Types and Operations for FDOs

A Conceptual Study for a Type Theoretical Basis of FDOs

Ulrich Schwardmann

1. Introduction and Overview

This presentation tries to cover the problem of bootstrapping type and attribute definitions for FDOs by providing a set of primitive types and type constructors that can be used to cover all possible types needed. It turns out that this can be achieved by connecting types closely to operations as it is the case in the logical-mathematical type theory, as for instance in Per Martin-Löf Type Theory [MaL1973] and its extension to Homotopy Theory as in [HoTT2013].

[KW2006] emphasizes reliable references in the ecosystem of digital objects, but also stresses that they are always typed and they need operations for access and management. The types are maintained in type registries, where new types can be defined from a limited set of basic types via two type construction operators *set-of* and *compose*. Types and operations are described as an intrinsic part of digital objects.

However neither the technical setting of the registry services nor the origin of the basic types, nor the logical construction of new types, nor the relation between operations and types have been made explicit there.

A first step for more explication for types was made by describing schemata for type definitions in the RDA-WG on Type Registries [LBM2015]. The second step in the direction of the logical background was made in [Scw2016] by the recursive construction process of new type descriptions from other type descriptions and the description of basic types Here also the automated schema extraction from these descriptions was shown and the distinction between the schema and their validation by an additional operation was explained.

The FDO TSIG WG described in [TSIG2024] more explicitly the construction of FDOs: that in the FDO record is always expected and that data FDOs need to provide a type of that data in that record. The attributes used in such FDO records are key value pairs, where the key is a reference to an attribute definition, again an FDO, which determines the value space of the attribute. Operations are only vaguely described as service FDOs that need FDOs of specified type as input providing FDOs of specified type as output. In [Scw2025] the FDO profile becomes the role of the type for the FDO record.

In all these considerations certain constructions allowed in markup languages like JSON and XML are used as the basis for type and attribute descriptions. While these constructions are mainly used to provide additional information about the objects as metadata, the deeperconnection between FDOs, types and operations remains still out of the focus.

Simple restrictions on basic types like regular expressions, intervals of numbers or *multipleOf* are provided by the schema language of these markup languages, and there is the vague idea that these type constructors must provided by some kind of operation; e.g. *multipleOf* obviously by the multiplication with a certain number for instance. But since these limited constructors are seen as sufficient for the needs in focus, their logical background is often not further challenged.

For the determination of the basic FDO type and attribute constructors such an approach is not sufficient, because even after some ad hoc extensions the whole wealth of all operations cannot be covered this way, not even the number theoretical ones, and there will be always types that remain undefineable. In the logical-mathematical type theory however operations in general are used as type constructors and a minimal set of such operation based type constructors is an important building block in the effort to provide an axiomatical fundament for mathematics at large [HoTT2013].

This paper describes that this minimal set of type constructors from type theory is also a basis for everything what is already used in the type and attribute definitions for existing FDOs. It advocates here to also use these type theoretical constructors for the bootstrapping of FDO types. A pre-definition of the

JSON schema language elements on the basis of these minimal constructors can build a bridge into the wealth of already existing schemata for metadata descriptions for digital objects.

2. Metadata Statements in a New Light

In traditional set theoretical notation $\{x \in \mathbf{R} \mid x > -273.15\}$ for elements of type $\mathbf{R}$ of reals describes the type for the value of temperature. $>: \mathbf{R} \times \mathbf{R} \to \mathbf{Bool}$ is an operation and $x>-273.15$ in infix notation is in the image of a function $ir_{-273.15}: \mathbf{R} \to \mathbf{R} \times \mathbf{R}$ with $x \mapsto (x,-273.15)$. Numbers greater then -273.15 have the type of the preimage $(>\circ ir_{-273.15})^{-1}(\mathbf{True})$ of $\mathbf{True}$. So preimages of operations between types are type constructors.

Similarly the type of strings that fulfill a regular expression is a preimaoge of the validation of the regular expression. And in the case of *multipleOf* the type constructor is the image of the multiplication opertion. Overall operation construct three kinds of types: one as the type of operations between source and target types, one by the image of the source and one by the preimage of parts of the target.

But what is the simple JSON example `{"temperature": 18.6}` logically? It consists of a pair of a special string as first component and as second component a member of a type dependent on the choice of the first entry, expressed as: $\textit{"temperature"} \mapsto (>\circ ir_{-273.15})^{-1}(\ \mathbf{True}\ )$. Generally a JSON dictionary type of $k+1$ string elements can be described by maps $\{0,\ldots,k\} \to \mathbf{S}$, which are then mapped by a function F to types. Such key value pair objects build therefore a special subset of a product, where the second component of the tuples are dependent on the first component as $(x,F(x))$ for each $x:\{0,\ldots,k\} \to \mathbf{S}$. This construction is an example of the so called dependent product type.

3. The Universe, Dependent Types and Operations

For the mapping F of the selected keys to types as above it is necessary to introduce the collection of all constructed and known types, called the universe $\mathbf{U}$ of inductively constructed types to build a function $F : (\{0,\ldots,k\} \to \mathbf{S}) \to \mathbf{U}$ for each key selection.

In general such maps $F:A \to \mathbf{U}$ are also types again and called type family or family, because they address a whole family of types parametrized by elements in a domain type. The dependent product type is a type constructor written as $\Sigma_{(x:A)}\ F(x)$ or $\Sigma_A\ F$ as a sub-type of $A \times \mathbf{U}$. Another dependent type constructor for such type families has the requirement for element a in A that $f(a)$ is of type $F(a)$, written as $f(a):F(A)$, and is called dependent operations type: $\prod_{(x:A)}\ F(x)$ or $\Pi_A F$.

4. Type Theory and the Digital Object Architecture

Obviously simple type constructors are the type of tuples (a,b) with elements $a:A$ and $b:B$ called the product of types A and B and elements in the disjoint union of types A and B, called sum of types. Together with those constructors exemplified by the simple example above already all basic type theoretical constructors are described and those should be part of the bootstrapping types for FDOs: sum and product of types, the operation type with its image and preimage constructors and the dependent operation and product type.

On the other hand the persistent identification of types as FDOs can help to concretise the concept of the so called universe of types in type theory towards a platform and domain independent interoperable, reliable, reusable and extensible definition.

In [Abr2018] a type theoretical interpretation of logical operation types is given as hardware components with input sockets, expecting data of a certain type and providing data of a certian type on output sockets, plugs called here. This interpretation can be seen as a mathematical grounding for a completely type based interpretation of the Digital Object Architecture.

5. Bibliography

[Abr2018] S. Ambroszkiewicz, "Functionals and hardware", 2018, https://arxiv.org/abs/1501.03043.

[TSIG2024] C. Blanchi, M. Hellström, L. Lannom, A. Pfeil, U. Schwardmann, and P. Wittenburg, “Implementation of Attributes, Types, Profiles and Registries,” Apr. 2024. DOI : 10.5281/zenodo.7825572.

[KW2006] R. Kahn, R. Wilensky, "A framework for distributed digital object services", 2006, Springer, DOI 10.1007/s00799-005-0128-x.

[LBM2015] L. Lannom, D. Broeder, G. Manepalli, "RDA Data Type Registries Working Group Output", 2015. DOI: https://doi.org/10.15497/A5BCD108-ECC4-41BE-91A7-20112FF77458

[MaL1973] P. Martin-Löf, "An intuitionistic theory of types: predicative part", 1973, In Logic Colloqium 1973, H. E. Rose and J. C. Shepherdson (Eds.). North-Holland, Amsterdam.

[Scw2016] U. Schwardmann, "Automated schema extraction for PID information types", in 2016 IEEE International Conference on Big Data (Big Data), IEEE, 2016. DOI: 10.1109/ bigdata.2016.7840957.

[Scw2025] U. Schwardmann, "How to Design FDO Profiles and Kernel Attributes?", Open Conference Proceedings, 2025, DOI: 10.52825/ocp.v5i.1392

[HoTT2013] S. Awodey, T. Coquand, V. Voevodsky et al., "Homotopy Type Theory: Univalent Foundations of Mathematics", 2013, Princeton, NJ: Institute for Advanced Study. MR 3204653, http://homotopytypetheory.org/book/