JSONRPC: Transaction semantics

What is a transaction in a yang modeled system? The question is not as idle as it seems. The concept of a transaction as well as related concepts of Atomicity, Consistency, Isolation and Durability (ACID) has no definition or standard when dealing with yang data. In addition to this, yang data is usually not just data - it is a living entity representing device and system state. A change to one element may result in a cascade of changes elsewhere, especially when this is done *outside* ODL.

Thus we cannot apply the normal concept of ACID as applicable to databases, because these presume that only fully compliant readers/writers perform changes only via the legitimate "front door" interface. As a result, they usually assume that the transaction integrity (ACID) applies to the whole database and the database is left in a consistent state after every transaction. This is a good fit to the tabular nature of most transactional databases.

For a yang modeled transaction we cannot have the same assumptions. In fact, we should generally assume that the state of the *whole* device is inconsistent and we should require consistency only in the areas we have manipulated. Thus, we shall define "validation scope" for yang modeled transactions. The validation scope is a list of Yang Instance Identifier (or its matching path anyxml as used in the JSON RPC draft) and the datastore implementation shall perform validation of all data UNDER each element of the list. By default the list contains the paths of all elements modified in a transaction (we verify from there and down). The default semantics are identical to ODL, but are by no means the only semantics that may be implemented. A fully featured yang-modeled datastore accessible via JSON-RPC 2.0 should be able to support adding or deleting elements from the validation list.

Another concern when exporting transactions beyond ODL is Isolation Level. ODL isolation level is relatively weak. It is in fact Read Commited for most use cases. There is no facility to perform reads or other complex operations within transaction scope - they see the result only after the transaction is committed. This forces complex transactions to be serialized into a sequence of individual operations effectively dropping the isolation level to Read Uncommited for complex use cases. There is absolutely no need for an external entity to match these semantics. In fact, because of the cost of talking to an external entity it is essential that it is capable of supporting transaction scoped reads as well as complex transactions with multiple operations. All in all, tunable isolation level up to and including Serializeable combined with the yang model validation scoping are desirable for any external datastore even if there is a performance cost incurred in implement them. The rationale is that an off-ODL datastore most obvious use case is the multiple reader/write use case where ODL and an actual device both manipulate the datastore. Simplistic implementations of this use case may work for levels lower than Serializeable and default validation scope. Complex ones - not likely.