CORUJA

Uma alternativa ao padrão observador em C++11

22 de Setembro de 2018

Ricardo Cosme

Instituto Tecgraf de Desenvolvimento de
Software Técnico-Científico da PUC-Rio
Tecgraf/PUC-Rio

CONTEXTO

- → Há 2 anos trabalhando com um software desktop para geofísicos da Petrobrás
  - → Visualização 2D e 3D, CUDA, análise de dados e projeções.
    - → Múltiplas visões reativas de um dado
  - → C++11 - Programação genérica(GP) e funcional(FP).

PADRÃO OBSERVADOR

PROBLEMA

A alteração do estado saliente de um objeto deve **imediatamente** atualizar zero ou mais visões do mesmo.

REQUISITOS

A solução deve considerar:

→ Baixo acoplamento entre as partes.

REQUISITOS

A solução deve considerar:

- → Baixo acoplamento entre as partes.
- → Número dinâmico de visões.

REQUISITOS

A solução deve considerar:

- → Baixo acoplamento entre as partes.
- → Número dinâmico de visões.
- → Contexto singlethread. (Widgets e OpenGL)

REQUISITOS

A solução deve considerar:

- → Baixo acoplamento entre as partes.
- → Número dinâmico de visões.
- → Contexto singlethread. (Widgets e OpenGL)
- → push model: atualizações imediatas via callback

PADRÃO OBSERVADOR

Objetivo

Definir uma dependência one-to-many entre objetos de forma que quando o estado de um objeto é alterado, todas as suas dependências são notificadas e atualizadas automaticamente. [1]

[1] Design Patterns: Elements Of Reusable Object-Oriented Software (1994, Addison Wesley)

OBSERVER

```cpp
struct observer {
  virtual ~observer() = default;
  virtual void update(Subject&);
};
```

OBSERVER

```cpp
struct observer {
  virtual ~observer() = default;
  virtual void update(Subject&);
};
```

Pouco flexível.

→ Preciso somente de uma função.

OBSERVER

```
struct observer {
  virtual ~observer() = default;
  virtual void update(Subject&);
};
```

Polimorfismo através de herança.

→ Restringe demasiadamente o design de quem usa.

OBSERVER

```
struct observer {
  virtual ~observer() = default;
  virtual void update(Subject&);
};
```

1. O que mudou?

OBSERVER

```
struct observer {
  virtual ~observer() = default;
  virtual void update(Subject&);
};
```

1. O que mudou?
2. Uma única função para todas as reações (complexidade).

SUBJECT (OBSERVABLE)

```
struct subject {
  virtual ~subject() = default;
  virtual void attach(observer&);
  virtual void detach(observer&);
  virtual void notify();

  list<observer*> observers;
};
```

SUBJECT (OBSERVABLE)

```cpp
struct subject {
  virtual ~subject() = default;
  virtual void attach(observer&);
  virtual void detach(observer&);
  virtual void notify();

  list<observer*> observers;
};
```

Polimorfismo através de herança.

→ Restringe demasiadamente o design de quem usa.

SUBJECT (OBSERVABLE)

```
struct subject {
  virtual ~subject() = default;
  virtual void attach(observer&);
  virtual void detach(observer&);
  virtual void notify();

  list<observer*> observers;
};
```

Demasiadamente operacional (baixo nível).

SUBJECT (OBSERVABLE)

```
struct subject {
  virtual ~subject() = default;
  virtual void attach(observer&);
  virtual void detach(observer&);
  virtual void notify();

  list<observer*> observers;
};
```

1. Notificação global.
   → O que mudou?

SUBJECT (OBSERVABLE)

```
struct subject {
  virtual ~subject() = default;
  virtual void attach(observer&);
  virtual void detach(observer&);
  virtual void notify();

  list<observer*> observers;
};
```

1. Notificação global.
   → O que mudou?
2. Notificação manual.

SIGNALS&SLOTS

```
// Signal (Observable)
signal<void(string)> name_sig;
```

SIGNALS&SLOTS

```
// Signal (Observable)
signal<void(string)> name_sig;
```

Soluciona

→ Composição ao invés de herança

SIGNALS&SLOTS

```
//Signal (Observable)
signal<void(string)> name_sig;
```

Soluciona

- → Composição ao invés de herança
- → Específico para um atributo: `name`

SIGNALS&SLOTS

```
// Slot (Observer) attach()
auto conn = name_sig.connect
  ([](string name){do_something(name);});

name_sig("abc"); // notify()

conn.disconnect(); // detach()
```

Soluciona

→ Observer é simplesmente um function object.

SIGNALS&SLOTS

```
// Slot (Observer) attach()
auto conn = name_sig.connect
  ([](string name){do_something(name);});

name_sig("abc"); // notify()

conn.disconnect(); // detach()
```

Soluciona

- → Observer é simplesmente um function object.
- → Função só se preocupa com uma coisa: `name`.

SIGNALS&SLOTS

```
// Slot (Observer) attach()
auto conn = name_sig.connect
  ([](string name){do_something(name);});

name_sig("abc"); // notify()

conn.disconnect(); // detach()
```

Soluciona

→ Observer é simplesmente um function object.

→ Função só se preocupa com uma coisa: `name`.

SIGNALS&SLOTS

```
// Slot (Observer) attach()
auto conn = name_sig.connect
  ([](string name){do_something(name);});

name_sig("abc"); // notify()

conn.disconnect(); // detach()
```

Soluciona

- → Observer é simplesmente um function object.
- → Função só se preocupa com uma coisa: `name`.

SIGNALS&SLOTS

É suficiente?

→ Inversão de Controle(IoC) (callbacks)

SIGNALS&SLOTS

É suficiente?

- → Inversão de Controle(IoC) (callbacks)
- → Notificação manual

SIGNALS&SLOTS

É suficiente?

→ Inversão de Controle(IoC) (callbacks)

→ Notificação manual

→ Código boilerplate

SIGNALS&SLOTS

É suficiente?

→ Inversão de Controle(IoC) (callbacks)

→ Notificação manual

→ Código boilerplate

→ Dependência da ordem de registro de slots

SIGNALS&SLOTS

É suficiente?

- → Inversão de Controle(IoC) (callbacks)
- → Notificação manual
- → Código boilerplate
- → Dependência da ordem de registro de slots
- → Complexidade em compor observáveis (sinais)

OBJETOS OBSERVÁVEIS

PROBLEMA #1

```
struct person_t {
  string first_name, surname;
};
```

PROBLEMA #1

```
struct person_t {
  string first_name, surname;
};
```

Problema #1: Reagir a alteração de first_name ou surname

PROBLEMA #1 - SOLUÇÃO AD HOC

```
struct person_t {
  //setter
  void first_name(string v) {
      _first_name = move(v);
      _change_first_name(_first_name);
  }

  //getter
  const string& first_name() const noexcept
  { return _first_name; }

  //connect
  template<typename F>
  void change_first_name(F&& f)
  { _change_first_name.connect(forward<F>(f)); }
private:
  signal<void(const string&)> _change_first_name;
  string _first_name;
};
```

PROBLEMA #1 - SOLUÇÃO AD HOC

```
struct person_t {
  void first_name(string v) {
      _first_name = move(v);
      _change_first_name(_first_name);
  }

  const string& first_name() const noexcept
  { return _first_name; }

  template<typename F>
  void change_first_name(F&& f)
  { _change_first_name.connect(forward<F>(f)); }

  void surname(string v) {
      _surname = move(v);
      _change_surname(_surname);
  }

  const string& surname() const noexcept
  { return _surname; }

  template<typename F>
  void change_surname(F&& f)
  { _change_surname.connect(forward<F>(f)); }
private:
  signal<void(const string&)> _change_first_name, _change_surname;
  string _first_name, _surname;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
struct person_t {
  coruja::object<string> first_name, surname;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
struct person_t {
  coruja::object<string> first_name, surname;
};
```

Ad hoc	Coruja
surname(v);	surname = v;
auto v = surname();	auto v = surname.observed();
change_surname.connect(f);	surname.after_change(f);

PROBLEMA #1 - SOLUÇÃO CORUJA

Observable

Notifica imediatamente observadores interessados em uma ação específica de alteração de estado de um objeto de tipo T.

Requisitos

```
Observable::observed_t

observed_t observed() const noexcept

//There has to be at least one observable action
//FunctionObject must be CopyConstructible
template<typename FunctionObject>
connection action(FunctionObject)
```

DefaultConstructible

PROBLEMA #1 - SOLUÇÃO CORUJA

ObservableObject

Refina o conceito Observable com uma ação observável geral `after_change`

Requisitos

```
//FunctionObject should be void(const Observable::
    observed_t&)
template<typename FunctionObject>
after_change_connection_t after_change(FunctionObject)

Observable::after_change_connection_t
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
    using Derived = typename std::conditional<
        std::is_same<Derived_, void>::value, object, Derived_>::type;

    using after_change_t=Signal<void(Derived&)>;
public:
    using observed_t = T;
    using after_change_connection_t = typename after_change_t::connection_t;

    object() = default;

    explicit object(observed_t observed) : _observed(move(observed)) {}

    object(object&&)
      noexcept(is_nothrow_move_constructible<observed_t>::value) {/* ... */}

    object& operator=(object&&)
      noexcept(is_nothrow_move_assignable<observed_t>::value) {/* ... */}

    const observed_t& observed() const noexcept { return _observed; }
    /* code */
private:
    observed_t _observed;
    after_change_t _after_change;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```cpp
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
    using Derived = typename std::conditional<
        std::is_same<Derived_, void>::value, object, Derived_>::type;

    using after_change_t=Signal<void(Derived&)>;
public:
    using observed_t = T;
    using after_change_connection_t = typename after_change_t::connection_t;

    object() = default;

    explicit object(observed_t observed) : _observed(move(observed)) {}

    object(object&&)
      noexcept(is_nothrow_move_constructible<observed_t>::value) {/* ... */}

    object& operator=(object&&)
      noexcept(is_nothrow_move_assignable<observed_t>::value) {/* ... */}

    const observed_t& observed() const noexcept { return _observed; }
    /* code */
private:
    observed_t _observed;
    after_change_t _after_change;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
    using Derived = typename std::conditional<
        std::is_same<Derived_, void>::value, object, Derived_>::type;

    using after_change_t=Signal<void(Derived&)>;
public:
    using observed_t = T;
    using after_change_connection_t = typename after_change_t::connection_t;

    object() = default;

    explicit object(observed_t observed) : _observed(move(observed)) {}

    object(object&&)
      noexcept(is_nothrow_move_constructible<observed_t>::value) {/*...*/}

    object& operator=(object&&)
      noexcept(is_nothrow_move_assignable<observed_t>::value) {/*...*/}

    const observed_t& observed() const noexcept { return _observed; }
    /* code */
private:
    observed_t _observed;
    after_change_t _after_change;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```cpp
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
    using Derived = typename std::conditional<
        std::is_same<Derived_, void>::value, object, Derived_>::type;

    using after_change_t=Signal<void(Derived&)>;
public:
    using observed_t = T;
    using after_change_connection_t = typename after_change_t::connection_t;

    object() = default;

    explicit object(observed_t observed) : _observed(move(observed)) {}

    object(object&&)
      noexcept(is_nothrow_move_constructible<observed_t>::value) {/*...*/}

    object& operator=(object&&)
      noexcept(is_nothrow_move_assignable<observed_t>::value) {/*...*/}

    const observed_t& observed() const noexcept { return _observed; }
    /* code */
private:
    observed_t _observed;
    after_change_t _after_change;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
    using Derived = typename std::conditional<
        std::is_same<Derived_, void>::value, object, Derived_>::type;

    using after_change_t=Signal<void(Derived&)>;
public:
    using observed_t = T;
    using after_change_connection_t = typename after_change_t::connection_t;

    object() = default;

    explicit object(observed_t observed) : _observed(move(observed)) {}

    object(object&&)
      noexcept(is_nothrow_move_constructible<observed_t>::value) {/* ... */}

    object& operator=(object&&)
      noexcept(is_nothrow_move_assignable<observed_t>::value) {/* ... */}

    const observed_t& observed() const noexcept { return _observed; }
    /* code */
private:
    observed_t _observed;
    after_change_t _after_change;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
public:

    /* code */

    template<typename F> //void(Derived&)
    enable_if_is_invocable_t<after_change_connection_t, F, Derived&>
    after_change(F&& f)
    { return _after_change.connect(forward<F>(f)); }

    template<typename F> //void(const observed_t&)
    enable_if_is_invocable_t<after_change_connection_t, F, const observed_t&>
    after_change(F&& f)
    { return _after_change.connect
            (detail::lift_to_observable{forward<F>(f)}); }
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<class T, class Derived_ = void, template <typename> class Signal = signal>
class object /* Models ObservableObject */ {
public:

    /* code */

    template<typename F> // void(Derived&)
    enable_if_is_invocable_t<after_change_connection_t, F, Derived&>
    after_change(F&& f)
    { return _after_change.connect(forward<F>(f)); }

    template<typename F> // void(const observed_t&)
    enable_if_is_invocable_t<after_change_connection_t, F, const observed_t&>
    after_change(F&& f)
    { return _after_change.connect
            (detail::lift_to_observable{forward<F>(f)}); }
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename F>
struct lift_to_observable_impl {
    template<typename... ObservableObjects>
    auto operator()(ObservableObjects&&... objects)
    CORUJA_DECLTYPE_AUTO_RETURN
    ( f(objects.observed()...) )

    F f;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename F>
struct lift_to_observable_impl {
    template<typename... ObservableObjects>
    auto operator()(ObservableObjects&&... objects)
    CORUJA_DECLTYPE_AUTO_RETURN
    ( f(objects.observed()...) )

    F f;
};
```

```
//ObservableObject<T1> vs ObservableObject<T2>
//T vs ObservableObject<T>
//ObservableObject<T> vs T
operator{==,!=,<,>,<=,>=}
```

→ O sinal é desconsiderado (sujeito a alterações)

PROBLEMA #1 - SOLUÇÃO AD HOC

```
auto print_fullname = [&p](string)
  { cout << p.first_name() + p.surname(); };

array<any_connection, 2> conns {
    p.change_first_name(print_fullname),
    p.change_surname(print_fullname),
};

p.first_name("jimmy");

for(auto& c : conns) c.disconnect();
```

PROBLEMA #1 - SOLUÇÃO AD HOC

```
auto print_fullname = [&p](string)
  { cout << p.first_name() + p.surname(); };

array<any_connection, 2> conns {
    p.change_first_name(print_fullname),
    p.change_surname(print_fullname),
};

p.first_name("jimmy");

for(auto& c : conns) c.disconnect();
```

PROBLEMA #1 - SOLUÇÃO AD HOC

```
auto print_fullname = [&p](string)
  { cout << p.first_name() + p.surname(); };

array<any_connection, 2> conns {
    p.change_first_name(print_fullname),
    p.change_surname(print_fullname),
};

p.first_name("jimmy");

for(auto& c : conns) c.disconnect();
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
auto print = [](string s){cout << s;};

auto fullname = p.first_name + p.surname;

auto c = fullname.after_change(print);

p.first_name = "jimmy";

c.disconnect();
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
auto print = [](string s){cout << s;};

auto fullname = p.first_name + p.surname;

auto c = fullname.after_change(print);

p.first_name = "jimmy";

c.disconnect();
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
auto print = [](string s){cout << s;};

auto fullname = p.first_name + p.surname;

auto c = fullname.after_change(print);

p.first_name = "jimmy";

c.disconnect();
```

PROBLEMA #1 - WIDGET::ENABLE

```
// AdHoc
void Widget::enable(bool)

auto conn = name.after_change
  ([&w](string s){ w.enable(!s.empty()); });

conn.disconnect();
```

PROBLEMA #1 - WIDGET::ENABLE

```
// AdHoc
void Widget::enable(bool)

auto conn = name.after_change
  ([&w](string s){ w.enable(!s.empty()); });

conn.disconnect();
```

- → Inversão de controle (IoC).
    - → Bug se `&w` mudar.

PROBLEMA #1 - WIDGET::ENABLE

```cpp
// AdHoc
void Widget::enable(bool)

auto conn = name.after_change
  ([&w](string s){ w.enable(!s.empty()); });

conn.disconnect();
```

- → Inversão de controle (IoC).
    - → Bug se `&w` mudar.
- → Tenho que gerenciar a conexão.

PROBLEMA #1 - SOLUÇÃO CORUJA

ObservableView

Refina o conceito Observable, é Semiregular e operações de cópia, move e atribuição são O(1).

```
//T2 F(T1)
//O<T2> transform(O<T1>, F)

template<typename ObservableObject, typename F>
inline enable_if_t<
    is_observable_object<ObservableObject>::value,
    detail::transform_object_t<ObservableObject, F>> //O<T2>
transform(ObservableObject&& o, F&& f)
```

PROBLEMA #1 - SOLUÇÃO CORUJA

ObservableView

Refina o conceito Observable, é Semiregular e operações de cópia, move e atribuição são O(1).

```
//T2 F(T1)
//O<T2> transform(O<T1>, F)

template<typename ObservableObject, typename F>
inline enable_if_t<
    is_observable_object<ObservableObject>::value,
    detail::transform_object_t<ObservableObject, F>> //O<T2>
transform(ObservableObject&& o, F&& f)
```

PROBLEMA #1 - SOLUÇÃO CORUJA

ObservableView

Refina o conceito Observable, é Semiregular e operações de cópia, move e atribuição são O(1).

```
//T2 F(T1)
//O<T2> transform(O<T1>, F)

template<typename ObservableObject, typename F>
inline enable_if_t<
    is_observable_object<ObservableObject>::value,
    detail::transform_object_t<ObservableObject, F>> //O<T2>
transform(ObservableObject&& o, F&& f)
```

PROBLEMA #1 - WIDGET::ENABLE

```
// ObservableObject::observed_t is bool
template<typename ObservableObject>
void Widget::enable(ObservableObject&&)

w.enable(transform(name, [](string s){return !s.empty();}));
```

PROBLEMA #1 - WIDGET::ENABLE

```
//ObservableObject::observed_t is bool
template<typename ObservableObject>
void Widget::enable(ObservableObject&&)

w.enable(transform(name, [](string s){return !s.empty();}));
```

PROBLEMA #1 - WIDGET::ENABLE

```
// ObservableObject::observed_t is bool
template<typename ObservableObject>
void Widget::enable(ObservableObject&&)

w.enable(transform(name, [](string s){return !s.empty();}));
```

→ Não há inversão de controle (IoC)

PROBLEMA #1 - WIDGET::ENABLE

```
//ObservableObject::observed_t is bool
template<typename ObservableObject>
void Widget::enable(ObservableObject&&)

w.enable(transform(name, [](string s){return !s.empty();}));
```

- → Não há inversão de controle (IoC)
- → Burocracias gerenciadas pelo widget
    - → Conexão
    - → Ponteiro para o widget

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename From, typename Transform>
struct transform_object /* Models ObservableView */ {
    using observed_t = result_of_t<Transform(detail::observed_t<From>)>;;
    using after_change_connection_t = typename From::after_change_connection_t;

    transform_object() = default;
    transform_object(From from, Transform transform)
        : _transform(move(transform))
        , _from(move(from))
    {}

    template<typename F>
    after_change_connection_t after_change(F&& f) {
        return _from.after_change
            (detail::after_change_cbk<From, Transform, F>
             {_transform, forward<F>(f)});
    }

    observed_t observed() const noexcept
    { return _transform(_from.observed()); }
private:
    ranges::semiregular_t<Transform> _transform;
    From _from;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename From, typename Transform>
struct transform_object /* Models ObservableView */ {
    using observed_t = result_of_t<Transform(detail::observed_t<From>)>;;
    using after_change_connection_t = typename From::after_change_connection_t;

    transform_object() = default;
    transform_object(From from, Transform transform)
        : _transform(move(transform))
        , _from(move(from))
    {}

    template<typename F>
    after_change_connection_t after_change(F&& f) {
        return _from.after_change
            (detail::after_change_cbk<From, Transform, F>
             {_transform, forward<F>(f)});
    }

    observed_t observed() const noexcept
    { return _transform(_from.observed()); }
private:
    ranges::semiregular_t<Transform> _transform;
    From _from;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename From, typename Transform>
struct transform_object /* Models ObservableView */ {
    using observed_t = result_of_t<Transform(detail::observed_t<From>)>;;
    using after_change_connection_t = typename From::after_change_connection_t;

    transform_object() = default;
    transform_object(From from, Transform transform)
        : _transform(move(transform))
        , _from(move(from))
    {}

    template<typename F>
    after_change_connection_t after_change(F&& f) {
        return _from.after_change
            (detail::after_change_cbk<From, Transform, F>
             {_transform, forward<F>(f)});
    }

    observed_t observed() const noexcept
    { return _transform(_from.observed()); }
private:
    ranges::semiregular_t<Transform> _transform;
    From _from;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```cpp
template<typename From, typename Transform>
struct transform_object /* Models ObservableView */ {
    using observed_t = result_of_t<Transform(detail::observed_t<From>)>;;
    using after_change_connection_t = typename From::after_change_connection_t;

    transform_object() = default;
    transform_object(From from, Transform transform)
        : _transform(move(transform))
        , _from(move(from))
    {}

    template<typename F>
    after_change_connection_t after_change(F&& f) {
        return _from.after_change
            (detail::after_change_cbk<From, Transform, F>
             {_transform, forward<F>(f)});
    }

    observed_t observed() const noexcept
    { return _transform(_from.observed()); }
private:
    ranges::semiregular_t<Transform> _transform;
    From _from;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
namespace coruja { namespace detail {

template<typename From, typename Transform, typename F>
struct after_change_cbk : protected Transform {
    after_change_cbk(Transform t, F f)
        : Transform(move(t))
        , _f(move(f))
    {}

    void operator()(const typename From::observed_t& from)
    { _f(Transform::operator()(from)); }

    F _f;
};

}}
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename Transform, typename... Objects>
auto lift(Transform&&, Objects&...)

template<typename O1, typename O2>
auto operator+(O1& o1, O2& o2)
{ return lift(Plus{}, o1, o2); }
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```cpp
template<typename Transform, typename... Objects>
auto lift(Transform&&, Objects&...)

template<typename O1, typename O2>
auto operator+(O1& o1, O2& o2)
{ return lift(Plus{}, o1, o2); }
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename T, typename Transform, typename... Objects>
class lift_object : view_base {
  using From = boost::fusion::vector<Objects...>;
public:
  using observed_t = T;
  using after_change_connection_t = connections<
    typename remove_reference_t<Objects>::after_change_connection_t...>;

  template<typename F>
  after_change_connection_t after_change(F&& f) {
    using namespace boost::fusion;
    using conns_t = typename after_change_connection_t::type;
    using Obj2Conn = vector<From&, conns_t&>;

    conns_t conns;
    auto obj2conn = zip_view<Obj2Conn>(Obj2Conn(_objects, conns));

    for_each(obj2conn, detail::connect_object
             <From, Transform, remove_reference_t<F>, after_change_connection_t>
             {_objects, _transform, f});

    return {std::move(conns)};
  }

  /* ... */
private:
  mutable ranges::semiregular_t<Transform> _transform;
  From _objects;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename T, typename Transform, typename... Objects>
class lift_object : view_base {
  using From = boost::fusion::vector<Objects...>;
public:
  using observed_t = T;
  using after_change_connection_t = connections<
    typename remove_reference_t<Objects>::after_change_connection_t...>;

  template<typename F>
  after_change_connection_t after_change(F&& f) {
    using namespace boost::fusion;
    using conns_t = typename after_change_connection_t::type;
    using Obj2Conn = vector<From&, conns_t&>;

    conns_t conns;
    auto obj2conn = zip_view<Obj2Conn>(Obj2Conn(_objects, conns));

    for_each(obj2conn, detail::connect_object
             <From, Transform, remove_reference_t<F>, after_change_connection_t>
             {_objects, _transform, f});

    return {std::move(conns)};
  }

  /* ... */
private:
  mutable ranges::semiregular_t<Transform> _transform;
  From _objects;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename T, typename Transform, typename... Objects>
class lift_object : view_base {
  using From = boost::fusion::vector<Objects...>;
public:
  using observed_t = T;
  using after_change_connection_t = connections<
    typename remove_reference_t<Objects>::after_change_connection_t...>;

  template<typename F>
  after_change_connection_t after_change(F&& f) {
    using namespace boost::fusion;
    using conns_t = typename after_change_connection_t::type;
    using Obj2Conn = vector<From&, conns_t&>;

    conns_t conns;
    auto obj2conn = zip_view<Obj2Conn>(Obj2Conn(_objects, conns));

    for_each(obj2conn, detail::connect_object
             <From, Transform, remove_reference_t<F>, after_change_connection_t>
             {_objects, _transform, f});

    return {std::move(conns)};
  }

  /* ... */
private:
  mutable ranges::semiregular_t<Transform> _transform;
  From _objects;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```cpp
template<typename T, typename Transform, typename... Objects>
class lift_object : view_base {
  using From = boost::fusion::vector<Objects...>;
public:
  using observed_t = T;
  using after_change_connection_t = connections<
    typename remove_reference_t<Objects>::after_change_connection_t...>;

  template<typename F>
  after_change_connection_t after_change(F&& f) {
    using namespace boost::fusion;
    using conns_t = typename after_change_connection_t::type;
    using Obj2Conn = vector<From&, conns_t&>;

    conns_t conns;
    auto obj2conn = zip_view<Obj2Conn>(Obj2Conn(_objects, conns));

    for_each(obj2conn, detail::connect_object
             <From, Transform, remove_reference_t<F>, after_change_connection_t>
             {_objects, _transform, f});

    return {std::move(conns)};
  }

  /* ... */
private:
  mutable ranges::semiregular_t<Transform> _transform;
  From _objects;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename T, typename Transform, typename... Objects>
class lift_object : view_base {
  using From = boost::fusion::vector<Objects...>;
public:
  using observed_t = T;
  using after_change_connection_t = connections<
    typename remove_reference_t<Objects>::after_change_connection_t...>;

  template<typename F>
  after_change_connection_t after_change(F&& f) {
    using namespace boost::fusion;
    using conns_t = typename after_change_connection_t::type;
    using Obj2Conn = vector<From&, conns_t&>;

    conns_t conns;
    auto obj2conn = zip_view<Obj2Conn>(Obj2Conn(_objects, conns));

    for_each(obj2conn, detail::connect_object
             <From, Transform, remove_reference_t<F>, after_change_connection_t>
             {_objects, _transform, f});

    return {std::move(conns)};
  }

  /* ... */
private:
  mutable ranges::semiregular_t<Transform> _transform;
  From _objects;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename Objects, typename Transform, typename F, typename Observed>
struct lift_f : private Transform {
    /* ... */
    void operator()(const Observed&)
    { _f(boost::fusion::invoke(static_cast<Transform&>(*this), _objects)); }

    Objects _objects;
    F _f;
};

template<typename From, typename Transform, typename F, typename Conn>
struct connect_object {
    template<typename Obj2Conn>
    void operator()(Obj2Conn&& obj2conn) const {
        using namespace boost::fusion;
        auto& object = at_c<0>(obj2conn);
        auto& conn = at_c<1>(obj2conn);

        using Obj = typename std::remove_reference<
            typename result_of::at_c<Obj2Conn, 0>::type>::type;

        //TODO: Update from
        conn = object.after_change
            (lift_f<From, Transform, F, typename Obj::observed_t>
             (_transform, _from, _f));
    }

    From& _from;
    Transform& _transform;
    F& _f;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename Objects, typename Transform, typename F, typename Observed>
struct lift_f : private Transform {
    /* ... */
    void operator()(const Observed&)
    { _f(boost::fusion::invoke(static_cast<Transform&>(*this), _objects)); }

    Objects _objects;
    F _f;
};

template<typename From, typename Transform, typename F, typename Conn>
struct connect_object {
    template<typename Obj2Conn>
    void operator()(Obj2Conn&& obj2conn) const {
        using namespace boost::fusion;
        auto& object = at_c<0>(obj2conn);
        auto& conn = at_c<1>(obj2conn);

        using Obj = typename std::remove_reference<
            typename result_of::at_c<Obj2Conn, 0>::type>::type;

        //TODO: Update from
        conn = object.after_change
            (lift_f<From, Transform, F, typename Obj::observed_t>
             (_transform, _from, _f));
    }

    From& _from;
    Transform& _transform;
    F& _f;
};
```

PROBLEMA #1 - SOLUÇÃO CORUJA

```
template<typename Objects, typename Transform, typename F, typename Observed>
struct lift_f : private Transform {
    /* ... */
    void operator()(const Observed&)
    { _f(boost::fusion::invoke(static_cast<Transform&>(*this), _objects)); }

    Objects _objects;
    F _f;
};

template<typename From, typename Transform, typename F, typename Conn>
struct connect_object {
    template<typename Obj2Conn>
    void operator()(Obj2Conn&& obj2conn) const {
        using namespace boost::fusion;
        auto& object = at_c<0>(obj2conn);
        auto& conn = at_c<1>(obj2conn);

        using Obj = typename std::remove_reference<
            typename result_of::at_c<Obj2Conn, 0>::type>::type;

        //TODO: Update from
        conn = object.after_change
            (lift_f<From, Transform, F, typename Obj::observed_t>
             (_transform, _from, _f));
    }

    From& _from;
    Transform& _transform;
    F& _f;
};
```

PERFORMANCE ADHOC X LIFT - ASSIGNMENT

```
object<string> s1, s2;

//AdHoc
auto l = [&r,&o1,&o2](string){r = o1.observed() + o2.observed();};
o1.after_change(l);
o2.after_change(l);
o1 = "abc";

//Lift
auto s1s2 = s1 + s2;
s1s2.after_change([&r](string s){ r = s; });
o1 = "abc";
```

Método	Média	Mediana	Desvio padrão
`lift`	33ns	33ns	0ns
`adHoc`	33ns	33ns	1ns

AMD Ryzen 7 1700X with GCC 8.2.0 -O3 and 10x repetitions

PERFORMANCE ADHOC X LIFT - AFTER_CHANGE()

```
//AdHoc
for(size_t i(0);i<state.range(0);++i) {
  auto c1 = o1.after_change(l);
  auto c2 = o2.after_change(l);
  conns.push_back({move(c1), move(c2)});
}

//Lift
for(size_t i(0);i<state.range(0);++i) {
  auto c = concat.after_change([&r](string s){r = s;});
  conns.push_back(move(c));
}
```

n	Lift(μs)	AdHoc(μs)	Diff	sd(μs)	sd(μs)
1	0,101	0,102	+0,01	0	0
10	1,032	1,033	+0,01	0,001	0,026
100	14,167	14,925	+0,05	0,006	0,101

AMD Ryzen 7 1700X with GCC 8.2.0 -O3 and 5x repetitions

RANGES OBSERVÁVEIS

PROBLEMA #2 - VISÃO EM ÁRVORE

```
struct city {
    string name;
};

struct country {
    string name;
    vector<city> cities;
};

vector<country> countries;
```

PROBLEMA #2 - VISÃO EM ÁRVORE

```cpp
struct city {
    string name;
};

struct country {
    string name;
    vector<city> cities;
};

vector<country> countries;
```

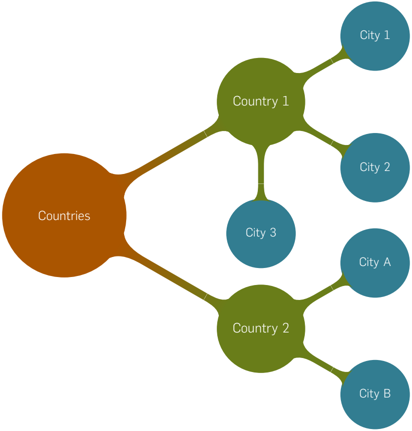
PROBLEMA #2 - VISÃO EM ÁRVORE

```
struct city {
    string name;
};

struct country {
    string name;
    vector<city> cities;
};

vector<country> countries;
```

PROBLEMA #2 - VISÃO EM ÁRVORE



- → Visão atualizada imediatamente quando o modelo é alterado
  - → Inserção/Remoção de nó
  - → Alteração do nome

PROBLEMA #2 - SOLUÇÃO AD HOC - COUNTRY

```
struct country {
  using cities_t = vector<city>;

  void push_back_city(const city& c) {
      _cities.push_back(c);
      _city_insert(_cities, prev(_cities.end()));
  }

  const cities_t& cities() const noexcept{ return _cities; };

  template<typename F>
  any_connection on_city_insert(F&& f)
  { return _city_insert.connect(forward<F>(f)); }
private:
  signal<void(const cities_t&, cities_t::iterator)>
    _city_insert;
  cities_t _cities;
};
```

PROBLEMA #2 - SOLUÇÃO AD HOC - COUNTRY

```cpp
struct country {
  using cities_t = vector<city>;

  void push_back_city(const city& c) {
      _cities.push_back(c);
      _city_insert(_cities, prev(_cities.end()));
  }

  const cities_t& cities() const noexcept{ return _cities; };

  template<typename F>
  any_connection on_city_insert(F&& f)
  { return _city_insert.connect(forward<F>(f)); }
private:
  signal<void(const cities_t&, cities_t::iterator)>
    _city_insert;
  cities_t _cities;
};
```

`push_back(city&&)`, `insert`, `emplace`, `emplace_back` e `std::sort()`?

PROBLEMA #2 - SOLUÇÃO AD HOC - COUNTRY

```
struct country {
  string name;
  using cities_t = vector<city>;

  void push_back_city(const city& c) {
      _cities.push_back(c);
      _city_insert(_cities, prev(_cities.end()));
  }

  cities_t::iterator erase(cities_t::iterator it) {
      _city_erase(_cities, it);
      return _cities.erase(it);
  }

  const cities_t& cities() const noexcept { return _cities; };

  template<typename F>
  any_connection on_city_insert(F&& f)
  { return _city_insert.connect(forward<F>(f)); }

  template<typename F>
  any_connection on_city_erase(F&& f)
  { return _city_erase.connect(forward<F>(f)); }
private:
  signal<void(cities_t&,cities_t::iterator)> _city_insert, _city_erase;
  cities_t _cities;
};
```

PROBLEMA #2 - SOLUÇÃO AD HOC - COUNTRIES

```
struct countries {
  using model_t = vector<country>;

  void push_back_country(const country& c) {
      _model.push_back(c);
      _country_insert(_model, prev(_model.end()));
  }

  model_t::iterator erase(model_t::iterator it) {
      _country_erase(_model, it);
      return _model.erase(it);
  }

  const model_t& countries() const noexcept { return _model; };

  template<typename F>
  any_connection on_country_insert(F&& f)
  { return _country_insert.connect(forward<F>(f)); }

  template<typename F>
  any_connection on_country_erase(F&& f)
  { return _country_erase.connect(forward<F>(f)); }
private:
  signal<void(model_t&, model_t::iterator)>
    _country_insert, _country_erase;
  model_t _model;
};
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        insert_row_child(st, it->name, gtk_it, pos);
      }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns,st](auto& countries, auto it) {
    auto pos = distance(countries.begin(),it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
    it->after_change(
      [gtk_it,st](string s){update_row(st, s, gtk_it);}));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it,st, &conns](auto& cities, auto it) {
        auto pos = distance(cities.begin(),it);
        auto gtk_it = insert_row_child(st, it->name, gtk_it,
            pos);
        conns.push_back(
        it->after_change(
          [gtk_it,st](string s){update_row(st, s, gtk_it);}));
        }));
  }));
for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
std::vector<any_connection> conns;
conns.push_back(
countries.on_country_insert(
  [&conns, st](auto& countries, auto it) {
    auto pos = distance(countries.begin(), it);
    auto gtk_it = insert_row(st, it->name, to_string(pos));
    conns.push_back(
      it->after_change(
      [gtk_it, st](string s){update_row(st, s, gtk_it);}));
    conns.push_back(
      it->on_city_insert.connect(
      [gtk_it, st,&conns](auto& cities, auto it) {
        auto pos = distance(cities.begin(), it);
        auto gtk_it = insert_row_child(st, it->name, gtk_it, pos);
        conns.push_back(
          it->after_change(
          [gtk_it, st](string s){update_row(st, s, gtk_it);}));
      }));
    conns.push_back(
      it->on_city_erase.connect(
        [&conns, st](auto& cities, auto it) {
          auto pos = distance(cities.begin(), it);
          remove_row(st, to_string(pos));
        }));
  }));

for(auto& c : conns) c.disconnect();
```

PROBLEMA #2 - SOLUÇÃO AD HOC - OBSERVAÇÃO

```
conns.push_back(
countries.on_country_erase(
  [&conns,st](auto& countries, auto it) {
    auto pos = distance(countries.begin(),it);
    remove_row(st,to_string(pos));
  }));
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
struct city {
    string name;
};

struct country {
    string name;
    coruja::vector<city> cities;
};

coruja::vector<country> countries;
```

RANGES

Range

Representa uma sequência de elementos através de [begin(), end())

```
countries | transform([](auto& c){return c.name;});
//{"Country", "Country2"}
```

PROBLEMA #2 - SOLUÇÃO CORUJA

ObservableErasableRange

Observable e Range permitindo a observação na inserção e remoção de elementos.

Requisitos

```
//FunctionObject: void(Rng&, Rng::iterator)
//  void(reference_t<Rng>)
template<typename FunctionObject>
for_each_connection_t for_each(FunctionObject)

//FunctionObject: void(Rng&, Rng::iterator)
//  void(reference_t<Rng>)
template<typename FunctionObject>
before_erase_connection_t before_erase(FunctionObject)

Observable::for_each_connection_t
Observable::before_erase_connection_t
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
template<typename ObservableErasableRange, typename F>
auto transform(ObservableErasableRange&&, F&&)

transform(countries, [](country& c){return c.name;});
//{"Country", "Country2"}
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
template<typename ObservableErasableRange, typename F>
auto transform(ObservableErasableRange&&, F&&)

transform(countries, [](country& c){return c.name;});
//{"Country", "Country2"}
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
//Rows is an ObservableErasableRange
//Rows::value_type should be Row:
//Row : string
//    | pair<string, Rows>
template<typename Rows>
struct tree_t {
    explicit tree_t(Rows) {/*impl*/}
    Rows rows;
};

//[("Country1", {"City1","City2","City3"}),
// ("Country2", {"CityA","CityB"})]

coruja::vector<string> v{"abc", "def"};
auto tree = make_tree(v);
v.emplace_back("ghi"); //update tree
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
//Rows is an ObservableErasableRange
//Rows::value_type should be Row:
//Row : string
//     | pair<string, Rows>
template<typename Rows>
struct tree_t {
    explicit tree_t(Rows) {/*impl*/}
    Rows rows;
};

//{("Country1", {"City1","City2","City3"}),
//  ("Country2", {"CityA","CityB"})}

coruja::vector<string> v{"abc", "def"};
auto tree = make_tree(v);
v.emplace_back("ghi"); //update tree
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
//Rows is an ObservableErasableRange
//Rows::value_type should be Row:
//Row : string
//     | pair<string, Rows>
template<typename Rows>
struct tree_t {
    explicit tree_t(Rows) {/*impl*/}
    Rows rows;
};

//[("Country1", {"City1","City2","City3"}),
// ("Country2", {"CityA","CityB"})]

coruja::vector<string> v{"abc", "def"};
auto tree = make_tree(v);
v.emplace_back("ghi"); //update tree
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
//Rows is an ObservableErasableRange
//Rows::value_type should be Row:
//Row : string
//    | pair<string, Rows>
template<typename Rows>
struct tree_t {
    explicit tree_t(Rows) {/*impl*/}
    Rows rows;
};

coruja::vector<coruja::object<string>> v;
auto tree = make_tree(v);
v.emplace_back("ghi"); //update tree
v.back() = "change"; //update tree
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
//Rows is an ObservableErasableRange
//Rows::value_type should be Row:
//Row : string
//      | pair<string, Rows>
template<typename Rows>
struct tree_t {
    explicit tree_t(Rows) {/*impl*/}
    Rows rows;
};

coruja::vector<coruja::object<string>> v;
auto tree = make_tree(v);
v.emplace_back("ghi"); //update tree
v.back() = "change"; //update tree
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
struct city { string name; };
struct country {
    string name;
    coruja::vector<city> cities;
};
coruja::vector<country> countries;

auto rows = transform(countries,
  [](country& c){
    return row(c.name, transform(c.cities,
               [](city& c){return c.name;})); });

auto tree = make_tree(rows);
//[("Country1", {"City1","City2","City3"}),
// ("Country2", {"CityA","CityB"})]
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
struct city { string name; };
struct country {
    string name;
    coruja::vector<city> cities;
};
coruja::vector<country> countries;

auto rows = transform(countries,
  [](country& c){
    return row(c.name, transform(c.cities,
                [](city& c){return c.name;})); });

auto tree = make_tree(rows);
//[("Country1", {"City1","City2","City3"}),
// ("Country2", {"CityA","CityB"})]
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
template<typename F>
class invoke_observer_impl : private invoke_observer_base<F> {
  using base = invoke_observer_base<F>;
  using base::base;

  template<typename From, typename It>
  void operator()(From& from, It it) {
    using namespace ranges;
    auto rng = coruja_transform_view{from, base::as_transform()};
    base::_f(rng, next(begin(rng), distance(begin(from), it)));
  }
};
```

PROBLEMA #2 - SOLUÇÃO CORUJA

```
template<typename F>
class invoke_observer_impl : private invoke_observer_base<F> {
  using base = invoke_observer_base<F>;
  using base::base;

  template<typename From, typename It>
  void operator()(From& from, It it) {
    using namespace ranges;
    auto rng = coruja_transform_view{from, base::as_transform()};
    base::_f(rng, next(begin(rng), distance(begin(from), it)));
  }
};
```

PUSH_BACK() - ADHOC X TRANSFORM

push_back()	Transform(μs)	AdHoc(μs)	Diff(μs)	sd(μs)	sd(μs)
50	83	83	0	4	4
200	435	423	−12	20	20
500	2000	2019	+19	24	32
1000	7398	7476	+78	203	249
1500	16206	16301	+95	352	389
2000	28043	28310	+267	326	441
5000	188646	191184	+2538	2408	578
10000	841242	854977	+13735	10371	9148

AMD Ryzen 7 1700X with GCC 4.8.2 -O3, GTK 2.24/GtkTreeStore and 10-250x repetitions

PUSH_BACK() - STD::VECTOR X CORUJA X BOOST.SIGNALS2

```
// wosignals
using vec_t = std::vector<size_t>;

// csignals(Coruja signal)
using vec_t = vector<size_t>;

// bsignals(Boost.Signals2)
using vec_t = vector<size_t, allocator<size_t>,
                     std::vector, void, boost_signals2>;
```

n	vector(ns)	coruja(ns)	boost(ns)	diff	sd(ns)	sd(ns)	sd(ns)
5	26,84	28,76	223,96	+6,79	0,10	0,32	14,36
1000	2083,38	2160,98	38497,10	+16,81	1,38	0,37	48,37
10000	20594,15	21388,84	388575,65	+17,17	0,78	6,27	3929,02

AMD Ryzen 7 1700X with GCC 8.2.0 -O3, Boost 1.67.0 and 5 repetitions

SIGNAL - CORUJA X BOOST.SIGNALS2 - COMPILE TIME

```
//Coruja signal
#include <coruja/support/signal.hpp>
coruja::signal<void(int)> sig;

//Boost.Signals2
#include <boost/signals2/signal.hpp>
boost::signals2::signal<void(int)> sig;
```

Coruja(s)	Boost.Signals2(s)	Diff	sd(s)	sd(s)
0.286	1.298	+3.54	0.005	0.004

AMD Ryzen 7 1700X with GCC 8.2.0 -O3, Boost 1.67.0 and 5 repetitions

OUTROS MODELOS

→ coruja::map, coruja::unordered_map e coruja::flat_map

→ coruja::set e coruja::flat_set

→ coruja::optional

→ coruja::variant

→ coruja::object_view e coruja::container_view

→ coruja::any_object_view

SUPORTE BOOST.SERIALIZATION

```
template<typename Archive>
void serialize(Archive& ar, country& o, unsigned int) {
  ar & o.name;
  ar & o.cities; // coruja::vector<city>
}
```

→ Operação de `load` notifica observadores

TRABALHOS RELACIONADOS

- RxCpp
  - Abstrações baseadas em FRP (streams e operations)
  - Notificações assíncronas
  - ‘rx::iterate‘ (Observables a partir de ranges)
    - Considera o range como imutável.
- Sodium e sfrp
  - Implementações puras de FRP (stream, cell e operations)

CONTINUAÇÃO

→ Revisitar classes reativas: `coruja::reactive_class`

→ Lançar versão 0.1

→ Revisitar suporte a FRP (streams e operations)

→ Melhorar o uso de concepts em compile time

→ Explorar widgets com suporte a Observables

OBRIGADO

github.com/ricardocosme/coruja

- → Benchmarks: `test/bench*.cpp`
- → Demos: `demo/{fullname,fullnames,hello}.cpp`