Expression Templates

or: How I Learned to Stop Worrying and Love Template Meta-Programming

Volker Krause

vkrause@kde.org
@VolkerKrause

KDAB

Template Meta-Programming

- Discovered by accident
- Template system is Turing-complete
  - Recursion
  - Conditions using template specialization

KDAB

Incomprehensible Gibberish

```
template <int N> struct F {
    static int const value =
       F<N - 1>::value + F<N - 2>::value;
};
template <> struct F<1> {
  static int const value = 1;
};
template <> struct F<0> {
  static int const value = 0;
};
```

KDAB

Using Haskell Syntax

```
f 0 = 0
f 1 = 1
f n = f(n – 1) + f(n - 2)
```

KDAB

Functional Programming

- “Pure” Functions: no side-effects
- Recursion instead of loops
- Strong type system
- Lazy evaluation
- Data structures modeled with terms

KDAB

Data Structures

- Haskell List:

  Node(1 Node(2 Node(3 Nil)))
- TMP List:

  node<1, node<2, node<3, void>>>
- Purely syntactic terms
- Recursion and pattern matching

KDAB

Compiler Errors

```
akonadi/itempayloadinternals_p.h: In instantiation of ‘Akonadi::Payload<T>::Payload(const T&) [with T = QObject]’:
akonadi/item.h:657:54:   required from ‘typename boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type
Akonadi::Item::setPayloadImpl(const T&) [with T = QObject; typename boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic,
void>::type = void]’
akonadi/item.h:635:3:   required from ‘void Akonadi::Item::setPayload(const T&) [with T = QObject]’
akonadi/tests/itemhydratest.cpp:114:21:   required from here
corelib/kernel/qobject.h:333:5: error: ‘QObject::QObject(const QObject&)’ is private
In file included from akonadi/item.h:28:0,
                 from akonadi/tests/itemhydratest.cpp:23:
akonadi/itempayloadinternals_p.h:288:40: error: within this context
In file included from QtCore/qmetatype.h:1:0,
                 from corelib/kernel/qvariant.h:48,
                 from corelib/tools/qlocale.h:45,
                 from corelib/io/qtextstream.h:48,
                 from QtCore/qtextstream.h:1,
                 from corelib/io/qdebug.h:50,
                 from kdebug.h:27,
                 from akonadi/entity.h:33,
                 from akonadi/item.h:26,
                 from akonadi/tests/itemhydratest.cpp:23:
corelib/kernel/qmetatype.h: In instantiation of ‘static int QMetaTypeId2<T>::qt_metatype_id() [with T = QObject]’:
corelib/kernel/qmetatype.h:230:44:   required from ‘int qMetaTypeId(T*) [with T = QObject]’
akonadi/itempayloadinternals_p.h:145:58:   required from ‘static int Akonadi::Internal::PayloadTrait<T>::elementMetaTypeId() [with T =
QObject]’
akonadi/item.h:658:3:   required from ‘typename boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type
Akonadi::Item::setPayloadImpl(const T&) [with T = QObject; typename boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic,
void>::type = void]’
akonadi/item.h:635:3:   required from ‘void Akonadi::Item::setPayload(const T&) [with T = QObject]’
akonadi/tests/itemhydratest.cpp:114:21:   required from here
corelib/kernel/qmetatype.h:169:80: error: ‘qt_metatype_id’ is not a member of ‘QMetaTypeId<QObject>’
corelib/kernel/qmetatype.h: In static member function ‘static int QMetaTypeId2<T>::qt_metatype_id() [with T = QObject]’:
corelib/kernel/qmetatype.h:169:83: error: control reaches end of non-void function [-Werror=return-type]
```

KDAB

It's just a backtrace!

- This is a failure of a program execution, not a C++ compiler error
- We know how to read backtraces
  - It's unlikely there are fundamental issues in the libraries you are using
  - Follow the trace from the error back to your code

KDAB

Great, but what can I use it for?

- Example:

```
Vector<double> x,y;
x = 42*x + x*y;
```

- Naïve implementation:

```
tmp1 = 42*x;
tmp2 = x*y;
tmp3 = tmp1 + tmp2;
x = tmp3;
```

KDAB

Great, but what can I use it for?

- Ugly workaround: +=, *=, ...
  - Only avoids some extra temporaries
- Hand-optimized implementation:
  `x[i] = 42*x[i] + x[i]*y[i]`
- Give the compiler a chance to do that
  → Lazy evaluation

KDAB

Expression Templates

- Build syntax tree of an entire expression
- Verify/transform entire expression
- Evaluate eventually to obtain result

```
PlusExpr<
   MultExpr< Val<42>, Var<x> >,
   MultExpr< Var<x> , Var<y> >
>
```

KDAB

That's never going to be fast!

- We see: complicated template structure
- Compiler sees:
  - Only code that matters at run-time
  - The full expression
  - Lots of inline code
- Optimizer has a field day

KDAB

I'll never going to need that!

- QStringBuilder
  - Reduce memory allocations on string concatenations
  - Used almost everywhere
- Eigen
  - Linear algebra
  - Used by: Krita, Digikam, KDE EDU, ...

KDAB

Use-Cases

- Embedded DSLs
  - Embedded: no extra tools required
  - Safer than string-based DSLs (e.g. SQL)
  - Examples: boost::spirit, Eigen, boost::bind
- Performance
  - boost::simd, Eigen, QStringBuilder

KDAB

QStringBuilder

- `#define QT_USE_FAST_OPERATOR_PLUS`
- “Seeding problem”
  `QString s = “Hello” + “World”;`
- Extra conversion step
  `QVariant v = s + “literal”;`
- qtbase/tests/benchmarks/corelib/tools/qstringbuilder

C++98 Limits

- Expression type usually too complicated to declare a variable for it
- Convert to “manageable” type, loses information
- C++11: auto, decltype

KDAB

DIY Expression Templates

- Boost::MPL
  - TMP functions
  - Data structures
- Boost::Proto
  - Toolkit for embedded DSLs
  - Managing expression tree structure
  - Operator overloads, grammar support

KDAB

DIY Expression Templates

- Boost preprocessor library
  - Allows iteration for code generation
- Boost enable_if
  - Allows control over selecting operator overloads
- Hope you don't have to support ancient compilers...

KDAB

One more thing...

```
QSqlQuery q =
  select(Book.title, Author.name)
    .from(Book)
    .innerJoin(Author,
      Book.author == Author.id)
    .where(Book.price > 50.0
      && Book.price <= 100.0);
```

- Will be published soon as free software

KDAB

Conclusion

- TMP is “just” functional programming
- Read compiler errors as backtraces
- TMP is not just theory, you are using it already!

KDAB

Questions?

KDAB

References

- David Vandervoorde, Nocolai M. Josuttis: “C++ Templates – The Complete Guide”, ISBN 0-201-73484-2, Addison-Wesly, 2003
- David Abrahams, Aleksey Gurtovoy: “C++ Template Metaprogramming”, ISBN 0-321-22725-5, Addison-Wesley, 2005
- Todd Veldhuizen, “Expression Templates”, C++ Report, Volume 7, 1995
- “The Boost Meta-Programming Library”, http://www.boost.org/doc/libs/1_54_0/libs/mpl/
- “Boost.Proto”, http://www.boost.org/doc/libs/1_54_0/doc/html/proto.html

KDAB