Meta Template Foo

Warum?

Um zur Compile-Zeit:

- Datenstrukturen
- Konstanten
- Funktionen

zu generieren.

Auswahl eines Typen:

```
template < bool Select, typename A, typename B >
struct select_type {
    typedef A type;
};

template < typename A, typename B >
struct select_type< false, A, B > {
    typedef B type;
};
```

Anwendung:

```
struct null_mutex {
    void lock() {}
    void unlock() {}
};

template < bool thread_safe = false >
class queue {
public:
    void func() {
        std::lock_guard< mutex_t > lock( mutex_ );
        // ...
    }
private:
    using mutex_t = typename select_type< thread_safe,
        std::mutex, null_mutex >::type;

    mutex_t mutex_;
};
```

Rekursionen

```
template < typename T >
struct list_size;

template <>
struct list_size< std::tuple<> > {
    static constexpr std::size_t size = 0;
};

template <
    typename T,
    typename ... Ts >
struct list_size< std::tuple< T, Ts... > > {
    static constexpr std::size_t size =
        1 + list_size< std::tuple< Ts... > >::size;
};
```

war ein blödes Beispiel:

```
template < typename T >
struct list_size;

template <
    typename ... Ts >
struct list_size< std::tuple< Ts... > > {
    static constexpr std::size_t size = sizeof ...(Ts);
};
```

noch mal Rekursion

```
template < typename T >
struct loop;

template <>
struct loop< std::tuple<> > {};

template <
    typename T,
    typename ... Ts >
struct loop< std::tuple< T, Ts... > >
    : loop< std::tuple< Ts... > >
{
    ~loop()
    {
        std::cout << "name: "
            << typeid( T ).name() << std::endl;
    }
};
```

hatten wir schon Rekursionen?

```
constexpr int facu( int f ) {
    return f == 1
        ? 1
        : f * facu( f - 1 );
}

int main()
{
    static_assert( facu( 1 ) == 1, "1" );
    static_assert( facu( 2 ) == 2, "2" );
    static_assert( facu( 3 ) == 6, "6" );
}
```

Substitution Failure is not an Error

```
struct no_such_type {};

template < typename T >
struct extract_meta_type
{
    template < class U >
    static typename U::meta_type check( U* )  { return U::meta_type(); }
    static no_such_type          check( ... ) { return no_such_type(); }

    typedef decltype( check( static_cast< T* >( nullptr ) ) ) type;
};
```

Mixins

```
template <
    typename locking        = single_threading_impl,
    typename time_mark      = times_in_milliseconds,
    typename files          = std_out >
class logger : locking, time_mark, files
{
    // ...
};
```

Beispiele:

```
typedef hammer::nrf51422_xxaa_s310 device;

typedef hammer::timer::interval_timer<
    base_timer_interval,
    hammer::callback_member< device_logic, &device_logic::timer_callback, &logic > > main_timer;

typedef hammer::gpio::output_pin< device::P0_26 > dac_clock;
typedef hammer::gpio::output_pin< device::P0_27 > dac_data;
typedef hammer::gpio::output_pin< device::P0_25, hammer::gpio::inverted > dac_sync;
typedef hammer::gpio::output_pin< device::P0_02 > fans_enabled;
typedef hammer::gpio::input_pin< device::P0_29, hammer::gpio::inverted > ami_overtemperature;

typedef ad5310< dac_sync, dac_clock, dac_data > high_voltage_dac;

typedef hammer::gpio::output_pin< device::P0_08, hammer::gpio::inverted, hammer::gpio::toggle > green_led;
typedef hammer::gpio::output_pin< device::P0_09, hammer::gpio::inverted > red_led;

typedef radio_sender radio;

struct hardware : hammer::device< hardware, device > {

    typedef boost::mpl::vector< dac_clock, dac_data, dac_sync > dac_pins;
    typedef boost::mpl::vector< green_led, red_led >            debug_pins;

    typedef boost::mpl::vector<
        dac_pins,
        debug_pins,
        main_timer,
        fans_enabled,
        ami_overtemperature
    > peripherals;
};
```

```
using namespace bluetoe;

static constexpr int io_pin = 19;

static std::uint8_t io_pin_write_handler( bool state )
{
    // the GPIO pin according to the received value: 0 = off, 1 = on
    NRF_GPIO->OUT = state
        ? NRF_GPIO->OUT | ( 1 << io_pin )
        : NRF_GPIO->OUT & ~( 1 << io_pin );

    return error_codes::success;
}

typedef server<
    service<
        service_uuid< 0xC11169E1, 0x6252, 0x4450, 0x931C, 0x1B43A318783B >,
        characteristic<
            free_write_handler< bool, io_pin_write_handler >
        >
    >
> blinky_server;

blinky_server gatt;

nrf51< blinky_server > gatt_srv;
```

```
// a can message is defined, by the message id and a list of signals to be
typedef mpl::map<
    // FAG0
    mpl::pair<
        mpl::integral_c< std::uint32_t, 0xf0 >,
        arm::vector< State_KM, State_SL>
    >,
    // FAG1
    mpl::pair<
        mpl::integral_c< std::uint32_t, 0xf1 >,
        arm::vector< PedalFrequency, PedalTorque, Direction >
    >,
    // FAG2
    mpl::pair<
        mpl::integral_c< std::uint32_t, 0xf2 >,
        arm::vector< CrankAngle >
    >,
```