MODULE *yarnregistry*

EXTENDS *FiniteSets*, *Sequences*, *Naturals*, *TLC*

This defines the YARN registry in terms of operations on sets of records.

Every registry entry is represented as a record containing both the path and the data.

It assumes that

1. operations on this set are immediate.
2. selection operations (such as ∀ and ∃ are atomic)
3. changes are immediately visible to all other users of the registry.
4. This clearly implies that changes are visible in the sequence in which they happen.

A Zookeeper-based registry does not meet all those assumptions

1. changes may take time to propagate across the *ZK* quorum, hence changes cannot be considered immediate from the perspective of other registry clients. (assumptions (1) and (3)).

2. Selection operations may not be atomic. (assumption (2)).

Operations will still happen in the order received by the elected *ZK* master

A stricter definition would try to state that all operations are eventually true excluding other changes happening during a sequence of action. This is left as an excercise for the reader.

CONSTANTS

PathChars,	the set of valid characters in a path
Paths,	the set of all possible valid paths
Data,	the set of all possible sequences of bytes
Address,	the set of all possible address n-tuples
Addresses,	the set of all possible address instances
Endpoints,	the set of all possible *endpoints*
PersistPolicies,	the set of persistence policies
ServiceRecords,	all service records

Registries, the set of all possile registries
PutActions, all possible put actions
DeleteActions, all possible delete actions
PurgeActions, all possible purge actions
MkdirActions all possible *mkdir* actions

the registry
VARIABLE *registry*

Sequence of actions to apply to the registry
VARIABLE *actions*

Tuple of all variables.

$vars \triangleq \langle registry, actions \rangle$

Persistence policy
$PersistPolicySet \triangleq \{$
"MANUAL", persists until manually removed
"CLUSTER-RESTART", persists until the cluster is restarted
"APPLICATION", persists until the application finishes
"APPLICATION-ATTEMPT", persists until the application attempt finishes
"CONTAINER", persists until the container finishes
"EPHEMERAL" the record is ephemeral
}

Type invariants.
$TypeInvariant \triangleq$
$\land \forall p \in PersistPolicies : p \in PersistPolicySet$

An Entry is defined as a path, an ephemerality flag, and the actual data which it contains.

By including the path in an entry, we avoid having to define some function mapping Path $\rightarrow$ *entry*. Instead a registry can be defined as a set of *RegistryEntries* matching the validity critera.
$RegistryEntry \triangleq [$
The path to the entry

$path : Paths,$

A flag to indicate when the entry is ephemeral

$ephemeral : \text{BOOLEAN}\ ,$

the data in the entry

$data : Data$

]

An endpoint in a service record

$Endpoint \triangleq [$

API of the endpoint: some identifier

$api : \text{STRING},$

A list of address n-tuples

$addresses : Addresses$

]

A service record

$ServiceRecord \triangleq [$

ID – used when applying the persistence policy

$id : \text{STRING},$

the persistence policy

$persistence : PersistPolicySet,$

A description

$description : \text{STRING},$

A set of *endpoints*

$external : Endpoints,$

Endpoints intended for use internally

$internal : Endpoints$

]

Action Records

$putAction \triangleq [$

$type : \text{“put”},$

$record : ServiceRecord$

]

$deleteAction \triangleq [$

$type : \text{“delete”},$

$path : \text{STRING},$

$$
\begin{array}{l}
\quad recursive : \text{BOOLEAN} \\
]
\end{array}
$$

$$
\begin{array}{l}
purgeAction \triangleq [\\
\quad type : \text{“purge”}, \\
\quad path : \text{STRING}, \\
\quad persistence : PersistPolicySet \\
]
\end{array}
$$

$$
\begin{array}{l}
mkDirAction \triangleq [\\
\quad type : \text{“mkdir”}, \\
\quad path : \text{STRING}, \\
\quad parents : \text{BOOLEAN} \\
]
\end{array}
$$

Path operations

parent is defined for non empty sequences

$$parent(path) \triangleq SubSeq(path, 1, Len(path) - 1)$$

$$isParent(path, c) \triangleq path = parent(c)$$

Registry Access Operations

Lookup all entries in a registry with a matching path

$$lookup(Registry, path) \triangleq \forall entry \in Registry : entry.path = path$$

A path exists in the registry iff there is an entry with that path

$$exists(Registry, path) \triangleq lookup(Registry, path) \neq \{\}$$

parent entry, or an empty set if there is none

$$parentEntry(Registry, path) \triangleq lookup(Registry, parent(path))$$

$$isRootPath(path) \triangleq path = \langle\rangle$$

the root entry

$$isRootEntry(entry) \triangleq entry.path = \langle\rangle$$

ephemeral

$$isEphemeral(entry) \triangleq entry.ephemeral$$

A path p is an ancestor of another path d if they are different, and the path d starts with path p

$$isAncestorOf(path,\ d) \triangleq$$
$$\quad \land\ path \neq d$$
$$\quad \land\ \exists\, k : SubSeq(d,\ 0,\ k) = path$$

$$ancestorPathOf(path) \triangleq$$
$$\quad \forall\, a \in Paths : isAncestorOf(a,\ path)$$

the set of all children of a path in the registry

$$children(R,\ path) \triangleq \forall\, c \in R : isParent(path,\ c.path)$$

a path has children if the *children*() function does not return the empty set

$$hasChildren(R,\ path) \triangleq children(R,\ path) \neq \{\}$$

Descendant: a child of a path or a descendant of a child of a path

$$descendants(R,\ path) \triangleq \forall\, e \in R : isAncestorOf(path,\ e.path)$$

$$ancestors(R,\ path) \triangleq \forall\, e \in R : isAncestorOf(e.path,\ path)$$

The set of entries that are a path and its descendants

$$pathAndDescendants(R,\ path) \triangleq$$
$$\quad \lor\ \forall\, e \in R : isAncestorOf(path,\ e.path)$$
$$\quad \lor\ lookup(R,\ path)$$

For validity, all entries must match the following criteria

$$validRegistry(R) \triangleq$$

there can be at most one entry for a path.

$$\quad \land\ \forall\, e \in R : Cardinality(lookup(R,\ e.path)) = 1$$

There's at least one root entry

$$\quad \land\ \exists\, e \in R : isRootEntry(e)$$

no root entry may be ephemeral

$$\quad \land\ \exists\, e \in R : isRootEntry(e) \Rightarrow \neg e.ephemeral$$

an entry must be the root entry or have a parent entry

$$\quad \land\ \forall\, e \in R : isRootEntry(e) \lor exists(R,\ parent(e.path))$$

If the entry has data, it must be a service record

$$\quad \land\ \forall\, e \in R : (e.data = \langle\rangle \lor e.data \in ServiceRecords)$$

if an entry is not root, it cannot have an ephemeral parent

$$\quad \land\ \forall\, e \in R : (isRootEntry(e) \lor\ \neg isEphemeral(Head(parentEntry(R,\ e.path))))$$

Registry Manipulation

An entry can be put into the registry *iff*
– its parent is present or it is the root *entry*
– if it is marked as ephemeral, there are no child entries in the *registry*
– if it is marked as ephemeral, it is not a change to the root entry

$$
\begin{array}{l}
canPut(R,\ e) \triangleq \\
\quad \land\ \ isRootEntry(e) \lor (\forall\, p \in parentEntry(R,\ e.path) : \neg p.ephemeral) \\
\quad \land\ (isEphemeral(e) \Rightarrow \neg hasChildren(R,\ e.path)) \\
\quad \land\ (isEphemeral(e) \Rightarrow \neg isRootEntry(e))
\end{array}
$$

put adds/replaces an entry if permitted

$$
\begin{array}{l}
put(R,\ e) \triangleq \\
\quad \land\ canPut(R,\ e) \\
\quad \land\ R' = (R \setminus lookup(R,\ e.path)) \cup e
\end{array}
$$

mkdir() adds a new empty entry where there was none before, *iff*
– the parent *exists*
– it meets the requirement for being "put"

$$
\begin{array}{l}
mkdirSimple(R,\ path) \triangleq \\
\quad \text{LET}\ \ record \triangleq [path \mapsto path,\ ephemeral \mapsto \text{FALSE},\ data \mapsto \langle\rangle\ ] \\
\quad \text{IN}\ \ \ \lor\ exists(R,\ path) \\
\qquad\quad \lor\ (exists(R,\ parent(path))\ \ \land\ canPut(R,\ record) \land (R' = R \cup record))
\end{array}
$$

For all parents, the *mkdirSimple* criteria must apply. This could be defined recursively, except what is not being defined here is a set.

It declares that the *mkdirSimple* state applies to the path and all its parents in the set R'

$$
\begin{array}{l}
mkdirWithParents(R,\ path) \triangleq \\
\quad \land \forall\, p2 \in ancestors(R,\ path) : mkdirSimple(R,\ p2) \\
\quad \land\ mkdirSimple(R,\ path)
\end{array}
$$

$$
\begin{array}{l}
mkdir(R,\ path,\ recursive) \triangleq \\
\quad \text{IF}\ recursive\ \text{THEN}\ mkdirWithParents(R,\ path)\ \text{ELSE}\ \ mkdirSimple(R,\ path)
\end{array}
$$

Deletion is set difference on any existing entries

$$
\begin{array}{l}
simpleDelete(R,\ path) \triangleq \\
\quad \land\ \neg isRootPath(path) \\
\quad \land\ children(R,\ path) = \{\} \\
\quad \land\ R' = R \setminus lookup(R,\ path)
\end{array}
$$

recursive delete: neither the path or its descendants exists in the new registry

TODO: Define the special case of root delete: the path remains but nothing else

$$recursiveDelete(R,\ path) \triangleq$$
$$\quad \land \neg isRootPath(path)$$
$$\quad \land R' = R \setminus (lookup(R,\ path) \cup descendants(R,\ path))$$

Delete operation which chooses the recursiveness policy based on an argument

$$delete(R,\ path,\ recursive) \triangleq$$
$$\quad \text{IF } recursive \text{ THEN } recursiveDelete(R,\ path) \text{ ELSE } simpleDelete(R,\ path)$$

Purge ensures that all entries under a path with the matching *ID* and policy are not there afterwards

$$purge(R,\ path,\ id,\ persistence) \triangleq$$
$$\quad \land (persistence \in PersistPolicySet)$$
$$\quad \land \forall\, p2 \in pathAndDescendants(R,\ path) :$$
$$\qquad (p2.id = id \land p2.persistence = persistence) \Rightarrow recursiveDelete(R,\ p2.path)$$

Resolve() resolves the record at a path or fails

$$resolveRecord(R,\ path) \triangleq$$
$$\quad \text{LET } l \triangleq lookup(R,\ path) \text{IN}$$
$$\qquad \land Cardinality(l) = 1$$
$$\qquad \land \text{CHOOSE } e \in l : \text{TRUE}$$

The specific action of putting an entry into a record includes validating the record

$$validRecordToPut(path,\ ephemeral,\ record) \triangleq$$
all records declared ephemeral must be put
as an ephemeral node -and vice versa
$$\quad \land (ephemeral \equiv record.persistence = \text{“EPHEMERAL”})$$

The root entry must have manual persistence
$$\quad \land (isRootPath(path) \Rightarrow record.persistence = \text{“MANUAL”})$$

putting a service record involves validating it then putting it in the registry marshalled as the data in the entry

$$putRecord(R,\ path,\ ephemeral,\ record) \triangleq$$
$$\quad \land validRecordToPut(path,\ ephemeral,\ record)$$
$$\quad \land put(R,\ [path \mapsto path,\ ephemeral \mapsto ephemeral,\ data \mapsto record])$$

The action queue can only contain one of the sets of action types, and by giving each a unique name, those sets are guaranteed to be disjoint

$$
\begin{array}{l}
QueueInvariant \triangleq \\
\quad \land \forall\, a \in actions : \\
\qquad \lor (a \in PutActions \land a.type = \text{“put”}) \\
\qquad \lor (a \in DeleteActions \land a.type = \text{“delete”}) \\
\qquad \lor (a \in PurgeActions \land a.type = \text{“purge”}) \\
\qquad \lor (a \in MkdirActions \land a.type = \text{“mkdir”})
\end{array}
$$

Applying queued actions

$$
\begin{array}{l}
applyAction(R,\, a) \triangleq \\
\quad \lor (a \in PutActions \land putRecord(R,\, a.path,\, a.ephemeral,\, a.record)) \\
\quad \lor (a \in MkdirActions \land mkdir(R,\, a.path,\, a.recursive)) \\
\quad \lor (a \in DeleteActions \land delete(R,\, a.path,\, a.recursive)) \\
\quad \lor (a \in PurgeActions \land purge(R,\, a.path,\, a.id,\, a.persistence))
\end{array}
$$

Apply the first action in a list and then update the actions

$$
\begin{array}{l}
applyFirstAction(R,\, a) \triangleq \\
\quad \land actions \neq \langle\rangle \\
\quad \land applyAction(R,\, Head(a)) \\
\quad \land actions' = Tail(a)
\end{array}
$$

$$
Next \triangleq applyFirstAction(registry,\, actions)
$$

All submitted actions must eventually be applied.

$$
Liveness \triangleq \Diamond(actions = \langle\rangle)
$$

The initial state of a registry has the root entry.

$$
\begin{array}{l}
InitialRegistry \triangleq registry = \{ \\
\quad [path \mapsto \langle\rangle,\, ephemeral \mapsto \text{FALSE},\, data \mapsto \langle\rangle] \\
\}
\end{array}
$$

The valid state of the “registry” variable is defined as Via the *validRegistry* predicate

$$
ValidRegistryState \triangleq validRegistry(registry)
$$

The initial state of the system

$$
\begin{array}{l}
InitialState \triangleq \\
\quad \land InitialRegistry \\
\quad \land ValidRegistryState \\
\quad \land actions = \langle\rangle
\end{array}
$$

The registry has an initial state, the series of state changes driven by the actions, and the requirement that it does act on those actions.

$$RegistrySpec \triangleq \\ \quad \land InitialState \\ \quad \land \Box[Next]_{vars} \\ \quad \land Liveness$$

Theorem: For all operations from that initial state, the registry state is still valid

THEOREM $InitialState \Rightarrow \Box ValidRegistryState$

Theorem: for all operations from that initial state, the type invariants hold

THEOREM $InitialState \Rightarrow \Box TypeInvariant$

THEOREM $InitialState \Rightarrow \Box QueueInvariant$