“ToyPrototype”: Event Metadata & Load on Demand

Jacek Becla
Stanford Linear Accelerator Center
ToyPrototype

- **Main requirements**
  - Shared event data objects ("metadata")
  - Load on demand
  - As non-intrusive as possible

- **Proof of concept**
  - Developed in ~2 weeks, fully functional
  - In BaBar CVS since end of April
    - ToyBrokers, ToyAsciiImpl, ToyBdbImpl, ToyRooImpl, ToyTestTools
  - Some other features include
    - Simplified model (comparing to existing scribe system)
    - Flexible persistency (2 sample implementations)
New Components

- **Brokers**
  - Think: “lightweight version of scribes”

- **Smart Pointers**

- **Cache Manager**
  - Caches persistent and transient objects
  - Replacement for “event registry”

- **RefResolvers, ToyIdentifiers**
  - Insulate system from persistency details

- “Design.tex” describes all details
  - in ToyBrokers package
How Does it Work?

- **Metadata**
  - Exactly like standard event data objects
  - “metadata” characteristics of given objects implemented inside brokers via scoping

- **Load on demand**
  - Cache Manager + Smart Pointers
    - Access to data must occur through smart pointers
    - Smart pointers resolve persistent references and notifies Cache Manager on 1st de-reference
More on Metadata

Class ToyValidityScope

- defines: `enum Scope {INVALID, EVENT, JOB};`
- For given ToyIdentifier it can tell whether it’s ok to destroy it when we exit scope S

Class ToyScopeAssistant

- Manages mapping:
  ToyIdentifier → ToyValidityScope::Scope
  ToyDataObject* → ToyValidityScope::Scope
ValidityScopes - cont

- Validity scope must be provided during insertion into cache:
  - ToyCacheMgr::insert(ToyIdentifier&, ToyDataObject*, ToyValidityScope::Scope);
  - ToyCacheMgr::insert(ToyIdentifier& id, ToyDataObject* obj, ToyDataObject* parent);

- Find scope from “parent”

- Reset() resets objects for given scope
  - reset(ToyValidityScope::Scope scope);
Class ToyScopeAssistant {
    public:
        // insert {tId, scope} into _id2Scope
        void insert(const ToyIdentifier& tId, ToyValidityScope::Scope scope);

        // insert {obj, scope} into _do2Scope
        void insert(ToyDataObject* obj, ToyValidityScope::Scope scope);

        // convert tId into scope, insert {obj scope} into _doScope
        void insert(ToyDataObject* obj, const ToyIdentifier& tId);

        // convert tId into scope, remove {tId, scope}, and {obj, scope}
        void remove(const ToyIdentifier& tId, ToyDataObject* obj);

        ToyValidityScope::Scope scope(ToyDataObject* obj) const;
        ToyValidityScope::Scope scope(const ToyIdentifier& tId) const;

        bool ok2Destroy(const ToyIdentifier& tId,
                         ToyValidityScope::Scope endScope) const;

    private:
        std::map<ToyIdentifier, ToyValidityScope::Scope> _id2Scope;
        std::map<ToyDataObject*, ToyValidityScope::Scope> _do2Scope;
};
More on CacheMgr

Public API:

// insert id->obj into _map, and both into scopeAssist.
void insert(ToyIdentifier& id, ToyDataObject* obj,
            ToyValidityScope::Scope scope);
void insert(ToyIdentifier& id,
            ToyDataObject* obj,
            ToyDataObject* parent);

// if ( in the map ) return dataObject;
// else { resolve, insert into map & scopeAsssistant }
ToyDataObject* retrieve(const ToyIdentifier& id);

void reset(ToyValidityScope::Scope scope);

Private members:

std::map< ToyIdentifier, ToyDataObject* > _map;
ToyScopeAssistant _sA;
Resolving DataObjects

- **RefResolvers**
  - `ToyDataObject* resolve(const ToyIdentifier&)`
  - Persistency specific

- **ToyIdentifier**
  - Uniquely identify persistent object
  - No direct dependency on persistency
  - Uses appropriate RefResolver do resolution
template< class T >
class ToyPtr {
public:
    inline ToyPtr();
    inline ToyPtr(T* t);
    inline virtual ~ToyPtr();
    inline ToyPtr& operator=(T* t);
    inline T* rawPtr() const;
    inline T& operator*();
    inline const T& operator*() const;
    inline bool valid() const;
    inline bool sameAs(const ToyIdentifier&) const;
    inline void setIdentifier(const ToyIdentifier&);
    inline ToyIdentifier getIdentifier() const;
    void setCacheMgr(ToyCacheMgr*);
private:
    inline T* internalPtr() const;
private:
    ToyIdentifier _id;
    T* _transient;
    mutable ToyCacheMgr* _cache;
};

“Non-smart” implementation. Very simple, easy to understand, maintain and debug.
More on Smart Pointers

◆ Alternative approach: Loki Smart Pointers
  – BbrSmartPtr<T, OwnerShipPolicy, ConversionPolicy, CheckPolicy, StoragePolicy>
  – Very flexible, but complex

◆ ToyPtr & Loki: comparable performance-wise

◆ Currently looking at:
  – Stripping down Loki pointers:
  – BbrSmartPtr<T, OwnerShipPolicy>, and:
    ConversionPolicy: DisallowConversion
    CheckPolicy: AssertCheck
    StoragePolicy: ToyPtr

◆ Or perhaps add ref counting to ToyPtr?
Other Items to Discuss

- Reference counting
  - Intrusive ref counting?
  - No cyclic references?

- Clustering
  - Old system: one big table
  - Proposed:
    - Class ToyDataObjectLocator {
      std::string _key;
      int _type; /* ooTypeNumber or "root classID"
      std::string _groupName;
    }
    - Locators distributed across brokers

- Gerhard’s suggestions:
  - Type checking
    - Typed keys? Get rid of ToyDataObject? (use AbsEvtObj)?
  - Separating conversion from cache
  - Obtaining cacheMgr from factory, via key lookup
Toy → Production System

1. Decide on Smart Pointers implementation
2. Fix known bugs
   - Introduce 2nd Cache Mgr (needed 1 for input & 1 for output)
   - Insure correct order of deletion of cache entries
3. Verify the prototype works with Root I/O
4. Re-structure code (re-package)
   - Also find acronym (perhaps Database Neutral Archive?)
5. Introduce new “Environment” (for long-lived objects)
6. Migrate few real BaBar classes & test
7. Full integration
   - Migrate all classes, rewrite brokers
     - amount of work known after “6)”
   - Bindings to BaBar tcl
   - Clustering issues (e.g. all-in-one clustering)?
   - Backwards compatibility
8. Tests and deployment
People

- Daniel Wang & Jacek Becla
  - ToyPrototype: core, Bdb & ASCII

- Akbar Mokhtarani
  - Loki smart pointers

- Jane Tinslay
  - RootIO

- Gerhard Raven and David Brown
  - Involved in some discussions

- Toy → production system
  - Daniel, Jacek, Jane, Akbar, Yemi
Goals for the Workshop

- Finalize detailed feature set
  - Type checking?
  - Smart pointers implementation?
- Discuss integration
- Discuss timetable