АРХИТЕКТУРНАЯ МОДЕЛЬ БИБЛИОТЕКИ РЕФЛЕКСИИ ДЛЯ НЕУПРАВЛЯЕМОГО C++

Михтонюк С. В., Приходченко Р. С.
Научный руководитель — д.т.н., проф. Хаханов В.И.
Харьковский национальный университет радиоэлектроники,
(61200, Харьков, пр. Ленина,14, каф. АПВТ, тел. (057) 702-13-26),

A reflection mechanism, available under most managed languages, allows to greatly facilitate the solution of several classes of problems connected with parsing and scripting. This article discusses architectural concerns of an implementation of reflection library in native C++ that allows saving of meta-level information about program structure up to the program runtime.

Большинство управляемых языков программирования позволяют разработчику получать доступ к метаинформации об исполняемой программе. Этот механизм называется рефлексией либо интроспекцией. Он позволяет сохранить информацию об иерархии классов, об их членах и методах во время выполнения программы. В unmanaged языках, из соображений производительности, встроенного механизма рефлексии нет, потому эта информация теряется на этапе компиляции.

При компиляции C++ кода, все имена переменных и методов заменяются на их адреса и смещения в памяти, что в большинстве случаев подходит программисту, однако, для определенного класса задач, информации уровня экземпляра и класса (M0 и M1-уровни) не хватает. Эту проблему можно проиллюстрировать на примере командной консоли, которая предоставляют доступ к параметрам объектов программы в текстовом виде. Интерпретатору команд необходимо иметь возможность связывать имена объектов с их текущими значениями в программе. Один из способов решения данной задачи — описать имена всех доступных

функций в модуле интерпретатора и определить их связь с функциями программы. Недостатки данного подхода очевидны: необходимо каждый раз модифицировать интерпретатор при добавлении новых функций, интерпретирование параметров откладывается до кода пользователя.

Более эффективным решением, рассматриваемым в данной работе, является закрепление за каждым объектом, используемым в скрипте, таблицы предоставляемых им свойств и методов. Чтобы облегчить интерпретацию параметров вводится объект, который может единообразно хранить значения любого типа и предоставляет функции преобразований между различными типами. Достоинствами данного подхода является возможность добавления новых типов данных без изменения кода интерпретатора (справедливо и для plug-in-ориентированных систем) и возможность вызова функций с помощью упаковки динамически типизированных параметров.