Парсим и кодогенерируем с использованием clang

Антон Наумович
Юрий Ефимочев

LOGICnow™

О нас

Разрабатываем бэкап-решение IASO MAXBackup™

Антон Наумович

Тимлид в LogicNow

В прошлом: разработчик в Microsoft (Hyper-V)

Специализация: производительность, отладка, дизайн

Юрий Ефимочев

Архитектор в LogicNow

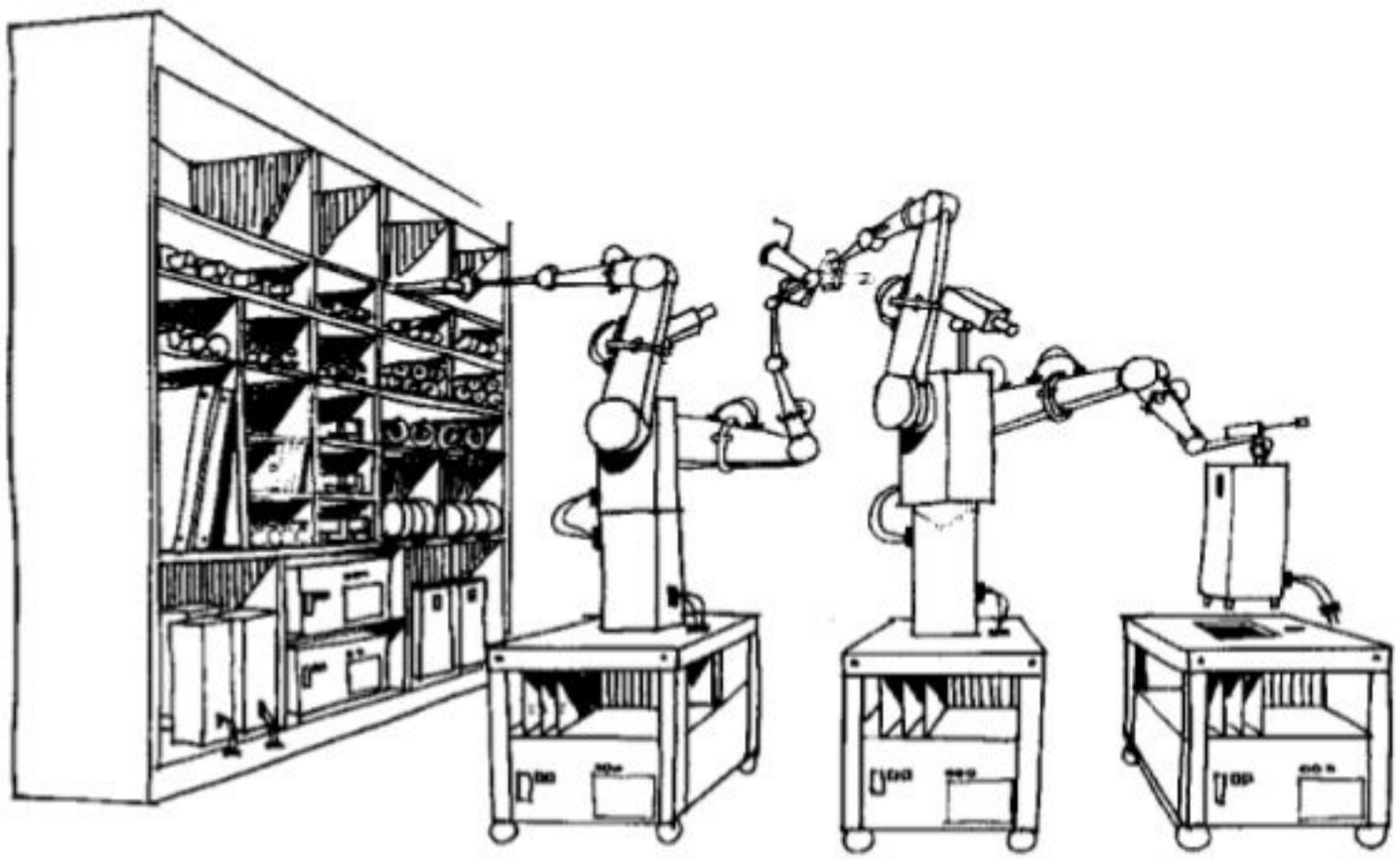
Специализация: высоконагруженные отказоустойчивые системы на C++

COREHARD

Кодогенерация

Классическое разделение

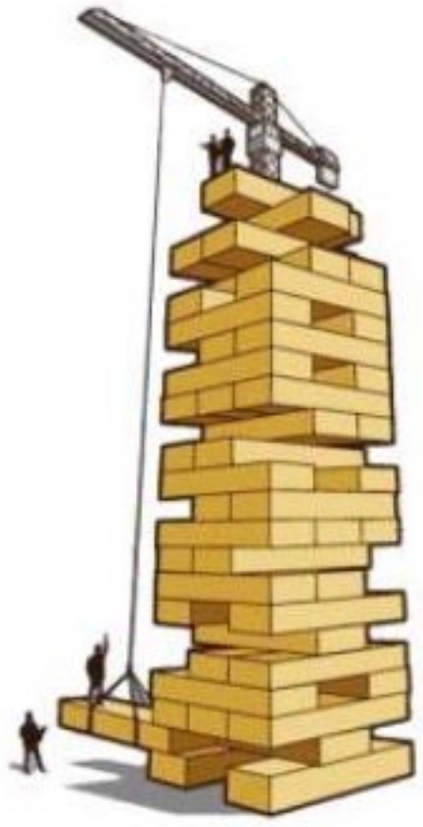
1. Пассивная – разовая, с ручными правками
2. **Активная** – автоматическая, регулярная, без ручных правок



Частые релизы проекта

Вызовы

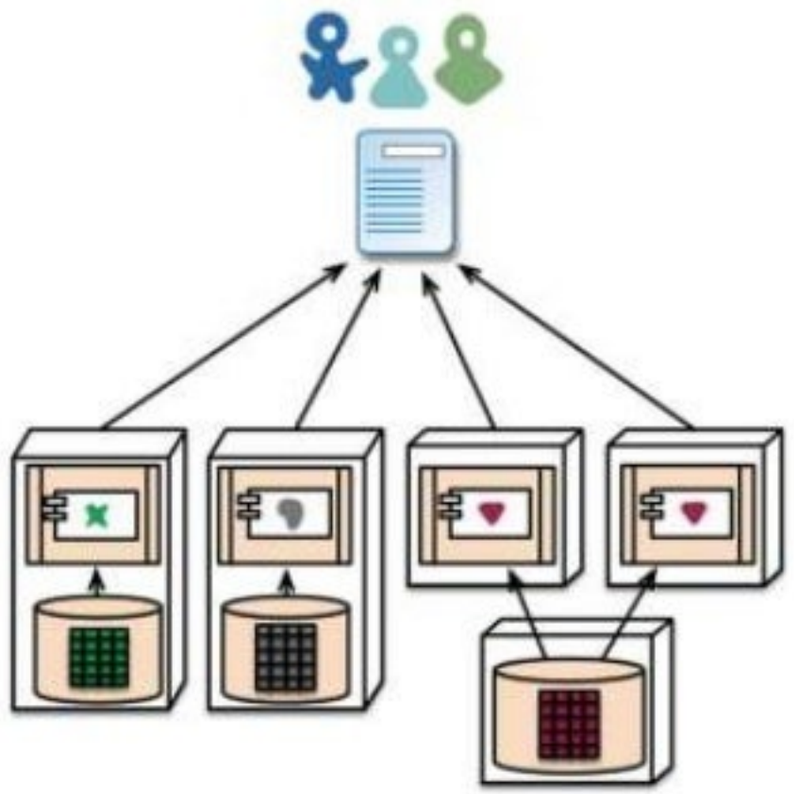
- быстрая реакция на изменение требований
- минимизация человеческих ошибок
- высокое покрытие тестами



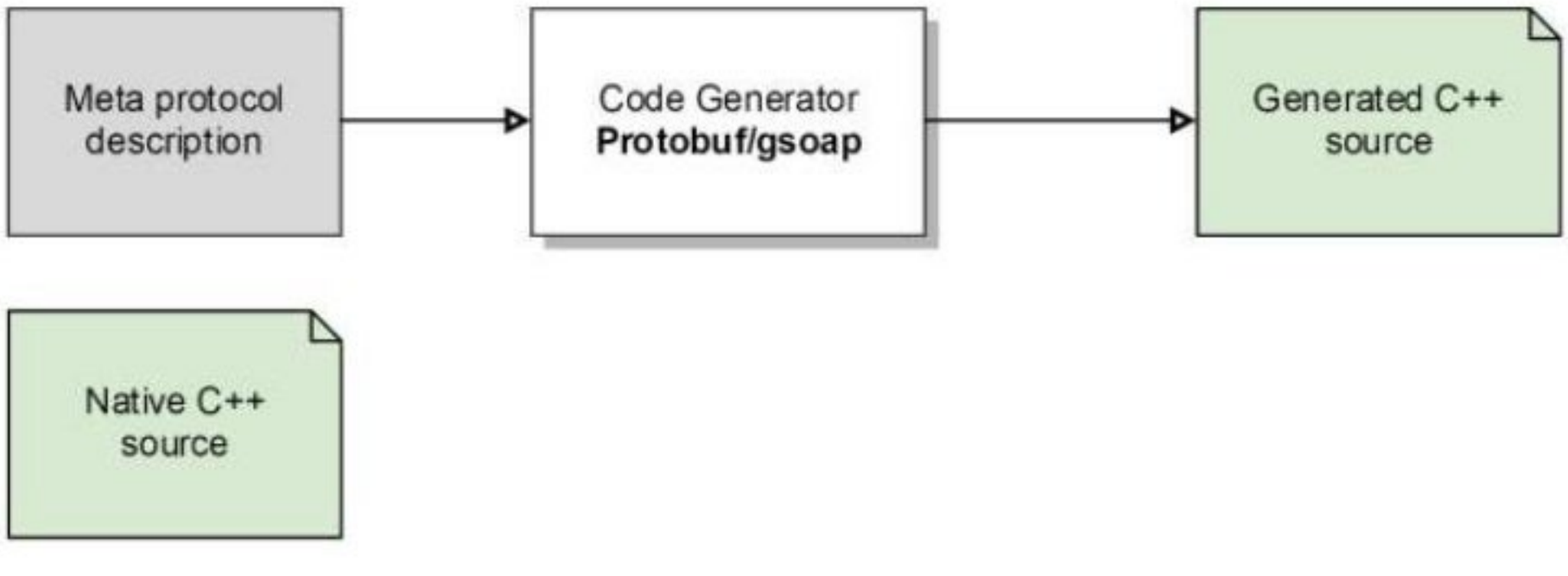
Microservice-архитектура

Рутинные задачи

- создание сетевых протоколов
- создание слоя хранения данных
- тесты, тесты, тесты



Как генерируют протоколы



Минусы

- нет контроля над процессом генерации
- генерированные исходники “чужеродны” для проекта
- дублирование кода (“родные” сущности сосуществуют с генерированными)

COREHARD

Protobuf: пример

```
// Types.proto
message CustomerInfo
{
    required int32 id = 1;
    required string name = 2;
    required string email = 3;
}
```

⇓

```
class CustomerInfo : public ::google::protobuf::Message {
 public:

static const ::google::protobuf::Descriptor* descriptor();
static const CustomerInfo& default_instance();
void Swap(CustomerInfo* other);

// implements Message ----------------------------------------------

CustomerInfo* New() const;
void CopyFrom(const ::google::protobuf::Message& from);
void MergeFrom(const ::google::protobuf::Message& from);
void CopyFrom(const CustomerInfo& from);
void MergeFrom(const CustomerInfo& from);
void Clear();
bool IsInitialized() const;
```

... только декларация 113 строк кода!

gsoap: пример

```
// Types.gsoap.h
class CustomerInfo
{
public:
    int id 1:;
    string name 0:;
    string email 0:;
};
```

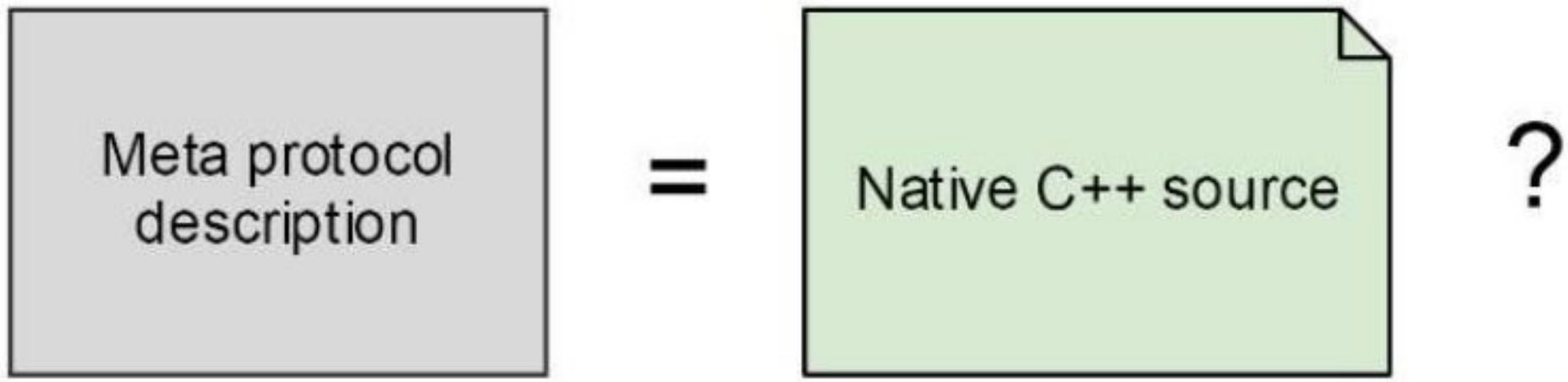
⇓

```
class SOAP_CMAC CustomerInfo
{
public:
        int id;
        std::string *name;
        std::string *emailAddress;
public:
        virtual int soap_type() const { return 38; } /* = unique id
SOAP_TYPE_CustomerInfo */
        virtual void soap_default(struct soap*);
        virtual void soap_serialize(struct soap*) const;
        virtual int soap_put(struct soap*, const char*, const char*) const;
        virtual int soap_out(struct soap*, const char*, int, const char*) const;
        virtual void *soap_get(struct soap*, const char*, const char*);
```

...

И это только декларации, сериализация еще похлеще :)

Single Source of Truth?



Single Source of Truth!

Сущности

```cpp
// CustomerInfo.h
struct CustomerInfo
{
    int Id;
    string Name;
    CustomerType Type;
};
```

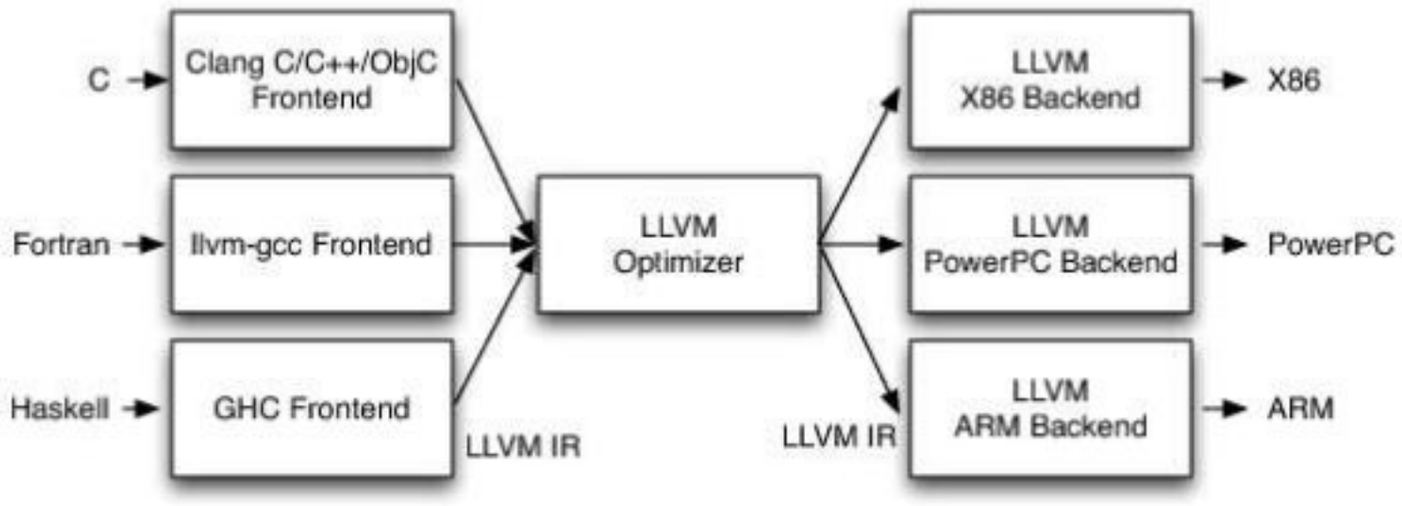
Интерфейсы

```cpp
// ICustomerManager.h
class ICustomerManager
{
    virtual int CreateCustomer(CustomerInfo const& customer) = 0;
    virtual void UpdateCustomer(CustomerInfo const& customer) = 0;
    virtual void DeleteCustomer(int customerId) = 0;
    virtual CustomerInfo GetCustomer(int customerId) const = 0;
    virtual ICustomerInfoIteratorPtr EnumerateCustomers() const = 0;

    virtual ~ICustomerManager() {}
};
```

Декларации на C++ и есть самодостаточное базовое мета-описание протокола

COREHARD

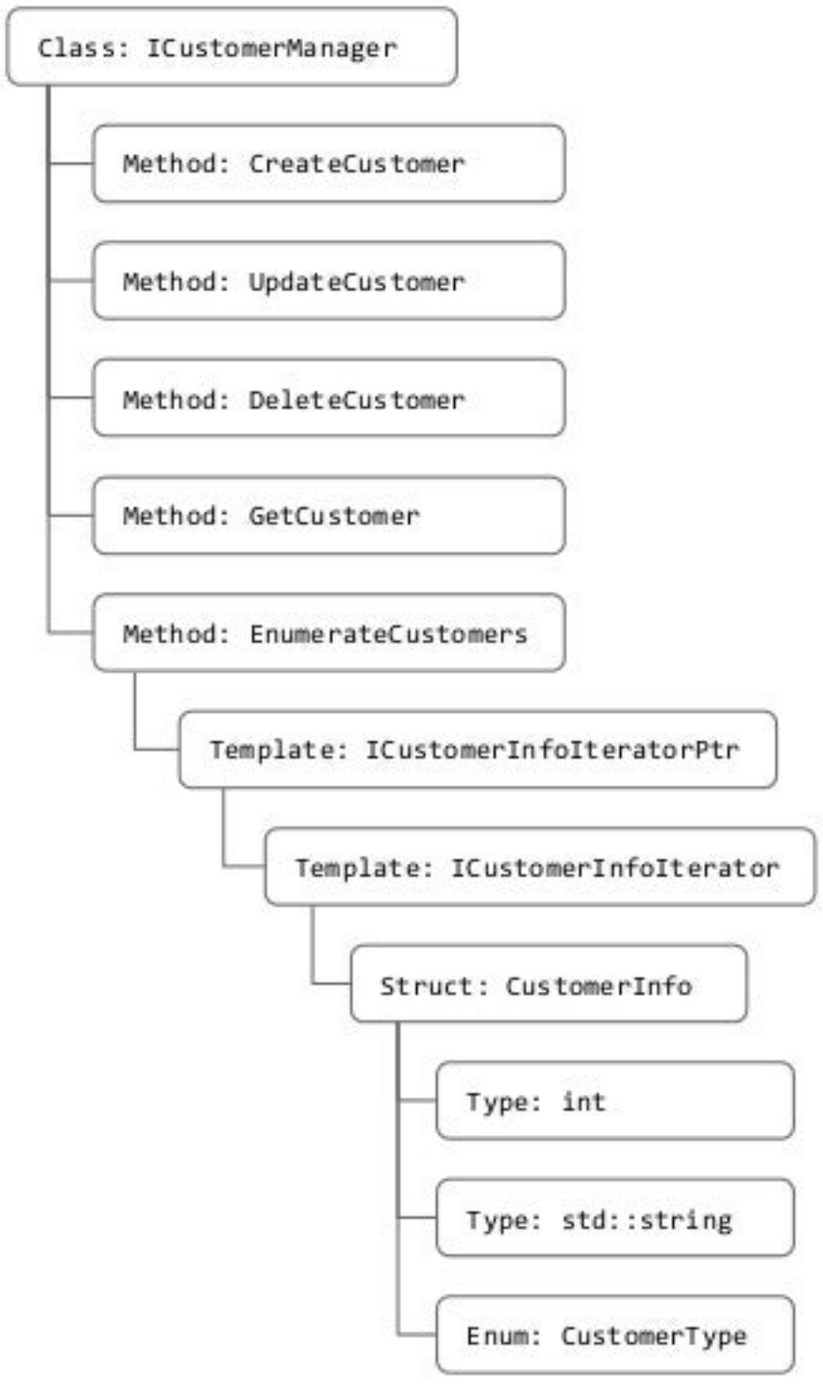
Что возможно с clang?



Инструменты на основе clang:

- clang-format
- clang-check
- clang-tidy
- статический анализ, индексирование кода, подсветка синтаксиса

ClangTool: парсер C++ деклараций



Вход ClangTool:

```
struct CustomerInfo
 {
     int Id;
     std::string Name;
     CustomerType Type;
 };

 typedef IIterator<CustomerInfo> ICustomerInfoIterator;
 typedef std::unique_ptr<ICustomerInfoIterator>
     ICustomerInfoIteratorPtr;

class ICustomerManager
{
public:
    virtual int CreateCustomer(CustomerInfo const& curstomer) = 0;
    virtual void UpdateCustomer(CustomerInfo const& customer) = 0;
    virtual void DeleteCustomer(int customerId) = 0;
    virtual CustomerInfo GetCustomer(int customerId) const = 0;
    virtual ICustomerInfoIteratorPtr EnumerateCustomers() const = 0;

    virtual ~ICustomerManager() {}
};
```

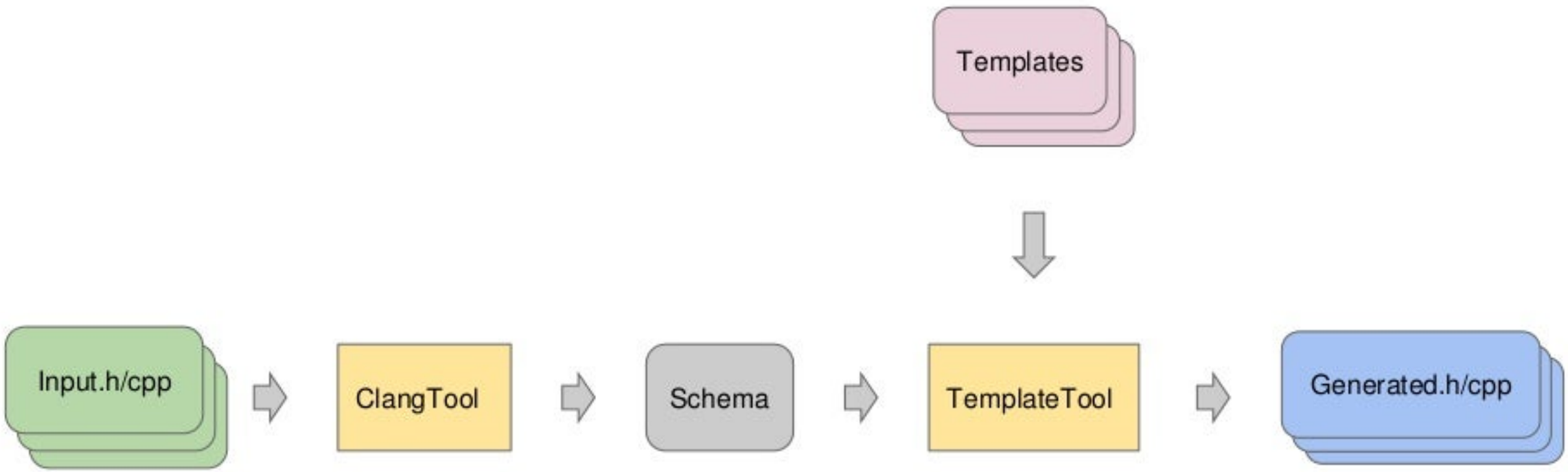
Реализация: ~500 строк кода

COREHARD

ClangTool: результат (schema)

```
{
    "Interface" : "ICustomerManager",
    "Enums" :
    [
        { "Name" : "CustomerType", "Values" : [ "Undefined", "Managed", "Unmanaged" ] }
    ],
    "Structs" :
    [
        {
            "Name" : "CustomerInfo",
            "Fields" :
            [
                { "Name" : "Id", "Type" : "int" },
                { "Name" : "Name", "Type" : "std::string" },
                { "Name" : "Type", "Type" : "CustomerType::Enum" }
            ]
        }
    ],
    "Methods" :
    [
        {
            "Name" : "AddCustomer",
            "Arguments" : [ { "Name" : "CustomerInfo", "Type" : "CustomerInfo" } ],
            "ReturnType" : "void"
        },
            ...

        {
            "Name" : "EnumerateCustomers",
            "Arguments" : [],
            "ReturnType" : "ICustomerInfoIteratorPtr"
        }
    ]
}
```

Схема кодогенерации



Шаблоны кодогенерации пишутся разово под класс задач

COREHARD

Реализация: TemplateTool

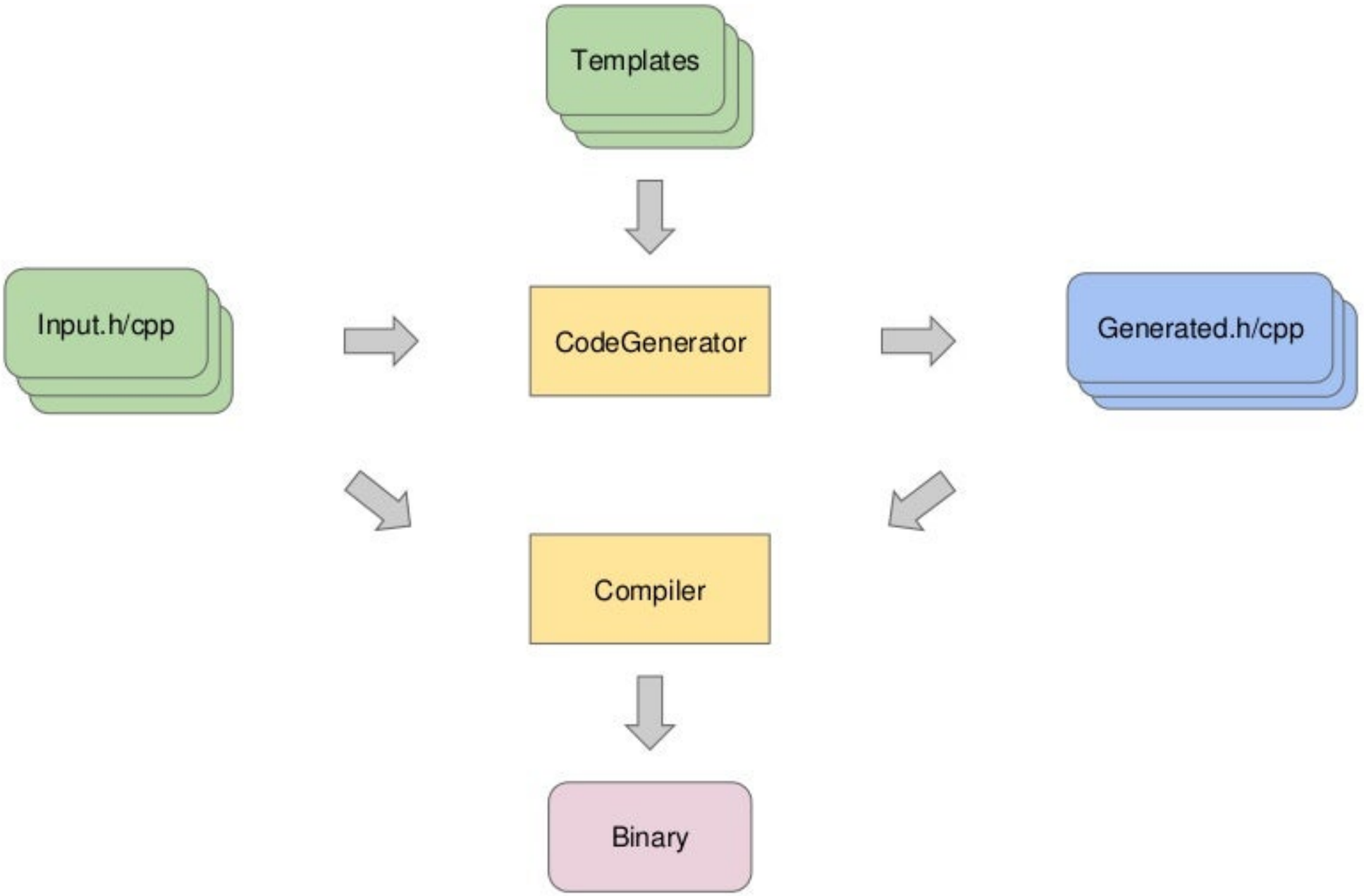
TemplateTool – кодогенератор на основе шаблонов

```
// StructSerialization.gen
void CppToJson(<%struct.Name%> const& native, Json::Value& json)
{
        json["typename"] = "<%struct.Name%>";

        <%foreach field in struct.Fields%>
        CppToJson(native.<%field.Name%>, "<%field.Name%>", json);
        <%end%>
}
```

⇓

```
// CustomerInfoSerialization.cpp
void CppToJson(CustomerInfo const& native, Json::Value& json)
{
        json["typename"] = "CustomerInfo";

        CppToJson(native.Id, "Id", json);
        CppToJson(native.Name, "Name", json);
        CppToJson(native.Type, "Type", json);
}
```

Общая схема компиляции



Code Generator = **ClangTool** + **TemplateTool**

Конфигурация кодогенератора (CMake)

```
JSONAPI_BEGIN(ManagementApi)

...

JSONAPI_ADD_INTERFACE
(
    NAME
        CustomerManager

    INTERFACE
        ICustomerManager

    INTERFACE_HEADER
        Interface/ICustomerManager.h

    INCLUDES
        Interface/CustomerInfo.h
        Core/Iterators/IIterator.h
)

...

JSONAPI_END()
```

Пример использования протокола

```cpp
// Server.cpp

#inculude "Management/CustomerManager.h"
#inculude "ManagementApi/Generated/CustomerManagerDispatcher.h"

int main()
{
    CustomerManager customerManager;
    JsonApiService apiService;
    apiService.Register<CustomerManagerDispatcher>(customerManager);
    apiService.Start(ports);
}
```

```cpp
// Client.cpp

#include "ManagementApi/Generated/CustomerManagerClient.h"

int main()
{
    JsonApiClient apiClient("https://domain/jsonapi");
    ICustomerManagerPtr customerManager(new CustomerManagerClient(apiClient));
    CustomerInfo const info = customerManager->GetCustomer(42);
}
```

Базы данных

```
// CustomerInfo.h
struct CustomerInfo
{
    int Id;
    CustomerType Type;
    string Name;
};
```

уже описанным способом

⇒

```
// CustomerInfo.ddl
CREATE TABLE CustomerInfo
(
    Id INT(4) NOT NULL,
    CustomerType  INT(4),
    Name CHAR(20)
)
```

1. Преобразуем C++ декларации в SQL (DDL)
2. Генерируем также и Object-Relational Mapping слой

COREHARD

Базы данных: шаг дальше

Если недостаточно синтаксиса по умолчанию

C++ 98/2003 (комментарии)

```
// CustomerInfo.h
struct CustomerInfo
{
    int Id;
    CustomerType Type; // FK: CustomerType.Id
    string Name;
};
```

⇒

```
// CustomerInfo.ddl
CREATE TABLE CustomerInfo
(
    Id INT(4) NOT NULL,
    CustomerType  INT(4) NOT NULL
REFERENCES CustomerType(Id),
    Name CHAR(20),
    KEY CustomerType (CustomerType)
)
```

C++ 11/14 (атрибуты)

```
// CustomerInfo.h
struct CustomerInfo
{
    int Id;
    [[FK: CustomerType.Id]]
    CustomerType Type;
    string Name;
};
```

Что еще пригодно для генерации?

- Типовые юнит-тесты (для протоколов, баз данных)
- **Клиенты для протоколов** (на любых языках!) - к примеру, для авто-тестов:

```
<%foreach method in Methods%>
@step(log_output=False)
def <%method.Name%>(<%foreach param in method.InputParams%><%param.Name%>, <%end%>):
    return __send_post_request('<%method.Name%>'
        <%foreach param in method.InputParams%>,<%param.Name%>=<%param.Name%><%end%>)
<%end%>

<%foreach struct in Structs%>
class <%struct.Name%>(JSONStruct):
    def __init__(self<%foreach field in struct.Fields%>, <%field.Name%>=None<%end%>):
<%foreach field in struct.Fields%>
        self.<%field.Name%> = <%field.Name%>
<%end%>
    def __repr__(self):
        return str(serialize(self))
<%end%>
```

- клиент на Python

COREHARD

Генерация дизайн паттернов *

Задача:

- Есть семейство типовых интерфейсов
- Требуется всему семейству добавить поведение (например, права доступа, потокобезопасность и т.д.)

```
class ThreadSafe<%persistencyName%>Persistency :
     public I<%persistencyName%>Persistency
{
public:
    ThreadSafe<%persistencyName%>Persistency(I<%persistencyName%>PersistencyPtr
decoratee, std::mutex& mutex);

private:
    <%foreach method in Methods%>
    virtual <%method.ReturnType%> <%method.Name%>(<%foreach param in
method.Params%><%if !param.IsFirst%>, <%end%><%param.ExactType%>
<%param.Name%><%end%>)<%if method.IsConstant%> const<%end%>;
     <%end%>

private:
    I<%persistencyName%>PersistencyPtr m_decoratee;
    std::mutex& m_mutex;
};
```

Итого

Выгоды

- Устранение рутинной работы
- Минимизация человеческих ошибок
- Решение типового набора задач “за бесплатно”
- Более высокий уровень абстракции

Проблемы

- Версионность (реакция на изменения кода)
- Сопряжение с рукописным кодом
- Сложность конфигурации генератора

COREHARD

Спасибо! Вопросы?

Антон Наумович
Anton.Naumovich@LogicNow.com

Юрий Ефимочев
Yury.Efimochev@LogicNow.com

LOGICnow™

COREHARD