Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 08:10:08 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Sun, 30 July 2006 19:49Is smart (shared) pointers of BOOST not enough to solve "value transfer semantics"? What advantages have Ptr / Pte? over them?

The difference and the main problem from my view is that

- boost smart shared pointers are "shared"
- boost smart shared pointers are "pointers"

It complicates the design because the entity represents both the pointer AND object and can be owned by more than single entity (something belongs everywhere . You have e.g. to watch carefuly for cyclical references, you have to alter your algorithms so that they work on pointers rather than on object itself.

Often, you never know when instance really is destructed due to "shared" nature of pointer.

Note that Ptr is quite different beast - it is just pointer, does not have any influence of pointee lifetime.

Anyway, Ptr to some degree represents "inverse" problem that U++ has.

In classic GC and also with boost-like smart pointers to some degree, you always know that as long as something points to object, the object itself exists. Means there are no dangling pointers possible. Con in such arrangement is that destructors are not really possible, which leaves you with manual non-memory resources management.

As U++ has taken opposite direction in resource management, using deterministic destructor cleanup, it has potential for dangling pointers. While in most situations this is really not a big trouble (as usually lifetime of pointer does no extent beyond lifetime of pointee), there are a couple of situation, esp. in CtrlCore, where situation is too fuzzy. Therefore Ptr/Pte - pointers that go NULL if pointee dies.

Mirek