Writing my own CMS

Why and how
I am writing my own CMS
in Qt & C++

Jens Weller
CppCon – Open Content Session

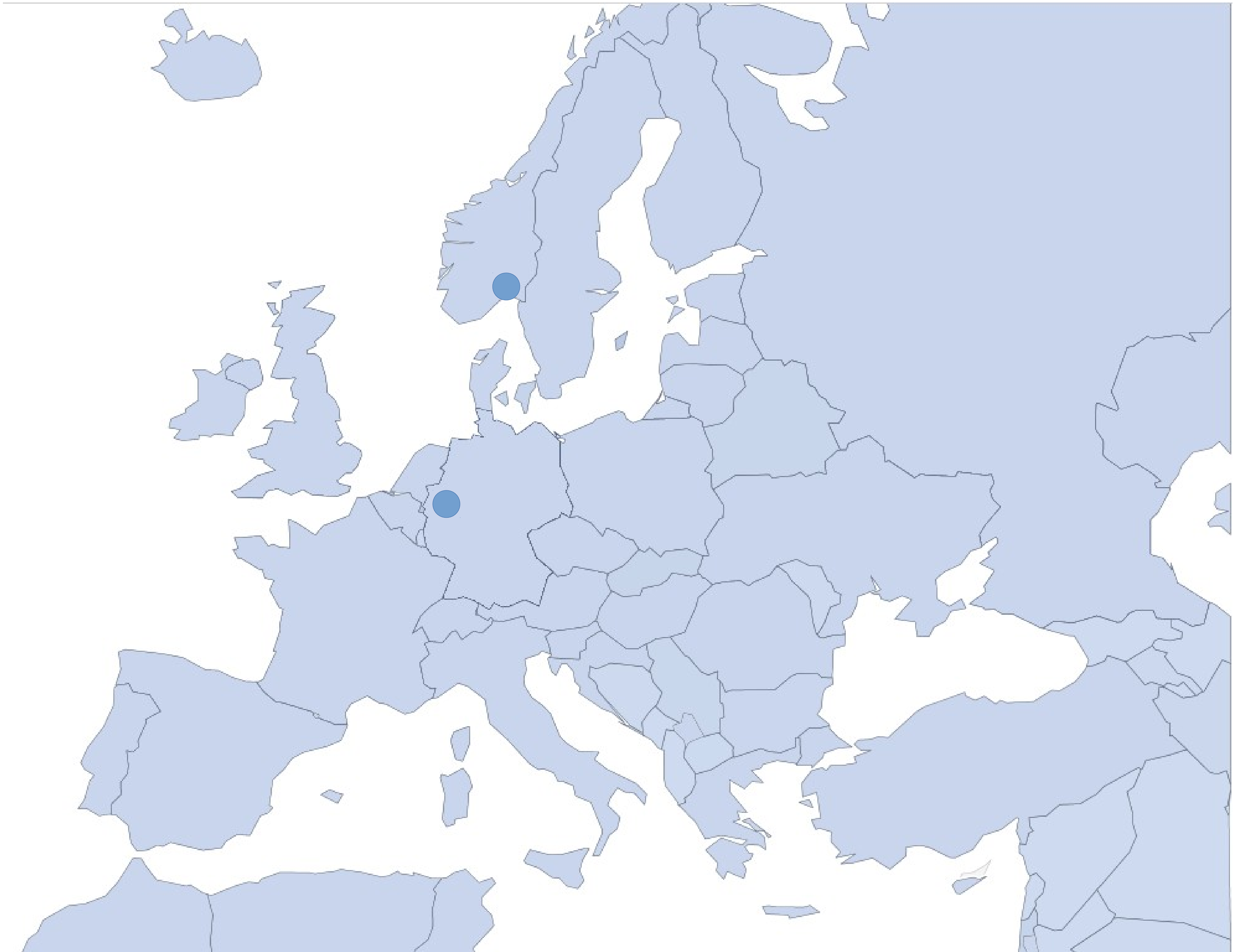
About me

- C++ Evangelist
  - @meetingcpp
- C++ since '98
- '02-'07 Vodafone
- '07 selfemployed / freelancer in C++
- '12 Meeting C++

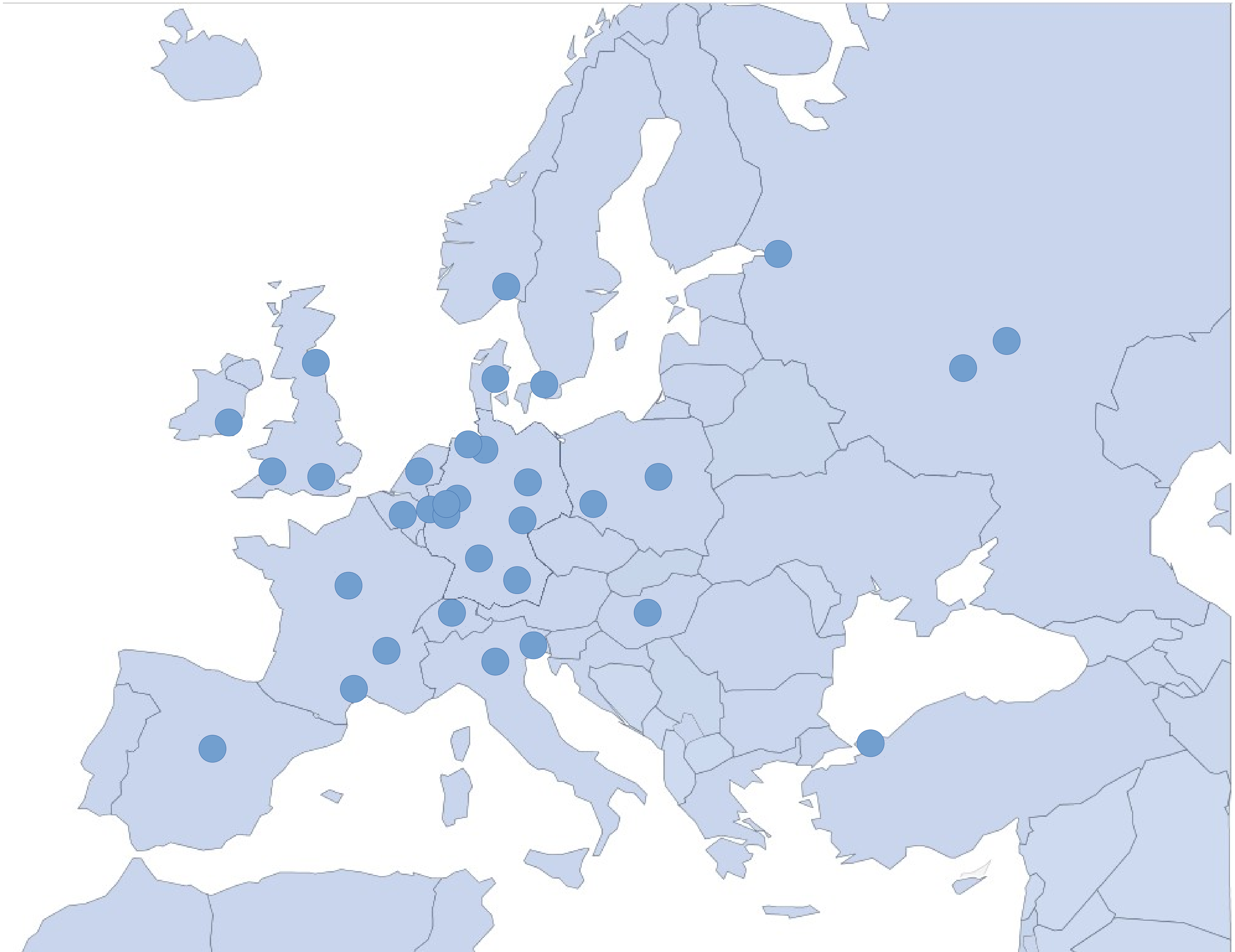
Meeting C++

- C++ Conference
  - 400 Attendees in Berlin
  - Funds my work for C++
- Platform for C++ User Groups
  - Monthly overview of User Group Meetings
- C++ News Network
  - Social Media
  - Weekly Blogroll

C++ User Groups – 2011



C++ User Groups – 2015



Writing my own CMS

- Thinking about “web things” ~ a year now
- I need a solution for my own websites
- Wordpress is not really an alternative
- Do I need a CMS at all?

Reasons

- Update Mess
  - Different Versions require rewriting major parts of my website
  - For every version

Reasons II

- Its a website not an webapp
  - Why a database?
  - Why PHP?
- Static HTML Page would do
  - Jekyll
    - Ruby, yet another language dependency

VIAGRA, CEALIS, PROZAC?
(Or maybe Watches?)

Reasons III

- Security
  - Bot(net)s
  - PHP
- Speed

Disadvantages

- No Dynamic Content
  - Except JavaScript...
- Current Links break
  - All links to pages and posts are invalid
  - Index.php
- Backend:
  - No MultiUser and/or MultiMachine

Writing my own CMS

- Brainfart in June
  - “cool usecase for boostache”
- Started planning in July
- Wasn't bored in August

Writing my own CMS

- Blogseries
  - 10 Entries
  - Documents my progress
- Goals
  - Modern C++ using Qt and boost
    - C++11, templates, generic reusability

Overview

Qt UI Layer

Standard C++ & boost Layer

Implemented Features

- Tree
- Factory
- Context Menu
- QWidgets and data
- Integration of an HTML Texteditor
- Filesystem access
- Serialization

Implemented Features

- Page Tree

```
template< class NameVisitor, class TypeIdVisitor, class IdVisitor, class ...types>
class TreeItem : public std::enable_shared_from_this<
TreeItem< NameVisitor, TypeIdVisitor, IdVisitor,types... > >
{
    using self = TreeItem;
    using const_item_t = std::shared_ptr< const self >;
    using weak_item_t = std::weak_ptr< self >;
    variant node;
    std::vector<item_t> children;
    weak_item_t parent;
public:
    using variant = boost::variant< types...>;
    using item_t = std::shared_ptr< self >;
```

Implemented Features

- Page Tree

```
template< class NameVisitor, class TypeIdVisitor, class IdVisitor, class ...types>
class TreeItem : public std::enable_shared_from_this<
TreeItem< NameVisitor, TypeIdVisitor, IdVisitor,types... > >
{
    using self = TreeItem;
    using const_item_t = std::shared_ptr< const self >;
    using weak_item_t = std::weak_ptr< self >;
    variant node;
    std::vector<item_t> children;
    weak_item_t parent;
public:
    using variant = boost::variant< types...>;
    using item_t = std::shared_ptr< self >;
```

Implemented Features

- Factories

```
template<class AbstractClass,class IdType = size_t,
        class MakeType = boost::function<AbstractClass*()> >
struct Factory
{
    boost::container::flat_map<IdType,MakeType> factory_map;
public:
    void register_factory(IdType type_id,const MakeType& make)
...
    template<class ...args>
    abstract_type* create(IdType id, args&&... a)const
...
```

Implemented Features

- Factories

```
template<class AbstractClass,class IdType = size_t,
        class MakeType = boost::function<AbstractClass*()> >
struct Factory
{
    boost::container::flat_map<IdType,MakeType> factory_map;
public:
    void register_factory(IdType type_id,const MakeType& make)
...
    template<class ...args>
    abstract_type* create(IdType id, args&&... a)const
…
factory.registerType(dir_typeid,boost::bind(boost::factory<DirPanel*>(),_1,_2));
```

Implemented Features

- Generic Context Menus

```
template<class context_sig, class hash_type = size_t>
class ContextMenu
{
    boost::container::flat_map<hash_type,QList<QAction*> > type2menu;
public:
    void registerAction(hash_type type_hash,const QString& text
                                    ,const context_sig& sig, QObject* parent )

    template<class ...args>
    void displayMenu(hash_type type_hash,QPoint pos,args&&... a)
```

Implemented Features

- Generic Context Menus

```
template<class context_sig, class hash_type = size_t>
class ContextMenu
{
    boost::container::flat_map<hash_type,QList<QAction*> > type2menu;
public:
    template<class ...args>
    void displayMenu(hash_type type_hash,QPoint pos,args&&... a)
    {
        auto action = QMenu::exec(type2menu[type_hash],pos);
        if(action)
            action->data(). template value< context_sig >()(std::forward<args>(a)...);
    }
```

Implemented Features

- QWidgets and data...

```
template<class control>
std::string getText(QObject* obj)
{
    control* c = qobject_cast<control*>(obj);
    return c->text().toStdString();
}

std::string getCurrentText(QObject* obj)
std::string getPlainText(QObject* obj)
bool getCheck(QObject* obj)
unsigned int getTimestamp(QObject* obj)
R getValue(QObject* obj)
```

Implemented Features

- QWidgets and data...

```
template<class control>
std::string getText(QObject* obj)
{
    control* c = qobject_cast<control*>(obj);
    return c->text().toStdString();
}

std::string getCurrentText(QObject* obj)
std::string getPlainText(QObject* obj)
bool getCheck(QObject* obj)
unsigned int getTimestamp(QObject* obj)
R getValue(QObject* obj)
```

Implemented Features

- QWidgets and data...

```
template<class SetType>
class Filter
{
    using sig = std::function<void(const SetType&)>;
    using qsig = std::function<SetType(QObject*)>;
public:
    Filter(sig setter, qsig getter,QEvent::Type type  = Qevent::FocusOut):...
    bool operator()(QObject* obj,QEvent* e)
    {
        if(e->type() == eventtype)
            setter(getter(obj));
        return true;
    }
};
```

Implemented Features

- QWidgets and data...

```
class EventFilter : public QObject
{
    Q_OBJECT
public:
    using eventfilter_sig = std::function<bool(QObject*,QEvent*)>;
    explicit EventFilter(eventfilter_sig filter, QObject *parent = 0);
...
protected:
    bool eventFilter(QObject *obj, QEvent *event)override
    {
        return filter(obj,event) && QObject::eventFilter(obj,event);
    }
    eventfilter_sig filter;
```

Implemented Features

- HTML Text Editor
- Integrating TinyMCE3 into my Qt Application
  - QWebView + Qwebkit
  - → Lightning talk tonight.

Implemented Features

- boost::filesystem

```
boost::container::flat_set<std::string> load_dir_recursive(const fs::path& path)
{
    boost::container::flat_set<std::string> set;
    std::string::size_type pathsize = path.generic_string().size()+1;
    for(fs::directory_entry& entry: fs::recursive_directory_iterator(path))
        set.insert(entry.path().generic_string().substr(pathsize));
    return set;
}
```

Implemented Features

- boost::filesystem

```
namespace fs = boost::filesystem;
boost::container::flat_set<std::string> load_dir_recursive(const fs::path& path)
{
    boost::container::flat_set<std::string> set;
    std::string::size_type pathsize = path.generic_string().size()+1;
    for(fs::directory_entry& entry: fs::recursive_directory_iterator(path))
        set.insert(entry.path().generic_string().substr(pathsize));
    return set;
}
```

Implemented Features

- boost::filesystem

```
namespace fs = boost::filesystem;

//create directories for a new project
fs::path p = basepath +"/"+ name;
fs::create_directories(p / "web" / "css");
fs::create_directory(p / "web" / "img");

//when loading document, check for existing archive
bool load_web = fs::exists(basepath + "/" + name +"/"+ "data.dat");
```

Implemented Features

- Serialization

```
#define ELEMENT(TE) TE
#define ELEMENT_MACRO(r, data, i, elem) ar & t. ELEMENT(elem);
#define FRIEND_ELEMENT(...) \
BOOST_PP_SEQ_FOR_EACH_I(ELEMENT_MACRO, _, \
BOOST_PP_VARIADIC_TO_SEQ(__VA_ARGS__))
#define SERIALIZE_IMPL(Type,...) \
template<class Archive>\
void serialize(Archive& ar, Type &t, const unsigned int )\
{ FRIEND_ELEMENT(__VA_ARGS__)}
#define SERIALIZE_DERIVED_IMPL(Type,Base,...) ...
```

Implemented Features

- Serialization
  - Lightning talk Wednesday Evening

```
#define ELEMENT(TE) TE
#define ELEMENT_MACRO(r, data, i, elem) ar & t. ELEMENT(elem);
#define FRIEND_ELEMENT(...) \
BOOST_PP_SEQ_FOR_EACH_I(ELEMENT_MACRO, _, \
BOOST_PP_VARIADIC_TO_SEQ(__VA_ARGS__))
#define SERIALIZE_IMPL(Type,...) \
template<class Archive>\
void serialize(Archive& ar, Type &t, const unsigned int )\
{ FRIEND_ELEMENT(__VA_ARGS__)}
#define SERIALIZE_DERIVED_IMPL(Type,Base,...) ...
```

Implemented Features

- Lists
  - Content Element
    - News, Blogs etc.
  - Start,end
    - Calendar?

Planned Features

- Feeds
  - Building on Lists
- Create HTML
  - Boostache
- FTP support/ upload
- DataStore
  - JSON?
- Content Server
  - Boost.Asio based
  - Import data
    - Lists etc.
- TESTS...

Timeline

- October/Nov
  - Maybe first beta version
  - github

- 1st Quarter '16
  - Add planned features
  - First working version

Demotime!

Questions?

?