UniqueID and IDSpace

TECHNISCHE UNIVERSITÄT DARMSTADT

A short guide on IDs in PeerfactSim.KOM and their usage within overlays

Björn Richerzhagen

Contents

1 General Concept of IDs in PeerfactSim.KOM

For most overlays and applications an ID is needed to distinguish and address hosts or files. In previous versions of PeerfactSim.KOM these IDs where generated differently by each overlay. The new UniqueID and IDSpace-Concept unifies the creation and usage of IDs throughout PeerfactSim.KOM and therefor eases development of new overlays, applications and services. An UniqueID is always part of an IDSpace, which is also responsible for the creation of new UniqueIDs out of numerical values or strings. UniqueIDs can be compared to each other even if they do not belong to the same IDSpace (ie. an UniqueID used as an overlay ID can be compared to an ID used as a key for files) according to the rules outlined in the following chapters. Furthermore, IDSpaces offer a range of utility functions allowing sorting or comparison of IDs.

The main objective of the new structure is to remove logic from the ID itself and place it inside the IDSpace to allow handling of IDs in a unified way. This makes the UniqueID the new interface for all IDs in PeerfactSim.KOM. In the following chapters the API and usage of both the IDSpace and the UniqueID is explained.

2 The API

Both interfaces are located in the package `api.idspace` within the PeerfactSim.KOM source tree.

2.1 UniqueID

The API for a UniqueID is very simple, because all comparisons and calculations are implemented in the corresponding IDSpace. The following functions are provided:

`IDSpace getIDSpace();` returns the corresponding IDSpace, ie. the IDSpace that created this ID. This method is used to access all calculation, comparison and utility functions provided by the IDSpace.

`BigInteger getBigInteger();` returns the BigInteger representation of the ID, may be used by the overlay to compare IDs or calculate the range of responsibility of a node.

`BigDecimal getBigDecimal();` returns the BigDecimal represenation of the ID, to be used in the same way as `getBigInteger()`.

Basically, the UniqueID is just a container for the numerical represenation of the ID and a pointer to the IDSpace to allow complex calculations and comparisons without the need to implement them within the ID. Please note, that an ID implementing UniqueID is *not* automatically unique within the szenario. Creation of new IDs is done in the IDSpace as described in the next section and the IDSpace may provide different methods to create *unique* IDs based on hashing or incrementation of a counter.

Important: You do *not* need to implement an UniqueID, as an implementation is provided by `DefaultID` in `impl.overlay`. If you need to add functionality to an ID, the IDSpace has to be extended, *not* the ID itself!

2.2 IDSpace

The basic API of an IDSpace provides some often needed comparisons and calculations. These functions are implemented in the `DefaultIDSpace` within `impl.overlay`. In most cases you do not need to implement your own IDSpace, but if you do, always extend this class.

`int getNumberOfBits();` the number of bits used to represent an ID in this IDSpace.

`int getNumberOfBytes();` number of bytes, ie. a 12 bit ID will result in two bytes.

`BigInteger getNumberOfDistinctIDs();` maximum number of IDs, ie. $2^{\text{numberOfBits}}$

`UniqueID getEmptyID();` an object representing an empty ID, this should be used instead of `null` as it behaves consistently with `equals` and `compareTo`.

`UniqueID createID(...);` different methods to create an ID based on numerical values, strings or a NetID. Depeding on the IDSpace many more implementations are provided, for example ID creation based on a position.

`UniqueID createIDUsingSHA1();` a helper to create an ID based on a string using SHA1-hashing.

`UniqueID createRandomID();` creates a random ID.

`BigInteger getMinDistance(UniqueID a, UniqueID b);` computes the minimal numerical distance between two IDs by considering the distance in clockwise- and counter-clockwise direction.

`BigInteger getClockwiseDistance(UniqueID a, UniqueID b);` computes the clockwise distance.

`BigInteger getCounterClockwiseDistance(UniqueID a, UniqueID b);` computes the counter-clockwise distance.

`int getDigit(UniqueID id, int i, int b);` returns the ith digit of the ID interpreted in base 2^b, ie. $b = 1$ will return the bit at position i, starting with the least significant bit for $i = 0$.

`int indexOfMSDD(UniqueID id, UniqueID otherID, int b);` returns the index of the most significant different digit if the ID is interpreted in base 2^b.

3 How to...

3.1 Create an IDSpace for an overlay

Most overlays need at least one IDSpace for the IDs of the participating nodes. In a DHT the same IDSpace can be used for the keys of objects stored within the overlay. For non-DHT overlays more than one IDSpace may be needed, for example to assign an ID to a message to prevent duplicates when flooding the message through the network. All IDSpaces should be created and configured within the Node-Factory and then be passed to each node upon component creation.

The example below shows the creation of an IDSpace with an ID length of 160 bit. The ID for the node is created by this IDSpace based on the IP-address of the host. The ID is stored within the node (this is implemented in `AbstractOverlayNode`) and can be accessed by `node.getOverlayID()`. The IDSpace is referenced by the ID as described in the API-section and can be accessed with `node.getOverlayIDSpace()`. This method is also used by applications to create new IDs without having to distinguish between different overlays.

```
public class MyNodeFactory implements ComponentFactory {

	private IDSpace idSpace = new DefaultIDSpace(160);

	public Component createComponent(Host host) {
		MyNode node = new MyNode(idSpace.createID(host.getNetLayer().getNetID()));
		return node;
	}
}
```

3.2 Create a new ID within the overlay

In the previous section an ID was created by hashing the IP-Address of the host. How an ID is created is determined by the `createID(...)` method of the IDSpace. The following listings provide some examples for often used scenarios.

Create an ID based on a BigInteger

```
UniqueID id = idSpace.createID(new BigInteger("1337"));
```

This method ensures that the created ID is within the range of valid IDs as determined by the bitlength of the IDSpace.

Create an ID based on a String

```
UniqueID id = idSpace.createIDUsingSHA1("String to hash");
```

Create an ID based on a NetID

```
UniqueID id = idSpace.createID(netID);
```

This is basically a shortcut for the commonly used task to create an ID based on the NetID, as it calls `netID.toString()` and hashes the value using `idSpace.createIDUsingSHA1()`.

Create a random ID

```
UniqueID id = idSpace.createRandomID();
```

This is most often used to assign an ID to a message to ensure that it is not forwarded multiple times.

Important: Please note, that the ID is always typed as UniqueID, as overlays do *not* implement own IDs to add functionality but instead rely on the IDSpace.

3.3 Compare IDs out of the same or different IDSpaces

To compare different IDs you should use `equals()` and `compareTo()`. It is possible to compare IDs out of different IDSpaces.

Equality of IDs

Two IDs are equal, if their IDSpaces are equal and they have the same numerical value according to their representation as BigDecimal. Two IDSpaces are equal, if they have the same size, ie. the IDs are represented with the same amount of bits. This is implemented in the `equals`-method of `DefaultID` and `DefaultIDSpace`.

Comparison of IDs

An ID is smaller than another ID, if the numerical value of the ID is smaller. IDs can only be compared if their IDSpaces are equal as described above. Otherwise, an exception is thrown. This is implemented in the `compareTo`-method of `DefaultID`.

3.4 use the IDSpace of an overlay in an application

An application can access the IDSpace via the `OverlayNodes` `getOverlayIDSpace()` and the `DHTNodes` `getOverlayKeySpace()` method. The IDSpaces can then be used to create IDs for new entries or to address a node in the overlay. Keep in mind, that wherever you need to use an ID you should type the ID as `UniqueID` and introduce an IDSpace rather than implementing your own IDs. In most cases you will not even have to implement your own IDSpace as the `DefaultIDSpace` provides all relevant functionality.