**СОВРЕМЕННЫЙ ИНСТРУМЕНТАРИЙ РАЗРАБОТКИ ПО СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ: ПРОБЛЕМЫ И РЕШЕНИЯ**

**Е. А. Зуев, к.ф.-м.н., доцент,  
А. В. Канатов, доцент**

**Аннотация**

В статье рассматриваются вопросы, возникающие на пути создания отечественного инструментария разработки ПО. Обсуждаются потенциальные проблемы в связи с использованием заимствованных инструментов, предлагаются возможные направления решения этих проблем. Ключевой идеей статьи является утверждение и обоснование необходимости смещения центра тяжести на создание отечественных инструментов разработки ПО.

**1. Инструментарий разработки ПО и импортозамещение**

В настоящее время проблема импортозамещения в сфере программного обеспечения (ПО) находится в центре внимания профессиональной общественности. Только на пути создания отечественных программных решений можно достичь необходимой степени безопасности и надежности функционирования сложных современных комплексов различного назначения.

Одним из наиболее существенных аспектов создания отечественного ПО, которое, к сожалению, находится на периферии усилий в этом направлении, представляется разработка инструментов и сред программирования. В первую очередь это относится к проблеме создания отечественных компиляторов для современных отечественных языков программирования (ЯП).

Традиционно считается, что ключевым аспектом безопасности являются надежное базовое и прикладное ПО (то есть, операционные системы и прикладные пакеты), и именно эти категории программ нуждаются в импортозамещении в первоочередном порядке. С другой стороны, инструментальное ПО (компиляторы и сопутствующие средства разработки) не считается критическим в аспекте безопасности, так как реализует сугубо технический механизм эквивалентного преобразования исходного текста программ в машинные коды. Поэтому для создания прикладного ПО было бы достаточно взять за основу какой-либо общедоступный инструмент, распространяемый по свободной лицензии, адаптировав его для целевой аппаратной платформы.

Представляется, что данная точка зрения на инструментальные средства разработки ПО является поверхностной и потенциально несущей в себе изъяны и угрозы в плане безопасности создаваемого ПО.

Обычная технология адаптации наиболее распространённых инструментов разработки (семейство компиляторов gcc [1] или программный комплекс llvm/clang [2]) предполагает использование машинно-независимой фронтальной части такого инструмента («front-end compiler») и разработку собственного машинно-зависимого генератора кода («back-end compiler») для целевого аппаратного оборудования.

Будучи на первый взгляд вполне естественной, такая технология, тем не менее, несёт в себе ряд потенциальных уязвимостей. Дело в том, что интерфейсом между «стандартной» фронтальной частью компилятора и генератором кода служит так называемое промежуточное представление (ПП) исходной программы. Это ПП порождается фронтальной частью компилятора и используется в качестве исходной информации для любого генератора кода, создаваемого в рамках описанной технологии. Формат и алгоритмы порождения ПП фронтальной частью полностью определяются авторами инструмента и считаются их «внутренней кухней». Этот формат, как правило, плохо документирован (особенно в случае комплекта gcc) и может быть изменен авторами в любой момент, о чём в документации имеются специальные предупреждения.

Однако, наиболее проблематичным служит то обстоятельство, что отсутствуют какие бы то ни было *гарантии адекватности* ПП, созданного фронтальным компилятором, тексту исходной программы. Чтобы удостовериться в такой адекватности, необходим детальный анализ алгоритмов работы фронтальной части, который из-за большого объёма исходного текста компилятора практически не представляется возможным.

Ситуация осложняется нестандартной постановкой задачи: целью такого анализа является не поиск ошибок (который в ряде случаев можно хотя бы частично автоматизировать), а установление *семантического соответствия* между фрагментами текста исходной программы и соответствующими фрагментами промежуточного представления.

Несоответствие структуры и содержания ПП исходному тексту прикладной программы может возникнуть как в результате ошибки проектирования или программирования компилятора, так и оказаться намеренной «закладкой» в его коде. По мнению авторов, оба этих варианта в равной степени делают созданный по описанной технологии компилятор неприемлемым для целей разработки надежного прикладного ПО, удовлетворяющего специальным требованиям.

Следует указать также на более фундаментальный недостаток использования заимствованных инструментальных средств. Дело в том, что природа распространенных «свободно-доступных» инструментов разработки исключает наличие какого-либо центра ответственности за их сопровождение и развитие. Такие инструменты, как правило, разрабатываются по принципу «базара» с участием сотен программистов, ни один из которых, строго говоря, не несет ответственности за проект в целом. Лицензионные соглашения, которыми снабжаются свободные инструменты, прямо снимают с их создателей какую-либо ответственность за ненадлежащее функционирование представляемых средств. То же относится к выработке стратегии развития «свободных» проектов: их пользователи, как правило, не могут существенно повлиять на направление и темпы эволюции проектов, а также на скорость исправления обнаруживаемых ошибок.

Преодоление описанных органических недостатков технологии использования общедоступных инструментальных средств возможно только на пути разработки собственных компиляторов ЯП. Таким образом, реализация отечественных инструментальных средств представляется одной из первоочередных задач в контексте общего направления на импортозамещение в сфере ПО.

**2. Язык С++ и его реализация**

Преодоление проблем с заимствованными компиляторами ЯП возможно только на пути создания альтернативных отечественных компиляторов для распространенных промышленных ЯП. Одним из первых кандидатов на реализацию, по нашему мнению, является язык С++ [3]. Являясь наиболее адекватным в настоящее время инструментом разработки больших и сложных систем, этот язык поддерживает последние тенденции и достижения науки и практики программирования.

В то же время, C++ представляет собой предельно сложный объект – как для изучения, так и для использования и реализации. Язык обременен значительным объёмом устаревших и опасных свойств и возможностей, доставшихся ему в наследство от языка-предшественника, а также внесённых в процессе многолетнего хаотического процесса усовершенствования. Грамматика языка существенно неоднозначна, изобилует неуклюжими и загадочными конструкциями. Семантика языка явно переусложнена и зачастую спрятана за причудливыми синтаксическими конструкциями. Всё это, а также весьма большой объём языка и его библиотек существенно усложняет его освоение и эффективное использование.

Несмотря на сказанное выше, по историческим (большой объём существующего кода, наличие большой армии разработчиков) и коммерческим (трудности перехода разработчиков на другие ЯП, затраты на переобучение специалистов) причинам язык C++ продолжает оставаться одним из наиболее распространённых инструментов разработки ПО, и данная ситуация будет оставаться неизменной на протяжении нескольких десятков лет.

Таким образом, *задача реализации языка С++* в интересах отечественной индустрии разработки ПО представляется важной и актуальной – несмотря на указанные выше проблемы и недостатки этого языка. Очевидно, данная работа должна включать тщательный и всесторонний анализ решаемой задачи, а именно:

* Анализ требований к компиляторам ЯП высокого уровня применительно к решению современных задач, стоящих перед отечественной сферой ИТ.
* Комплексный анализ языка С++ стандарта ISO 2020 с точки зрения современного понимания задач компиляции.
* Описание и обоснование принципиальной схемы компиляции С++, удовлетворяющей выдвинутым требованиям.
* Описание и обоснование ключевых проектных решений, положенных в основу проекта компилятора С++ нового поколения.

В последние два года в университете Иннополис проводится работа по указанным направлениям, а также реализуется ряд экспериментальных проектов [4, 5], имеющих конечной целью создание эффективного компилятора С++ с продвинутыми характеристиками.

Следует отметить, что решение подобной задачи, несмотря на свою сложность, имеет все шансы на успех. В начале 2000-х гг. в компании Интерстрон (Москва) под руководством одного из авторов статьи был разработан компилятор С++ [6] для ряда отечественных микропроцессоров. Опыт, полученный в ходе выполнения проекта, используется в текущей работе.

**3. Языковое обеспечение разработки ПО**

Следует остановиться на более глубокой проблеме, связанной с современными ЯП. Язык программирования как таковой является фундаментально важным компонентом процесса разработки любого ПО, и его достоинства (как и недостатки) могут радикально повлиять на качество ПО, создаваемого с его помощью [7]. Поэтому одним из важнейших негативных факторов, сдерживающих создание высококачественного программного обеспечения, служит недостаточное качество современного инструментария разработки – языков программирования.

Широко используемые в настоящее время языки программирования в большинстве были созданы 20 и более лет назад. Они не обеспечивают должный уровень надежности ПО, архаичны, неуклюжи, сложны для изучения и использования, отражают устаревшие взгляды и подходы к процессу разработки ПО, несут отпечаток личных вкусов и пристрастий их авторов и, что самое главное, не соответствуют современным требованиям, предъявляемым к инструментам создания ПО промышленного и специального назначения.

В последние годы предпринимаются попытки преодоления указанных недостатков за счет проектирования и реализации новых ЯП. Практически все крупные зарубежные компании, чей профиль связан с разработкой ПО различного назначения, решают эти проблемы созданием собственных средств программирования. Большинство новых ЯП, созданных компаниями, в большей степени отражают современные подходы к созданию ПО, однако по-прежнему несут в себе многие устаревшие свойства. В значительной степени этот недостаток связан с требованиями обратной совместимости с уже разработанным программным кодом; но так или иначе, эти попытки нельзя признать полностью удачными.

По мнению авторов, наиболее конструктивным способом преодоления проблем с современными средствами разработки ПО служит проектирование и реализация *нового языка программирования* и соответствующего инструментария – компилятора, редактора связей (компоновщика), конфигуратора программ, комплекса стандартных библиотек.

**4. SLang: проект общецелевого ЯП нового поколения**

В ходе выполнения исследовательских и предпроектных работ, проводившихся в университете Иннополис в 2017-2019 гг. был получен ряд практических результатов [8, 9], позволяющих сделать вывод о возможности создания в приемлемые сроки современного комплекса разработки, в значительной степени свободного от известных недостатков современных средств и обеспечивающих создание ПО с высокими показателями надежности и производительности.

Представляется, что актуальность указанных исследований обусловлена насущной необходимостью повышения надежности и производительности отечественных программных и аппаратно-программных комплексов.

Целевая ориентация проекта (рабочее название **SLang**) предусматривает использование нового языка в широком спектре приложений, важнейшими из которых служат следующие:

* Разработка больших и сложных программных и программно-аппаратных систем с повышенными требованиями к надежности (mission-critical applications).
* Встроенные системы промышленного назначения: робототехнические комплексы, беспилотные летательные аппараты, автомобили и другие мобильные устройства.
* ПО для обеспечения безопасной и эффективной передачи данных: поддержка технологии блокчейн.
* Программные системы с ограниченными возможностями по энергопотреблению.
* Системы с элементами искусственного интеллекта: нейронные сети, обработка и анализ больших массивов информации («big data»), реализация эффективных алгоритмов распознавания образов.
* Системы управления процессами разработки ПО; программные комплексы верификации и сертификации ПО по различным критериям.

Представляется, что ключевые концептуальные требования, предъявляемые к современным инструментам разработки программ, и одновременно фундаментальные свойства современного языка программирования могут быть сведены к следующим:

* Один язык для различных аппаратных платформ; один язык для разных классов задач.
* Поддержка современных парадигм программирования –процедурной, объектно-ориентированной, обобщенной, функциональной и параллельной.
* Модульность и раздельная компиляция.
* Использование гибких моделей выполнения – непосредственный машинный код, интерпретируемый код, смешанный режим, оптимизации времени выполнения – в зависимости от особенностей целевой аппаратуры и от требований к разрабатываемому ПО.

Важнейший смысл проекта заключается в том, чтобы предложить разработчикам программного обеспечения XXI века инструмент, который бы позволил решать разнообразные задачи наиболее простым и надежным способом в соответствии с предъявляемым требованиям и применительно к разнообразным уровням квалификации разработчиков.

**5. Язык SLang: основные характеристики и ожидаемые преимущества**

Язык SLang воплощает адекватное понимание существа процесса проектирования и разработки современного ПО. Он включает свойства, обеспечивающие надежность, безопасность, эффективность и масштабируемость программ, создаваемых с его использованием. В то же время он достаточно прост для обучения, освоения и использования, что обеспечит «гладкий» процесс разработки и сопровождения, а также предоставит возможность включить в сферу разработки ПО более широкие, нежели в настоящее время, сообщества разработчиков, в том числе, специалистов прикладных областей, не являющихся профессионалами-программистами.

SLang включает широкий набор свойств, в совокупности обеспечивающих весь спектр современных подходов, методик и парадигм разработки программ (объектно-ориентированное, функциональное, обобщенное, модульное и параллельное программирование). При этом язык спроектирован таким образом, чтобы избежать или свести к минимуму те недостатки современных ЯП, которые не позволяют считать их подходящими для создания эффективных и высоконадежных программ, отвечающих современным требованиям.

Ниже приводится список основных свойств языка SLang, проектируемого в рамках одноименного проекта:

* Строгая типизация с продвинутой поддержкой механизма выведения типов на этапе компиляции.
* Непротиворечивая семантика и ясный синтаксис языка.
* Автоматическое управление памятью.
* Многоуровневый, безопасный и простой в использовании параллелизм.
* Возможности для оптимизации программ.
* Защитное программирования с предикатами и поддержка автоматической верификации.
* Типовая безопасность; контроль неинициализированных объектов.
* Поддержка параметризации разных видов.
* Оригинальный подход к модульности – контейнеры.

Использование языка SLang для разработки ПО, решающего широкий класс задач, может принести ряд несомненных выгод, осторожные оценки которых приводятся в следующей таблице.

|  |  |  |
| --- | --- | --- |
| **Ожидаемое преимущество** | **Консервативная оценка (по сравнению с использованием С++)** | **Свойства языка, за счёт которых преимущество будет достигнуто** |
| Надежность разрабатываемого кода | В 3-5 раз выше | * Предикаты: предусловия, постусловия, инварианты. * Надежность системы типов: полная поддержка конформности и явных преобразований * Отсутствие нулевых указателей и неинициализированных данных |
| Стоимость разработки и поддержки ПО | В 3-5раз ниже | * Множественное наследование с эффективным разрешением конфликтов. * Безопасное взаимодействие с существующими программными компонентами. * Поддержка процедурного, объектно-ориентированного, функционального и параллельного программирования. |
| Производительность создаваемого ПО | Не хуже C++ | * Языково-зависимые высокоуровневые оптимизации. * Оптимизации на стадии генерации кода (с использованием возможностей платформ класса LLVM). * Оптимизации времени выполнения. |
| Сложность параллельного программирования | В 3-5раз меньше | * Явный параллелизм на уровне входного языка: несколько простых принципов и одно дополнительное служебное слово. * Автоматический параллелизм для тел подпрограмм. |

**ЛИТЕРАТУРА**

# GCC, the GNU Compiler Collection, <https://gcc.gnu.org/>

1. The **LLVM** Compiler Infrastructure, <https://llvm.org>
2. Working Draft, Standard for Programming Language C++, <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4700.pdf>
3. Е.А. Зуев, Инфраструктура семантического анализа программ на С++. В: Языки программирования и компиляторы — 2017 Труды конференции (3–5 апреля 2017 г.) Под редакцией Д. В. Дуброва, Ростов-на-Дону 2017, http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf
4. Е.А. Зуев, Инфраструктура семантического анализа программ на С++. Конференция С++ Russia 2018, Санкт-Петербург, 21.04.2018.
5. Е.А. Зуев, Назначение и основные особенности компилятора Си++. В: Приложение к журналу КомпьюЛог, N3 (39), 2000, стр. 4-9.
6. E. Dijkstra, How do we tell truths that might hurt? In: Edsger W. Dijkstra, Selected Writings on Computing: A Personal Perspective, Springer-Verlag, 1982. ISBN 0–387–90652–5.
7. А. Канатов, Е. Зуев, Beyond C++: проект современного языка программирования общего назначения. В: Языки программирования и компиляторы — 2017 Труды конференции (3–5 апреля 2017 г.) Под редакцией Д. В. Дуброва, Ростов-на-Дону 2017, <http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf>
8. А. Канатов, Е. Зуев, Проект СЛанг: текущее состояние и перспективы. Конференция CEE-SECR-2017, Санкт-Петербург, 20-22 октября 2017.