Adding a Policy-Based Smart Pointer Framework to the Standard Library

David B. Held

WG21/N1739=J16/04-0179

Introduction

- Users demand smart pointer flexibility
- A default smart pointer type is a Good Thing™
- A single smart pointer type is insufficient
- Conclusion: add a PBSP framework to the Standard Library

Ownership Strategies I

Shared Ownership

- External count (e.g.: `shared_ptr<>`)
- Intrusive count (e.g.: COM and CORBA® pointers)
- Reference linked
- Collected (e.g.: `managed_ptr<>`)

Ownership Strategies II

- No copy (e.g.: `scoped_ptr<>`)
- Deep copy (e.g.: `grin_ptr<>`)
- Move copy (e.g.: `move_ptr<>`)

Storage Policies

- Scalar storage (default)
- Array storage (leak-safe wrapper)
- `FILE*` wrapper?
- Win32® `HWND` wrapper?
- Mutex wrapper?

Checking Strategies

- Checking for null on dereference adds up to 40% time overhead
- Assert vs. throw
- Compile-time default-init rejection (require explicit initialization)

Observations

- Users want choice so badly they will and do hand-roll their own smart pointers
- The Standard Library will be underutilized if it only offers one point in the SP design space
- A proliferation of independent smart pointer types leads to redundancy

Proposed Solution

- Add a PBSP framework to the Standard Library
- Reduces avoidable boilerplate across types
- Simplifies and helps customization

Concerns

- Usability of any complex policy-based library is affected by template alias support (c.f.: N1489)
- A move configuration may only be practical with intrinsic move support (à la N1377)
- Proliferation of types may complicate interoperability

Impact

- `tr1::shared_ptr<> + weak_ptr<>` can be emulated
- With proper move support, `std::auto_ptr<>` can be emulated
- Remaining Boost smart pointer types, including `scoped_ptr<>` and `intrusive_ptr<>` can be emulated

Result

- Eventually, `std::auto_ptr<>`, `tr1::*_ptr<>`, etc. should be mandated as policy configurations
- `shared_ptr<>`, due to its broad use and general-purpose nature, should be the default configuration

Implementation I

- Framework should follow the `policy_ptr<>` design soon to be reviewed by Boost
- This design is directly derived from `Loki::SmartPtr<>`, which has had users since its debut in ’01
- The design has been refined through experience and community criticism

Implementation II

- Modern compilers can handle the complexity of PB-designs
- Framework will benefit from acceptance of N1377 and N1489 (move semantics and template aliases)

Conclusion

- The existing set of Standard Library and TR smart pointers are necessary but not sufficient
- There are no significant obstacles to adoption of this proposal
- A PBSP framework will meet smart pointer demands now and for the future