema languages of key importance are grammar based and therefore do not support “incremental validation” – starting with a minimum of constraints, and adding more along the way. We cannot use XSD, RelaxNG or JSON Schema in order to express some very specific key expectation, without saying many things about the document as a whole, which may be a task causing disproportional effort, or which is pointless (because of frequent changes) or even impossible (because of great variability). Rule based schema languages (like Schematron) do support incremental validation, but they are inappropriate for comprehensive validation as accomplished by grammar based languages.

As a consequence, schema validation enables unrelated acts of resource validation, but it cannot accomplish the integration of validation results. Put differently, schema validation may contribute to, but cannot accomplish, system validation. The situation might change in an interesting way if we had a schema language for validating *file system contents* – arbitrary trees of files and folders. This simple abstraction suffices to accommodate any software project, and it can accommodate system representations of very large complexity.

This document describes an early version of **greenfox**, a schema language for validating file system contents. By implication, it can also be viewed as a schema language for the validation of *systems*. Such a claim presupposes that a meaningful reflection of system properties, state and behaviour can be represented by a collection of *data* (log data, measurement results, test results, configurations, …) distributed over a set of files arranged in a tree of folders. It should then often be possible to translate meaningful definitions of system validity into constraints on file system contents. At other times it may not be possible, for example if the assessment of validity requires a tracking of realtime data.

The very notion of system validation implies that extensibility must be a key feature of the language. The language must not only offer a scope of expressiveness which is immediately useful in many scenarios. It must at the same time serve as a rigorous *framework*, within which current capabilities, future extensions and third-party extensions appear as uniform components of a coherent whole. The approach we took is a generalization of the key concepts underlying SHACL, the validation language for RDF data. These concepts are the (shape, constraint, target, focus, etc.) appear to us as the building blocks of a simple metamodel of validation, which offers clear guidance for extension work.

Validation relies on the key operations of navigation and comparison. File system validation must accomplish them in the face of divers mediatypes and the necessity to combine navigation within as well as between resources. In response to this challenge, greenfox is based on a *unified data model* (XDM) and a *unified navigation model* (foxpath/XPath/XQuery) built upon it.

Validation produces results, and the more complex the system, the more important it may become to produce results in a form which preserves as much information about the issues as possible, and to turn this information into resources which can be integrated with other information resources in an unforseeable variety of ways. This goal is best served by a *vocabulary* for expressing schema contents and validation results in a way which does not require any context for being understood. We choose an RDF based definition of validation schema and validation results, combined with a bidirectional mapping between RDF and more intuitive representations, XML and JSON, assumed to be used in most practical work.

Before providing a more detailed overview of the greenfox language, a simple example should give a first impression of how the language can be used.