Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Робототехника и комплексная автоматизация»

РЕФЕРАТ

«*История языков программирования*»

Выполнил аспирант: Бурча А.А.

Группа: РК6-21А

Научный руководитель: к.ф.-м.н., доцент, Соколов А.П.

Проверил: д. филос. н., профессор **Багдасарьян Н. Г. _____**

Оценка _____ Дата _____

2021 г.

Оглавление

Введение

Программирование появилось задолго до 50-х годов XX века. Первые идеи высказал ещё Чарльз Бэббидж (1792-1871), которого по праву считают отцом компьютера. Он изобрел механический аппарат, предназначенный для автоматизации вычислений путём аппроксимации функций многочленами и вычисления конечных разностей. Этот аппарат был назван «Большая разностная машина Бэббиджа». Развила идею графиня Ада Лавлейс (1815-1852). Ада описала алгоритм вычисления чисел Бернулли на разностной машине Бэббиджа. Было признано, что это первая программа, специально реализованная для воспроизведения на компьютере. По этой причине Ада Лавлейс считается первым программистом несмотря на то, что машина Бэббиджа так и не была построена при жизни Ады. Благодаря её трудам стало понятно, что путь к эффективному использованию машин – алгоритмы, описанные в коде [1].

Но программирование не могло развиваться в отрыве от компьютеров. Поэтому вплоть до 1950-х языки программирования представляли из себя набор машинных инструкций, которые часто были узкоспециализированными и использовались только вместе с целевым устройством.

1. Изобретение ассемблера

Первые программы писались на машинном языке, так как для ЭВМ того времени еще не существовало развитого программного обеспечения, а машинный язык – это единственный способ взаимодействия с аппаратным обеспечением компьютера.

[1]Fuegi. J, Francis. J, Lovelace & Babbage and the creation of the 1843 'notes', Annals of the History of Computing (IEEE). — Т. 25 (4): 16–26

Каждую команду машинного языка записывали в виде последовательности номеров операций и ячеек памяти. Человеку воспринимать программу на таком языке сложно. Кроме того, даже небольшая программа получалась состоящей из множества строк кода. Ситуация осложнялась еще и тем, что каждая вычислительная машина понимает лишь свой машинный язык.

Людям, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами, а не цифрами привело к появлению в 1949 году языка ассемблера.

Ассемблер был разновидностью низкоуровневого языка программирования, который упростил язык машинного кода. Другими словами, это язык, в котором вместо численных обозначений команд и областей памяти используются символы и слова.

Название Ассемблер происходит от английского слова assemble – собирать, монтировать, что вполне точно описывает процесс. Позднее символы стали касаться не только простейших операций, но и адресации, что значительно упростило читаемость кода.

Сейчас это кажется элементарным решением, но тогда реализация была сложным процессом, требующим созданий таблиц соответствия, присвоения обозначения каждой ячейке памяти. Это привело к появлению понятия переменная и к созданию таблиц, с помощью которых вы могли найти соответствие символов, операций и ячеек памяти.

Ассемблер позволил создать простые превращения. Например, перевод 01 в ADD. Макроассемблер расширил эту идею и подарил программистам возможность сворачивать несколько инструкций в одну. К примеру, если в программе вы постоянно добавляли значение в ячейку памяти и проверяли, не переполнена ли она, всё это можно было записать в макрос INCRT и

использовать его, меняя лишь переменные. По сути, макроассемблеры превратились в первые языки высокого уровня[2].

Но в таком подходе заключалась важная проблема – каждый раз перед созданием кода необходимо было сворачивать базовые операции в макросы. Требовался инструмент, который освободит программистов от постоянного копирования. Так появились трансляторы.

С ними проявилась другая трудность – арифметические выражения. Их исполнение не совпадает с тем, как машина читает код. Машина читает код в одном направлении, а это часто не соответствует приоритету выполнения операций в выражении.

2. Появление первых высокоуровневых языков

Найти алгоритм анализа приоритета выполнения операций удалось Джону Бэкусу, создателю Fortran. Полное название Fortran – The IBM Formula Translating System, или FORmula TRANslator. Fortran был создан в 1957 году и считается старейшим языком программирования, используемым сегодня. Этот язык программирования был создан для научных, математических и статистических вычислений высокого уровня. Fortran до сих пор используется в некоторых из самых передовых суперкомпьютеров в мире. Именно в Fortran впервые были одновременно реализованы многие атрибуты языка высокого уровня, среди которых:

- арифметические и логические выражения;
- цикл DO;
- условный оператор IF;
- подпрограммы;

[2] Алекперов А. Организация распределенных вычислений на базе GRID-технологии // Баку: Искусственный интелект. - 2011. - С. 6-14.

- массивы [3].

Следующим фундаментальным языком стал Algol (ALGOrithmic Language), предназначенный для научных отчётов и публикаций. Алгоритмический язык или Algol был создан совместным комитетом американских и европейских компьютерных ученых в 1958 году. Algol послужил отправной точкой для разработки некоторых из наиболее важных языков программирования, включая Pascal, C, C ++ и Java.

В том же 1958 году в Массачусетском технологическом институте (MIT) Джоном Маккарти был изобретен процессор списков или LISP. Первоначально предназначенный для искусственного интеллекта, LISP является одним из старейших языков программирования, которые все еще используются сегодня. Главной его особенностью стала работа не с императивными данными, а с функциями. Для этого Джону Маккарти пришлось предусмотреть множество механизмов для нормальной работы: динамическую типизацию, автоматическое распределение памяти, сборщик мусора. В конечном счёте, именно Лисп стал прародителем таких языков, как Python и Ruby, а сам до сих пор активно применяется в ИИ.

В 1959 году увидел свет Cobol (общий бизнес-ориентированный язык) - язык программирования, лежащий в основе многих процессоров кредитных карт, банкоматов, телефонных и сотовых вызовов, сигналов больниц и систем сигналов светофора. Разработкой языка руководила доктор Грейс Мюррей Хоппер, и он был разработан таким образом, чтобы его можно было использовать на компьютерах всех марок и типов. Cobol до сих пор используется в первую очередь для банковских систем. Синтаксис у него максимально приближен к естественному английскому языку. Cobol стал

[3] Леонтьев Д.В., Тарасов В.Г., Харитонов И.Д. Модель обработки данных вычислительных экспериментов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции. М.: ИПМ им. М.В.Келдыша, 2018. - С. 373–386.

олицетворением максимального удаления от прежнего машинно-арифметического мышления к общечеловеческому[4].

3. Высокоуровневые языки в СССР и мире

К началу 1960-х годов было создано два крупных комитета. Первый – академический, был создан эгидой Министерства Обороны США. Второй комитет, более крупный, представлял коммерческие интересы и имел статус международного. Изначально все они работали над развитием Fortran. Но из-за отсутствия нормальной работы с текстом он не мог в полной мере удовлетворить потребности бизнеса и науки. Так в привилегированной группе языков программирования появились COBOL и Algol.

Тем временем в СССР именно в 1960-е начинают применять «комитетские» языки. Первым начали применять ALGOL 60, транслятор с него был создан для машины БЭСМ-6. Долгое время именно он и ALGOL 68 считались «базовыми» для применения в научной сфере.

Экспансия Fortran началась с физиков. Из-за того, что большинство вычислений в CERN проводилось на этом языке, советским учёным пришлось создавать компиляторы для своих машин. Сначала это был компилятор для Минск-2 в 1967 году, а через год был выпущен ФОРТРАН-ДУБНА для всё той же БЭСМ-6. С этого момента Fortran входит в категорию избранных языков программирования в СССР, его начинают преподавать в вузах и научных институтах.

А вот Lisp был ввезён официально самим создателем, Джоном Маккарти. Он прочитал ряд лекций в Москве и Новосибирске в 1968 году, рассказав про язык и «искусственный интеллект». Кстати, этот термин придумал тоже он. Маккарти высказывал сомнения, что его язык будет

[4] McCormic B. Visualization in Scientific Computing // New York: Computer Graphics. 1987, 81 p.

работать на советских машинах. Наши учёные вызов приняли и в том же году адаптировали его под БЭСМ-6. Лисп-система на этих машинах успешно применялась вплоть до середины 1980-х, пока элементная база физически и морально не устарела[5].

Главным же итогом 1960-х в мире IT стало вовсе не создание комитетов и не формирование облика современных высокоуровневых языков, а то, что программирование стало народным. Теперь ему повсеместно обучали в университетах, студентам больше не требовалось быть инженерами, чтобы понимать, как функционирует машина с написанным ими текстом. Огромную роль в этом сыграли специально придуманные образовательные языки программирования.

4. Развитие образовательных и универсальных языков

Один из важнейших образовательных языков того времени BASIC был разработан Джоном Кемени и Томасом Курцем в Дартмутском колледже в 1964 году. BASIC – симбиоз FORTRAN II и ALGOL 60. Несмотря на большое количество ограничений, он стал первым языком, позволившим обучать программированию на «взрослом» уровне. То есть сделать переход от теории к работе с Algol или Fortran быстрым и безболезненным.

BASIC достаточно долго шёл к мировой известности поэтому, многие учёные продолжали создавать образовательные языки. Никлаус Вирт руководствовался теми же идеями, что и Кемени с Курцем. Он хотел создать язык, пригодный для обучения информатике в школах и вузах, при этом он должен был работать на мини-ПК и быть походим на один из комитетских языков[6].

[5] Godec, P., Pančur, M., Democratized image analytics by visual programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, C. 33-40.

[6] Kudryavtsev A.O., Koshelev V.K., Izbyshev A.O. Development and implementation of the cloud system for solving problems of high // Proceedings of the Institute for System Programming of Russian Academy of Sciences, Volume 24, 2011, p. 13-33.

Взяв за основу Algol, в 1970 году Вирт создал язык Pascal, названный в честь французского математика XVII века Блеза Паскаля. Упрощение коснулось не только синтаксиса, но и компиляции. Например, функции и процедуры должны быть определены в начале программы, это позволяло использовать однопроходный компилятор[7].

Пока BASIC и Pascal боролись за умы начинающих программистов, многие корпорации были озабочены разработкой мощного универсального языка высокого уровня, который бы был производительнее и удобнее тех, над которыми трудились комитеты. Больше всего в этом преуспела компания Bell Labs – именно здесь увидели свет операционная система Unix и язык программирования C.

Одним из недостатков существующих популярных компьютерных языков была их ограниченность. Например, если вы хотели написать операционную систему, сначала приходилось иметь дело с низкоуровневыми функциями аппаратного обеспечения, и лишь заложив фундамент, можно было перейти к программированию на языке высокого уровня.

В результате в Великобритании появились сначала системный язык CPL, а потом его урезанный младший брат – BCPL (базовый CPL). Его слава добралась до другой стороны Атлантики, где Кен Томпсон, сотрудник Bell Labs, выпустил еще более упрощённую версию BCPL – B. В то время Unix разрабатывался на ассемблере, поэтому B стал настоящим панацеей для компании.

Позднее Деннис Ритчи переработал некоторые идеи Томпсона и в 1972 году создал C (Он был назван C, потому что был основан на более раннем B). Язык сочетал в себе функции низкого уровня со структурой Algol. Таким

[7] Zurek R.W, Martin L.J. GridPP: development of the UK computing grid for particle physics // New York: Journal of Physics G: Nuclear and Particle Physics, 2008, p. 1–20.

образом, вначале C получил известность как структурированный язык ассемблера[8].

Самым ярким доказательством качества языка являлось то, что основная часть Unix, кроме небольшого ядра, была полностью переписана на C. Именно так, рука об руку, язык и ОС Unix существовали первые годы. Но слава распространялась, и C начал использоваться в других операционных системах, и как итог – получил славу языка общего назначения.

5. Рассвет объектно-ориентированных языков

Вне всяких сомнений, самой большой революцией 80-х годов стало появление объектно-ориентированных языков. Однако первопроходцем принято считать язык Simula, разработанный в 1967 году в Норвежском Вычислительном Центре тремя учёными – Оле-Йоханом Далем, Бьорном Мирхаугом и Кристен Нигаард. Язык был развитием Algol 60 и на первый взгляд не содержал ничего нового. Реальная же важность Simula 67 заключалась в философии разделения самой программы и её экземпляра. До этого большинство программистов не разделяло эти понятия. Даль и Нюгор, напротив, были озадачены именно созданием симуляций (отсюда и название языка).

Чтобы понять отличие, рассмотрим пример, пусть будет автомобиль. В традиционных языках (Fortran, Algol, BASIC или Cobol) вам необходимо писать подпрограмму или модуль для каждого автомобиля, хотя все они будут более или менее одинаковыми. В Simula 67 был реализован иной подход – вы создаёте лишь один модуль, определяющий автомобиль как набор переменных, процедур и функций, а затем просто создаёте экземпляры с

[8] Farkas Z., Kacsuk P. P-GRADE Portal: A generic workflow system to support user communities // Elsevier. - Volume 27 (Future Generation Computer Systems), 2010, p. 455–465.

частными параметрами. С этой простой идеи начала развиваться философия объектно-ориентированного программирования.

Следующим важным шагом в развитии объектно-ориентированных языков стал Smalltalk. Алан Кей жил на западном побережье США в эпицентре аппаратной революции. Главным его стремлением было создать компьютер, с которым мог работать даже ребёнок. Устроившись на работу в Xerox Palo Alto, Кей создал специальную исследовательскую группу. С компьютером не получилось, зато на свет появились Smalltalk 72 и 76, заимствовавшие идеи Simula. Об этом странном норвежском языке Кей прочитал в одном из научных журналов. Идея ему понравилась, и он положил её в основу своих разработок. Однако в 1981 году Кей покинул проект, после чего акценты Smalltalk изменились.

Отныне группа получила название «группа концепций программного обеспечения», а Smalltalk 80 позиционировался как «серьезный язык». Большой вклад в это внесли Адель Голдберг, Дэвид Робинсон, Ларри Теслер, Дэн Ингаллс и Питер Деат.

Smalltalk стал первым полноценным объектно-ориентированным языком и задал тренд на долгие годы. Кроме того, со Smalltalk связано ещё одно эпохальное событие. На этом языке был написана первая графическая среда. Однажды в Xerox Palo Alto зашёл Стив Джобс, восхитился этой разработкой и завербовал Ларри Теслера для создания LISA – предшественника первого Macintosh. Окна, иконки, мышь – всё это берёт начало от Smalltalk.

Объектно-ориентированный подход стал настолько популярен, что многие традиционные языки стали включать в себя этот подход. Бьёрн Страуструп из компании AT&T расширил возможности языка C и в 1983 году

создал C++. Главная особенность C++ – это полноценное расширение для C, делающее его полностью объектно-ориентированным языком [9].

Другими значимыми расширениями существующих языков были ответвления Pascal - Microsoft и Borland. Хоть они и не были совместимы, в подходах они схожи. Fortran, Lisp и Forth тоже включили объектно-ориентированные расширения, но в сравнении с C++ это были незначительные доработки. Появились и новые языки, например Ada. Изначально Ada была разработана командой во главе с Джин Ичбиа из CUU Honeywell Bull по контракту с Министерством обороны США. Названный в честь ученой середины 19-го века Ады Лавлейс, упомянутой ранее, Ada представляет собой структурированный, статически типизированный, императивный, объектно-ориентированный язык программирования высокого уровня с широким спектром возможностей.

6. Дальнейшая эволюция языков программирования

Basic и Pascal были искусственно ограничены в угоду массовости. Для сложных вычислений, моделирования и построения графиков программисту требовалось использовать сторонние библиотеки, а чаще создавать свои. Это порождало массу сложностей, куда проще было освоить ещё один язык. Кроме того, Pascal и Basic уже сильно устарели в плане синтаксиса.

В 1991 Гвидо Ван Россум при помощи Python решил все эти проблемы. Синтаксис стал по-настоящему высокоуровневым, настолько, что код мог понять даже не программист. Это вкупе с распространением интернета дало мощный толчок для создания узкоспециализированных библиотек, с помощью который можно решать безграничный спектр задач. Если в 1980-е каждый мог

[9] Godec, P., Pančur, M., Democratized image analytics by visual programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, C. 33-40.

сделать программирование своим хобби, то в следующем десятилетии оно уже стало элементом многих профессий.

Пожалуй, главный вклад 1990-х в современный мир состоит в появлении двух невероятно мощных и схожих языков – Java и C#. Java появился хронологически раньше. Язык стремился решить насущную проблему – создание такого кода, который бы мог работать на любой платформе. Реализация стала возможна благодаря трансляции в байт-код с использованием виртуальной машины. Идея не нова, она использовалась на машине Atlas в Кэмбридже ещё в 1960-х, но только сейчас она стала по-настоящему актуальной. Синтаксис Java был схож с популярными языками, в том числе C++, но функциональность была расширена до полноценных возможностей ООП [10].

Концепция Java очень быстро нашла отклик у программистов, и к концу века каждый четвертый программист на планете был знаком с этим языком. Конечно, такая популярность не могла пройти мимо внимания самой могущественной IT-компании – Microsoft. На базе Java, учтя все претензии и пожелания, они выпустили свой язык – C#.

7. Основные сформировавшиеся подходы к программированию

Описанные до этого момента языки программирования за исключением LISP придерживались подхода, который называется императивное программирование.

[10] US 2004/0056908 Al - Turbo Worx, Inc. (New Haven, CT, US) - Method and system for dataflow creation and execution, 2009, p. 3-14.

Императивное программирование — это парадигма программирования (стиль написания исходного кода компьютерной программы), для которой характерно следующее:

- в исходном коде программы записываются инструкции (команды);
- инструкции должны выполняться последовательно;
- данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;
- данные, полученные при выполнении инструкции, могут записываться в память.

При императивном подходе к составлению кода (в отличие от функционального подхода, относящегося к декларативной парадигме) широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и делает императивные программы подверженными специфическим ошибкам, не встречающимся при функциональном подходе.

Основные черты императивных языков:

- использование именованных переменных;
- использование оператора присваивания;
- использование составных выражений;
- использование подпрограмм;
- и др.

В 80-е годы стали все более часто появляться языки, придерживающиеся не императивной, а декларативной парадигмы программирования.

Декларативное программирование — парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается, что представляет собой проблема и ожидаемый результат.

К подвидам декларативного программирования также зачастую относят функциональное и логическое программирование — несмотря на то, что программы на таких языках нередко содержат алгоритмические составляющие, архитектура в императивном понимании (как нечто отдельное от кодирования) в них также отсутствует: схема программы является непосредственно частью исполняемого кода.

Функциональные языки программирования появились ещё на рассвете программирования: в конце 50-х был Lisp, который получил международное признание и долгое время являлся одним из китов программирования. Однако с тех пор, как появились Basic и Pascal, общественный интерес к функциональному программированию угас – их возможностей хватало на решение подавляющего большинства задач персонального компьютера. Но рост требований и появление интернета изменило требования и интерес к функциональному программированию вспыхнул с новой силой. Это стало катализатором появления и популяризации языков Erlang (1986) и Haskell (1990) – чистых языков функционального программирования. Это означает:

- Отсутствие побочных эффектов использования памяти и ввода-вывода информации, то есть при изменении частных параметров не меняется структура программы и объекты.
- Использование функций высших порядков. Функции могут быть аргументами и возвращаться в качестве результата вычисления.
- Активное использование рекурсий. В то время, как в императивном программировании в основе лежат циклы в классическом представлении, в функциональном в основе была рекурсия.

Функциональное программирование было актуальным решением в начале эпохи персональных компьютеров. Тогда сложные коды, привязанные к конкретным данным, было разумно замещать моделями, сокращая объем кода и увеличивая производительность. Новые языки и появление

полноценного ООП заставили большинство разработчиков вернуться к императивному программированию, оставив за функциональным роль дополнения. Но в XXI веке все изменилось.

Частично этому способствовала новая волна маломощных машин (смартфонов). Высокие требования при скромной производительности вынуждали программистов мыслить функциями, а не привязываться к данным. Частично на популярность ФП повлиял новый язык — Clojure, диалект Lisp, работающий на основе виртуальной машины Java. Он будто связал десятилетия, позволив в новой эре применить известные принципы функционального программирования.

Некоторые подходы из функционального программирования вобрали в себя узкоспециализированные языки и системы, направленные на сложные вычисления, используемые в инженерной среде. Среди таких систем можно выделить Matlab, Simulink и Labview.

LabView – система для построения моделей технических систем и устройств. Для этой цели LabView включает в состав множество специализированных библиотек реализованных компонентов, на базе которых возможно построение новых систем из конкретных технических областей.

Simulink – это графическая среда имитационного моделирования, позволяющая при помощи блок-диаграмм в виде ориентированных графов, строить модели динамических, дискретных, непрерывных и гибридных, нелинейных и разрывных систем. Simulink также ориентирован на решение задач моделирования сложных технических систем и представляет собой модельно-ориентированную систему проектирования. В первую очередь Simulink позволяет создавать графическую модель системы, собрав ее на основе стандартных базовых элементов, которой будет соответствовать

конкретная математическая модель системы на основе систем обыкновенных дифференциальных уравнений или уравнений в частных производных [11].

Заключение

Таким образом, можно проследить тенденцию к созданию специализированных языков программирования, которые также должны быть достаточно простыми и производительными. Каждый язык отвечает тому или иному требованию больше других. С целью объединить лучшие стороны различных языков в одном создаются новые подходы к разработке. Различные гибридные подходы на базе декларативного и императивного подходов предлагаются как решение, упрощающее использование мощных инструментов разработки.

В своей кандидатской работе я занимаюсь исследованием нового подхода к программированию, направленного на упрощение разработки программных реализаций методов, используемых в инженерных расчетах. Такой подход получил название графоориентированный.

Графоориентированный подход заключается в том, что пользователь представляет алгоритм сложного вычислительного метода (СВМ) в виде ориентированного графа. Сложным вычислительным методом будем называть множество вычислительных методов, взаимосвязанных друг с другом по входным и выходным данным.

Цель подхода – обеспечить инженера-математика набором инструментов для организации своего кода, позволяющих уменьшить трудозатраты на его создание, отладку, дальнейшие верификацию, валидацию и поддержку. Такой подход позволяет инженеру-математику использовать графовую модель алгоритма как уже готовую реализацию СВМ.

[11] Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – Т.47, No5– 2019, с. 43-55.

Графоориентированный подход включает в себя принципы как императивного, так и декларативного (функционального) программирования. Черты императивного программирования проявляются при написании модулей графовой модели на языках C++ и Python. Из декларативного программирования используется задание пользователем алгоритма сложного вычислительного метода в виде ориентированного графа, который представляет собой поток данных. При таком подходе пользователь составляет схему алгоритма и запускает расчет, что упрощает его работу.

Этот подход соответствует описанию принципа действия гибридного подхода к разработке и представляет исследовательский интерес.

Список литературы

1. Алекперов А. Организация распределенных вычислений на базе GRID-технологии // Баку: Искусственный интелект. - 2011. - С. 6-14.
2. Леонтьев Д.В., Тарасов В.Г., Харитонов И.Д. Модель обработки данных вычислительных экспериментов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции. М.: ИПМ им. М.В.Келдыша, 2018. - С. 373–386.
3. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – Т.47, No5– 2019, с. 43-55.
4. Farkas Z., Kacsuk P. P-GRADE Portal: A generic workflow system to support user communities // Elsevier. - Volume 27 (Future Generation Computer Systems), 2010, p. 455–465.
5. Fuegi. J, Francis. J, Lovelace & Babbage and the creation of the 1843 'notes', Annals of the History of Computing (IEEE). — Т. 25 (4): 16–26
6. Godec, P., Pančur, M., Democratized image analytics by visual programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, С. 33-40.
7. Kudryavtsev A.O., Koshelev V.K., Izbyshev A.O. Development and implementation of the cloud system for solving problems of high // Proceedings of the Institute for System Programming of Russian Academy of Sciences, Volume 24, 2011, p. 13-33.
8. McCormic B. Visualization in Scientific Computing // New York: Computer Graphics. 1987, 81 p.
9. US 2004/0056908 Al - Turbo Worx, Inc. (New Haven, CT, US) - Method and system for dataflow creation and execution, 2009, p. 3-14.
10. Zurek R.W, Martin L.J. GridPP: development of the UK computing grid for particle physics // New York: Journal of Physics G: Nuclear and Particle Physics, 2008, p. 1–20.