Repository

The objective is to get a working implementation of the repository in two months. To make this possible we will

1. Not worry about a middleware layer. That will be done later. Note therefore that the funcitonality of the repository will be testable without IPC.
2. Not worry about move semantics. This will be done later
3. Add support for bags, maps and sets later.

The repository should support new types without the need for installation of dlls. This is achieved by allowing object schema to be sent down the wire to the repository. This allows the repository to dynamically support new types. Advantages

- No need to install dlls on the repository
- The repository has a very small memory footprint
- The repostory only stores data, never executes object behaviours. Therefore it is immune frome many forms of attack. This is good because we must not lose our data!
- The reopsitorty is closed for change. It will be reasonable to complete it by next early year, complete with extensive unit testing. It is likely that minimal further development will be needed, apart from introducing platform independence. Don the trck some more efficient implementations may by found. However, new functionality should not be required!

Type System

The repository requires a type system. A struct is identified by a GUID. A struct has members. W support additions of members over time. Each member is identified with a GUID?

Consider for now we only have leaf types, vectors and stucts. We could have the follwing type

```
struct S1
{
   vector <int > m1
};

struct S2
{
   int m2;
   S1 m3;
};

struct S3
{
   vector< vector < S2 > > m4;
};
```

An operation may want to insert an element into an m1. This requires a "path" through the structs and vectors. We assumes that an operation is always expressed in the context of a known schema, therefore a simple 32 bit int can be used to identify a member of a struct. Therefore the path can be regarded as a list in integers. Some integers index into the members of a struct, and some index into a vector. Note that structs can contain structs, and vectors can contain vectors.

Under IT it is necessary to transform the vector index positions along the path.

The type system can be represented efficiently using the byte code idea. It seems that in practise we only need to recurse downwards through the types from a given struct. The byte code representation is ideal for this purpose.

Tasks

- Develop the type system for the repository
- Allow for operations to be represented that make use of the type system
- Define IT/ET/Merge on operations

The most important initial task is to work out how to represent the type system and the operations. We are interested in compact, efficient representations. Then we need to define IT/ET and Merge. We should be able to heavily unit test these with the existing test code. Finally we can write the entire repository, and unit test it.

Lastly, we need

- Synchronisation of repositories
- Read access to detailed information in the repository. Nothing at all should be hidden! This includes types, all checkins etc.
- Possibly we need the repository to store timestamps of events.
- The middleware protocol
- Package up repository as a service
- Sets, Bags, Maps
- Move semantics
- Write a client that allows for browing or even editing of the repository
- Think about security.