СРЕДСТВО ГЕНЕРАЦИИ КОДА ИМИТАЦИОННОЙ МОДЕЛИ, СОВМЕСТИМОЙ СО СТАНДАРТОМ HLA

В. А. Антоненко, Д. Ю. Волканов, М. В. Чистолинов (Москва)

В настоящее время разработан целый ряд средств и языков моделирования. Основным стандартом в области специализированных средств моделирования распределённых систем является High Level Architecture (HLA) (IEEE 1516, 1516.2010) [1].

Одним из главных компонентов системы моделирования, описанным в стандарте HLA, является Runtime Infrastructure (RTI). Под RTI понимают среду передачи данных между компонентами, входящими в модель. Исходя из стандарта HLA, необходимо для каждой имитационной модели отдельно описать несколько десятков интерфейсов для взаимодействия с RTI. Следовательно, помимо затрат на построение самой модели, разработчику требуется дополнительно заниматься описанием интерфейсов для взаимодействия с RTI. В результате актуальна задача создания среды разработки HLA-совместимых моделей, в которой построение моделей осуществлялось бы при помощи текстового или графического языка описания моделей.

Требования к средству разработки модели

Основная цель разрабатываемого средства – абстрагирование пользователя (разработчика имитационных моделей) от описания интерфейсов для взаимодействия с RTI. При этом необходимо придерживаться подхода с использованием графической оболочки для построения модели, так как разработчик модели не обязан быть программистом. Исходя из этого, список требований выглядит следующим образом:

• *Графический интерфейс.* Наличие графической оболочки для построения имитационных моделей.

• *Создание HLA-совместимых моделей.* Возможность получения на выходе исходных кодов моделей с описанным интерфейсом для подключения к RTI.

• *Генерация кода по шаблонам.* Возможность генерации исходного кода модели (на C++) по представленной диаграмме состояний.

• *Свободная лицензия*. Необходимо, чтобы базовое средство распространялось по GPL лицензии.

• *Возможность верификации.* Следует проверять заданные функциональные свойства разрабатываемой имитационной модели, например, с использованием библиотеки UPPAL.

Существующие средства описания имитационных моделей

При создании средств описания имитационных моделей разработчики используют один из следующих подходов [2]:

• Создание языка с функциями поддержки моделирования.

• Написание специализированных модулей (библиотек классов), решающих задачи поддержки моделирования, для языка общего назначения.

• Создание графического интерфейса для описания моделей.

В первом случае создается (как правило, на основе синтаксиса существующего языка) новый язык, в котором разработчиками предусматриваются специальные конструкции, предназначенные для решения задач имитационного моделирования. Данные конструкции могут использоваться для управления модельным временем, для описания взаимодействия компонентов модели, функции, позволяющие записывать изменяемые параметры в трассу эксперимента. Примером такого языка моделирования является ММ-язык, используемый в среде моделирования DYANA [3].

При использовании второго подхода описанные выше функции реализуются в виде модуля языка общего назначения, например библиотеки классов и шаблонов C++ или Java. Примером такого языка моделирования является SLX [4].

Применение уже существующего языка позволяет упростить разработку моделей благодаря тому, что не требуется изучение разработчиками нового языка и возможно использование существующих программных средств (таких как среда разработки, транслятор, отладчик) и существующих библиотек программных компонентов.

Современные тенденции развития языков моделирования [5, 6] заключаются в уменьшении роли текстовых специализированных языков описания моделей, сведения их к использованию только для внутреннего представления. На практике же разработчики специализированных языков предоставляют графический интерфейс описания моделей (например, в виде диаграмм классов).

Из анализа открытой литературы было выделено несколько основных языков имитационного моделирования, доминирующих на рынке специализированного ПО для построения моделей, а так же проанализированы возможности языка UML. Рассмотрим особенности этих языков.

Modelica – свободно распространяемый объектно-ориентированный язык для моделирования сложных физических систем [7]. Язык имеет хорошую техническую поддержку разработчиков, для него существует большое количество библиотек компонентов. Modelica обеспечивает создание различных моделей: механических, электрических, гидравлических, химических.

В основе языка Modelica лежит концепция соединяемых блоков. При соединении в соответствии с требуемой схемой автоматически генерируются соответствующие уравнения. Это делает язык простым для понимания и использования специалистами нематематического профиля.

SLX. Язык SLX [4] основывается на широких возможностях языка имитационного моделирования GPSS/H. Обеспечивает большие возможности для моделирования современных систем, описываемых языками, похожими на C.

Основные особенности SLX позволяют создавать разнообразную логику, петли управления, расширения, диагностики. В дополнение к новым директивам SLX поддерживает традиционные макросы и модули расширения, применимые как при компиляции, так и в процессе исполнения.

Simulink – это интерактивная графическая система для анализа линейных и нелинейных динамических систем. Она позволяет моделировать систему простым перемещением блоков в рабочую область и последующей установкой их параметров. Simulink может работать с линейными, нелинейными, непрерывными, дискретными, многомерными системами [8].

Наиболее интересным с точки зрения моделирования встроенных систем является пакет Stateflow Simulink [9]. Stateflow – это интерактивный инструмент разработки в области моделирования сложных управляемых событиями систем. Он основан на теории конечных автоматов. Stateflow предлагает решение для проектирования встроенных систем с контролирующей логикой. Для описания логики модели Stateflow использует диаграммы состояний и переходов.

Диаграммы состояний языка UML являются хорошо известным средством описания поведения систем. Они определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате влияния некоторых событий. Из конкретного состояния в данный момент времени может быть осуществлен только один переход; таким образом, условия являются взаимно исключающими для любого события. Существует два особых состояния: вход и выход. Любое действие, связанное с событием входа, выполняется, когда объект вхо-

дит в данное состояние. Событие выхода выполняется в том случае, когда объект выходит из данного состояния.

Данные языки моделирования были проанализированы на предмет соответствия поставленным требованиям. Результаты представлены в табл. 1.

Таблица 1

Отображение UML примитивов на XML объекты и шаблоны кода

Критерий	Modelica	SLX	Simulink Stateflow	UML
Графический интерфейс	+	-	+	+
Работа диаграммами состояний	+	-	+	+
Создание HLA совместимых моделей	-	-	-	-
Генерация кода по шаблонам	+	-	+	-
Свободная лицензия	-	+	-	+
Возможность верификации	+	-	+	+ (*)

Анализ показал несоответствие специализированных языков и сред разработки имитационных моделей предъявленным требованиям. Следовательно, необходимо разработать собственную систему построения имитационных моделей и генерации исходных кодов моделей, совместимых со стандартом HLA. Языки моделирования Modelica, Simulink Stateflow и UML удовлетворяют наибольшему количеству требований, но свободной лицензией из них обладает только язык моделирования UML. Далее описано средство для генерации кода имитационных моделей на основании диаграмм состояний UML.

Генерация кода моделей по диаграммам состояний UML

Средство для генерации кода имитационных моделей на основании диаграмм состояний UML (Генератор) было реализовано на языке Питон. Для создания и редактирования диаграмм состояний UML использовалось средство ArgoUML [10]. Однако генератор не работает напрямую с диаграммой UML, а только с её XML представлением. В качестве XML представления был выбран специализированный XML формат, предназначенный для работы с диаграммами состояний – State Chart XML [11]. Данный формат позволяет описывать конечные автоматы в общем виде на основе диаграмм состояний Харела. Используя SCXML, можно описать различные типы структур конечных автоматов. В качестве примера можно привести такие случаи, как вложенность, параллельность, синхронизация или параллельность подавтоматов.

Для работы с шаблонами генерации кода использовалась специализированная библиотека шаблонов Cheetah [12]. Суть работы данной библиотеки следующая: шаблон компилируется с помощью cheetah-компилятора в представление шаблонов на языке Питон. Данный файл используется загрузчиком питона для генерации выходного файла. Шаблон Cheetah состоит из комбинации выходного текста и кода, похожего на исходный код языка Питон.

На вход генератору подается диаграмма состояний языка UML и шаблон для генерации кода. Далее диаграмма переводится во внутренние представление языка Питон. После создания SCXML файла, он подаётся на вход непосредственно генератору кода по шаблону, который генерирует **.h .cpp .fed** файлы. Для генерации кода федератов (с правильными интерфейсами для подключения к RTI) были созданы особые шаблоны: отдельно для **.h, .cpp** и **.fed** файлов. На рис. 1 представлена схема работы генератора кода моделей совместимых со стандартом HLA.

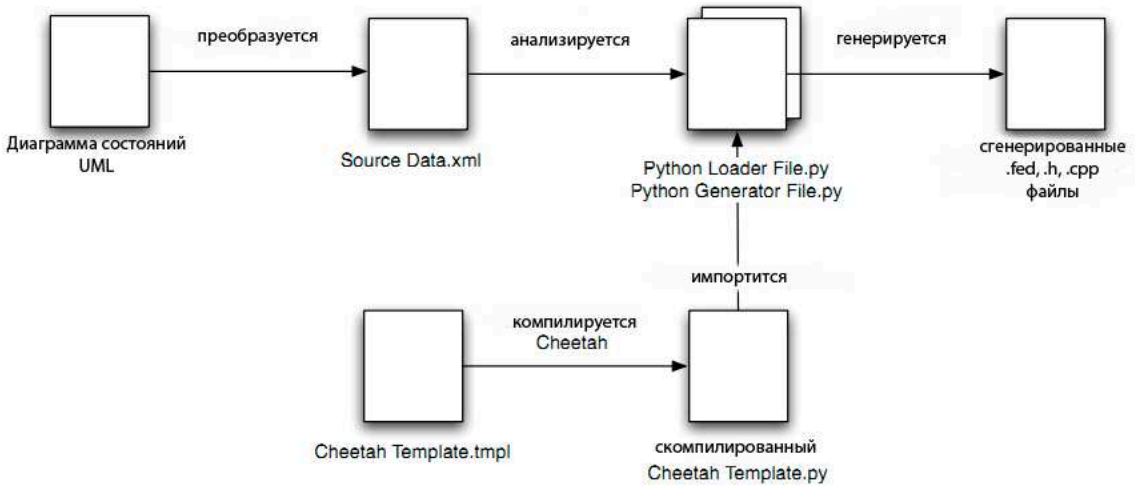

Рис. 1. Схема работы генератора кода

В результате на выходе получены исходные файлы моделей с интерфейсами для подключения к RTI. Сравнение сгенерированного и вручную написанного кода представлено на рис. 2 и 3.

```
#ifndef SM_STATE_FED_ARMED_H
#define SM_STATE_FED_ARMED_H

#include "sm_state.h"
#include "federate.h"

class SM_State_FED_Armed : public SM_State, Federate
{
public:

  //-------------------------------------------------------------------------
  SM_State_FED_Armed (std::wstring _federationName, std::wstring _fddName);

  //-------------------------------------------------------------------------
  ~SM_State_FED_Armed ();

  //-------------------------------------------------------------------------


  //-------------------------------------------------------------------------

  void onentry ();

protected:

private:
  //-------------------------------------------------------------------------
        void LockDoors();

  //-------------------------------------------------------------------------
  // copy constructor not implemented
  SM_State_FED_Armed( const SM_State_FED_Armed& );

  // assignment operator not implemented
  SM_State_FED_Armed& operator=( const SM_State_FED_Armed& );


  //-------------------------------------------------------------------------

  void getHandles();
  void customInit();
  void loop();


  //-------------------------------------------------------------------------
}; // SM_State_FED_Armed
```

Рис. 2. Пример сгенерированного header файла

```
//-------------------------------------------------------------------------

#include "sm_state_FED_Armed.h"     // class SM_State_FED_Armed


#include <iostream>


//-------------------------------------------------------------------------
//
// Class Name : SM_State_FED_Armed
//
//-------------------------------------------------------------------------

SM_State_FED_Armed::SM_State_FED_Armed ()
{}

//-------------------------------------------------------------------------
SM_State_FED_Armed::~SM_State_FED_Armed ()
{}

//-------------------------------------------------------------------------
void SM_State_FED_Armed::onentry ()
{
  LockDoors();
}

//-------------------------------------------------------------------------
void SM_State_FED_Armed::LockDoors ()
{

  std::cout << "SM_State_FED_Armed::LockDoors\n";


}

//-------------------------------------------------------------------------
```

Рис. 3. Пример сгенерированного cpp файла

Заключение

Обзор специализированных языков моделирования показал, что рассмотренные языки моделирования не обладают необходимой функциональностью для удобного описания HLA-совместимых моделей. Поэтому был сделан выбор в пользу создания специализированного средства для построения HLA-совместимых моделей с использованием в качестве языка моделирования диаграммы состояний UML. На основании

графического описания, дополненного кодом на C++, генерируется код HLA-совместимой модели на языке C++.

Было разработано средство для генерации кода HLA-совместимых моделей по шаблонам. Экспериментальные исследования показали работоспособность генерируемого кода в разрабатываемой системе имитационного моделирования. Реализованное средство позволяет сократить время разработки имитационной модели и будет входить в разрабатываемый комплекс полунатурного моделирования РВС РВ.

Литература

1. Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification. 2000.
2. **Manuel Alfonseca, Juan de Lara, Estrella Pulido**. An object-oriented continuous simulation language and its use for training purposes. (http://www.ii.uam.es/~alfonsec/docs/simul98f.htm)
3. **Smeliansky R. L., Bakhmurov A. G., Kostenko V. A.** DYANA – An Environment For Simulation And Analysis Of Distributed Multiprocessor Computer Systems // Proceedings of TACAS'99, 1999.
4. The SLX-HLA-Interface Homepage [HTML] (http://www.strassburger-online.de/SLX.html)
5. **Rajesh Mansharamani**. An overview of discrete event simulation methodologies and implementation // SADHANA. 1997. Vol. 22. No 5. P. 611–627.
6. **Nance R. E.** A history of discrete event simulation programming languages. Blacksburg, United States: Virginia Polytechnic Institute and State University, 1993.
7. Modelica – A Unified Object-Oriented Language for Physical Systems Modeling Tutorial [PDF] (https://www.modelica.org/documents/ModelicaTutorial14.pdf)
8. Simulink Tutorial [PDF] (http://academic.csuohio.edu/dong_l/EEC510/material/Simulink%20Tutorial.pdf)
9. OMG Unified Modeling Language Specification [PDF] (http://www.omg.org/spec/UML/1.4/PDF)
10. ArgoUML site [HTML] (http://argouml.tigris.org/)
11. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 26 April 2011 [HTML] (http://www.w3.org/TR/scxml/)
12. Cheetah Users' Guide [PDF] (http://www.cheetahtemplate.org/docs/users_guide.pdf)