УДК 004.42 doi: 10.15827/0236-235X.142.202-211 2023. Т. 36. № 2. С. 202–211

Разработка механизма самосборки программ на основе сокетов

Е.А. Кольчугина

Ссылка для цитирования
Кольчугина Е.А. Разработка механизма самосборки программ на основе сокетов // Программные продукты и системы. 2023. Т. 36. № 2. С. 202–211. doi: 10.15827/0236-235X.142.202-211
Информация о статье
Поступила в редакцию: 30.01.2023 После доработки: 06.03.2023 Принята к публикации: 06.03.2023

Аннотация. Предметом исследования являются методы и алгоритмы спонтанной самосборки и самоорганизации программных систем. Среди моделей искусственной химии известны методы, допускающие самоформирование программ. Но эти методы очень специфичны и проблематичны в случае интеграции с обычными, широко распространенными и хорошо известными инструментами императивного программирования. Следовательно, необходимы другие виды инструментов, позволяющие динамически устанавливать отношения между программами или процессами. Разработанный автором данной статьи метод основан на использовании сокетов Internet, соединяющих программные единицы различных типов. Одни из этих единиц являются серверами, другие – клиентами, а третьи относятся к гибридному типу, сочетающему функции и клиента, и сервера. Программные единицы обычно рассматриваются как искусственные атомы, вступающие в реакцию друг с другом и образующие сложные вещества (то есть программы различной структуры). В данной работе предлагаются алгоритмы реализации таких программных единиц. Эти алгоритмы позволяют создавать коллективы независимых взаимодействующих единиц, способных формировать различные вычислительные конфигурации. Разработанные алгоритмы являются основой для реализации концепции, допускающей спонтанное формирование ПО в соответствии с заданными правилами при заданных условиях. В ходе экспериментов были получены вычислительные структуры, подобные полимерам реального мира и способные прокачивать данные через себя. Полученные результаты необходимы для организации полностью автоматизированного процесса разработки ПО, основанного на моделировании спонтанности. Процесс разработки программы потребует меньшего участия человека и потому станет более эффективным и экономически выгодным.

Ключевые слова: самоорганизация и самосборка программ, моделирование химических реакций и заряженных частиц, сокеты

Дальнейший путь автоматизации разработки ПО связан, по мнению автора, с наделением программных систем способностью к самоорганизации, то есть к спонтанной самосборке и самоупорядочению. И то, и другое предполагает возможность спонтанного, обусловленного внутренними закономерностями установления устойчивых связей между отдельными программными компонентами или, напротив, закономерного избегания установления таких связей. В случае спонтанной самосборки речь может идти о первоначальном формировании программной структуры. Понятие самоупорядочения шире: оно предполагает как формирование исходной упорядоченной программной структуры, так и переупорядочение ее компонентов на стадии последующего функционирования, например, с целью поддержания дальнейшего существования программной структуры [1].

Для обеспечения протекания процессов самоорганизации необходимо:

– выявить условия протекания процессов самоорганизации программ;

– ввести модельные аналоги для понятий энергии (свободной энергии в частности), энтропии, термодинамического равновесия, сил притяжения и отталкивания;

– обеспечить среду, в которой протекание процессов самоорганизации станет возможным;

– задать набор правил, в соответствии с которыми будут регламентироваться процессы самоорганизации;

– создать механизмы, обеспечивающие формирование целостных устойчивых программных структур на основе относительно независимых законченных программных компонентов, и определить принципы построения таких программных компонентов, играющих роль строительных блоков [1].

Остановимся подробнее на последнем пункте.

Механизмы установления связей между готовыми целостными программными единицами рассматриваются в теории искусственной жизни и ее подразделе – искусственной химии. Это направление относится к компьютерным наукам и исходит из аналогии между программами и молекулами и атомами химии реального мира [2, 3]. Подобно тому, как молекулы реального мира конструируются из атомов и более простых молекул в ходе реакций с привлечением или высвобождением энергии,

сложные программы потенциально могут формироваться в результате взаимодействия более простых программ с привлечением затрачиваемого процессорного времени. В этом случае программы выступают в роли аналогов молекул и атомов, их взаимодействие, приводящее к появлению новых программных структур, служит аналогом реакций, а доступное процессорное время, затрачиваемое в ходе образования структур или поддержания их существования, является аналогом свободной энергии, затрачиваемой на совершение работы в сторону уменьшения энтропии [1, 4]. Полномасштабная реализация подобной аналогии способна обеспечить поддержку процессов самоорганизации и полную автоматизацию разработки ПО.

Приведенная концепция наглядна и очевидна. Но ее эффективная реализация связана со следующими проблемами и затруднениями:

– имитация физики реального мира средствами компьютерного моделирования имеет очень ограниченные возможности [4];

– сложно подобрать или разработать средства, обеспечивающие собственно формирование структур из независимых или относительно независимых программных единиц; при этом взаимодействие между программными единицами должно быть избирательным и допускать перестройку структур.

Выявлению и анализу проблем и затруднений первого рода, связанных с адекватностью воспроизведения физики реального мира средствами компьютерной имитации, посвящена работа [4].

Вторая группа выделенных проблем, связанная с выработкой средств, обеспечивающих формирование сложных программных структур из более простых программных единиц, была обозначена давно, на заре формирования искусственной жизни как научного направления. На сегодняшний день среди таких средств можно выделить указанные еще в классических работах по искусственной химии, а именно:

– формирование сложных структур путем композиции функций, представляющих более простые структуры [5, 6];

– использование понятия соседства в пространстве памяти, как в клеточных автоматах [7];

– использование комплиментарных двоичных шаблонов, двоичных последовательностей – меток и программных переходов, кодируемых битовыми последовательностями [8, 9];

– склейка путем использования специальных атрибутов (пигментов) [10–12];

– формирование слабосвязанных неявно заданных вычислительных структур из множества независимых процессов (вполне возможно, представляющих некие прикладные программы), использующих общие внешние структуры данных, например БД [13].

Большинство этих средств, за исключением последнего, скорее, представляют чисто научный академический интерес, нежели имеют отношение к практическому программированию. Некоторые из них были предложены в рамках создания и разработки концепций, приведших к появлению новых языков программирования (как, например, Grapple [10]), стилей и систем программирования. Однако широкого распространения результаты этих разработок не получили, прежде всего, в силу их высокой концептуальной сложности и отсутствия возможности интеграции с массовыми популярными средствами императивного программирования. Анализ публикаций в ведущих зарубежных журналах за последние годы как в области самой искусственной химии, так и в близкой к ней области эволюционных вычислений – генетическом программировании показывает, что в целом ситуация не изменилась: речь идет либо о чисто академических исследованиях организации вычислений как с позиций развития искусственной химии как таковой [3, 14–16], так и с точки зрения применения результатов в других областях компьютерных наук [17–19], либо о методах эволюционной разработки программ на языках лишь функционального и логического программирования [20–22]. Таким образом, цель данного исследования, состоящая в создании механизмов саморазработки программ на языках императивного программирования, уже уникальна, что определяет уникальность результатов.

Последнее из перечисленных средств, обеспечивающих формирование сложных программных структур из более простых программных единиц, выделенное и рассмотренное в [13], отличается тем, что его применение способно вызвать формирование структурных образований на основе множества процессов, соответствующих совершенно независимым обычным прикладным программам, причем такой эффект возникает непреднамеренно. Однако получающиеся в итоге структуры так же легко разрушаются, как и образуются. Причина в том, что связи между отдельными процессами, являющимися элементами структуры, формируются неявным и непрямым образом. Эти связи слабы, неустойчивы и, в частности, легко разрушаются сторонними процессами.

В работах [4, 23] был предложен и разработан еще один механизм формирования сложных программных структур из независимых процессов, соответствующих обычным прикладным программам, созданным средствами императивного программирования. Этот механизм пытается воспроизвести модельный аналог действия электрического заряда и взаимодействие заряженных частиц, находящихся на внешних орбиталях атомов. В качестве базового средства для реализации такой аналогии используются сокеты Internet.

Как показали результаты экспериментов, приведенные в [23], применение разработанного в [4, 23] механизма образования структур способно привести к формированию разнообразных сложных программных структур из простых блоков. Здесь и далее рассматриваются вопросы реализации механизма взаимодействия между программными единицами, предложенного автором в [4, 23], в том числе и на примере модельной программы Water 29.04.2022 [24], функционирующей на платформе Linux.

Реализация данной программы была выполнена на языке программирования Perl. Этот язык обеспечивает более высокий уровень абстракции, чем C/C++ или Паскаль, позволяя быстро реализовать сложные проекты небольшому коллективу или даже одиночному разработчику. Программы на Perl отличаются компактностью. Однако обилие спецсимволов в синтаксисе языка затрудняет восприятие исходного текста программы, поэтому изложение сопровождается не примерами программ, а описанием алгоритмов, которые легко могут быть реализованы на любом современном языке программирования.

Приведенный список алгоритмов является безызбыточным и минимально необходимым для реализации предложенного механизма.

Метод реализации

Суть проведенного автором модельного эксперимента, описанного в [23], состояла в имитации образования химических соединений из множества модельных атомов кислорода и водорода. Искусственные атомы были представлены независимыми программными единицами, взаимодействие которых имитирует реакции и приводит к формированию новых программных структур путем их самосборки.

Искусственные атомы кислорода были стационарными, неподвижно закрепленными в ячейках модельного клеточного пространства и способными лишь оттягивать от других модельных атомов электроны с целью заполнения своих внешних орбиталей, играя роль акцепторов. Таким образом, искусственные атомы кислорода играли роль пассивных серверов, а количество свободных позиций на внешних орбиталях соответствовало допустимому количеству одновременно поддерживаемых соединений.

Искусственные атомы водорода устроены сложнее. Прежде всего, они были летучими и могли перемещаться произвольным образом в пределах модельного клеточного пространства. Кроме того, они могли предоставлять электрон со своей внешней орбитали для акцепторов, выполняя роль доноров. В модельных взаимодействиях такие искусственные атомы играли роль клиентов, инициирующих установление соединений с серверами.

Эксперимент, описанный в [23], состоял из двух частей. В первой части возможности искусственного атома водорода ограничивались вышеперечисленными. Тип соединений, которые удавалось получить в этом эксперименте, ограничивался модельными аналогами молекулы воды либо гидроксидной группы.

Во второй части эксперимента модельный атом водорода мог как предоставлять свой свободный электрон, так и притягивать электрон от другого искусственного атома водорода на имеющуюся вакантную позицию на внешней орбитали, уподобляясь в своем поведении искусственным атомам кислорода. Фактически во второй части эксперимента искусственные атомы водорода сочетали в себе функции и клиентов, инициирующих взаимодействия, и серверов, пассивно реагирующих на запросы других клиентов.

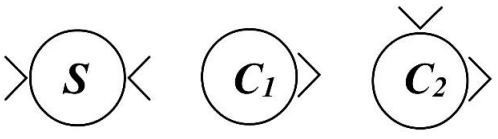
Такой подход позволил получить разнообразие модельных искусственных химических соединений, имеющих аналоги в реальном мире. Двойная роль атомов водорода, выполняющих функции и клиентов, и серверов, позволила строить длинные цепочки аналогов водородных соединений.

Все эти возможности были реализованы на основе сокетов домена Internet, которые, согласно [4, 23], могут использоваться как базовый механизм для реализации модельного аналога электрического заряда.

Таким образом, в модели реализованы три типа программных единиц, различающихся по характеру их взаимодействия с другими программными единицами: сервер, клиент,

клиент-сервер, способный как инициировать соединение с другими программными единицами, так и отвечать на запросы на соединение от других единиц.

На рисунке приведены условные графические обозначения для атомов, выполняющих роль клиентов, серверов или их гибридов [23]. Основой для такой системы обозначений послужило представление об атоме как доноре или акцепторе электронов. Угловая скобка играет роль воронки, показывающей направление притяжения или отталкивания модельного электрона. Таким образом, сервер притягивает модельные электроны к себе, а клиент предоставляет их вовне. Модельный электрон – это метафора для представления взаимодействия через сокеты между программными единицами [4].



Условные графические обозначения модельных искусственных атомов: S – сервер, C_1 – клиент, C_2 – клиент-сервер

The conditional graphical symbols of model artificial atoms: S – server, C_1 – client, C_2 – client-server

Как отмечалось ранее, использованный механизм организации взаимодействия программных единиц уже сам по себе на концептуальном уровне является новаторским. Поэтому вопросы алгоритмической и программной реализации такого механизма на уровне процессов-серверов, процессов-клиентов и процессов, сочетающих в себе свойства клиентов и серверов, заслуживают особого рассмотрения.

Остановимся подробнее на алгоритмах реализации средств взаимодействия для программных единиц, представляющих искусственные атомы кислорода и водорода двух видов (чистый клиент и клиент-сервер).

Рассмотрим алгоритмы реализации искусственного атома кислорода oxygen, child_oxygen, listen_oxygen.

Алгоритм oxygen реализации главного процесса искусственного атома кислорода.

Шаг 1. Определить номер порта для приема входящих подключений, идентифицирующий орбиту и ее орбиталь.

Шаг 2. Создать серверный сокет для приема входящих подключений.

Шаг 3. Создать дочерний процесс, реализующий алгоритм listen_oxygen, для журнализации поступающих сообщений. Зафиксировать идентификатор процесса.

Шаг 4. Если не установлен признак завершения, продолжить выполнение. В противном случае перейти к шагу 7.

Шаг 5. Определить недостающее число дочерних процессов, осуществляющих прием и обработку входящих подключений и реализующих алгоритм child_oxygen, равное количеству вакантных позиций на текущей орбитали. Породить нужное количество процессов child_oxygen и зафикисировать их идентификаторы.

Шаг 6. Ожидать завершения любого из дочерних процессов child_oxygen. Снять с учета идентификатор завершенного процесса. Перейти к шагу 4.

Шаг 7. Разослать оповещение о завершении всем дочерним процессам по их идентификаторам. Конец.

Алгоритм child_oxygen реализации процесса обработки входящего подключения для искусственного атома кислорода.

Шаг 1. Через серверный сокет, полученный при вызове из oxygen, принять входящее подключение, ответвив от него клиентский сокет.

Шаг 2. Если не установлен признак завершения и клиент не отключился от сокета, продолжить выполнение. В противном случае перейти к шагу 5.

Шаг 3. Принять данные через клиентский сокет.

Шаг 4. Передать полученные на предыдущем шаге данные процессу listen_oxygen через вспомогательный сокет домена UNIX. Перейти к шагу 2.

Шаг 5. Если получен сигнал завершения, закрыть клиентский сокет. Конец.

Алгоритм listen_oxygen реализации процесса журналирования поступающих данных в составе искусственного атома кислорода.

Шаг 1. Создать сокет домена UNIX, в который будут перенаправляться данные, направленные для обработки искусственному атому кислорода.

Шаг 2. Если не установлен признак завершения, продолжить работу. В противном случае перейти к шагу 5.

Шаг 3. Принять входящее подключение к сокету UNIX, ответвить дочерний сокет.

Шаг 4. Пока имеются входные данные и не установлен признак завершения, осуществлять прием данных через дочерний сокет.

Шаг 5. По завершении данных и при наличии признака завершения закрыть сокет UNIX

и завершить выполнение. Конец. В противном случае перейти к шагу 2.

В сущности искусственный атом кислорода реализуется как сервер с предварительным ветвлением [25], который поддерживает на постоянном уровне пул дочерних процессов, готовых к немедленной обработке входящих подключений. Каждый такой процесс моделирует вакантную позицию на внешней орбитали искусственного атома, способствующую формированию связи с другим атомом благодаря притягиванию имеющегося у другого атома свободного электрона. Моделирование таких вакантных позиций приводит к необходимости порождения главным процессом oxygen дочерних процессов child_oxygen, число которых должно поддерживаться на постоянном уровне.

Роль процесса listen_oxygen состоит в линеаризации потока данных, поступающих от дочерних процессов child_oxygen. В рассматриваемой реализации модели искусственной химии из [23] обработка данных искусственным атомом кислорода практически не производится, сводясь исключительно к журналированию данных, поступающих в виде сообщений от других искусственных атомов, взаимодействующих с текущим.

Если же впоследствии в более развитых и ориентированных на практическое применение разработках возникнет необходимость осуществления более сложной обработки данных, таковая может быть возложена как на процесс линеаризации потока поступающих данных, аналогичный listen_oxygen, так и на его подпроцессы или даже специально выделенный отдельный независимый процесс.

Представим алгоритм реализации искусственного атома водорода из первой части эксперимента, описанного в [23]. Это простейший вариант искусственного атома водорода, который мог выступать только в роли клиента при взаимодействиях с другими искусственными атомами. Функция, которую выполнял данный вид искусственных атомов, состоит в передаче своих данных процессу-серверу, то есть искусственному атому кислорода, к которому мог присоединяться водород.

Алгоритм hydrogen реализации искусственного атома водорода для первой части эксперимента.

Шаг 1. Если не установлен признак завершения, перейти к шагу 2. Иначе – к шагу 5.

Шаг 2. В текущей пространственной ячейке произвести попытку подключения к серверу. Если найти сервер не удается, то перейти к шагу 4. В противном случае – к шагу 3.

Шаг 3. Разыграть случайное значение от 0 до 1. Если оно больше порогового значения, то передать свои данные серверу, к которому произведено подключение, повторить шаг 3. Если оно меньше, перейти к шагу 4.

Шаг 4. Если разыгранное случайное число меньше порогового значения или не удалось подключиться к серверу, то вычислить новые пространственные координаты ячейки. Отсоединиться от сервера, если было подключение. Переместиться в соответствующую ячейку. Перейти к шагу 1.

Шаг 5. Конец.

Как уже отмечалось, искусственный модельный водород в эксперименте из [23] был летучим, то есть атомы искусственного водорода могли свободно мигрировать по клеточному двумерному пространству модели. В каждой новой посещаемой клетке атом водорода пытался соединиться со стационарным атомом кислорода, если таковой присутствовал в клетке (некоторые из клеток были пустыми). Если клетка была пустой или соединение с модельным атомом кислорода не формировалось, то искусственный атом водорода перемещался в другую клетку, выбранную случайным образом.

Соединение с искусственным атомом кислорода не только устанавливалось, но и разрушалось по инициативе искусственного атома водорода, после чего модельный атом водорода также перемещался в другую клетку.

Рассмотрим алгоритмы для более сложного варианта реализации искусственного атома водорода из второй части эксперимента, описанного в [23]. Этот вариант реализации соответствует модельной программе Water 29.04.2022 [24].

Алгоритм hydrogen2 реализации главного процесса искусственного атома водорода для второй части эксперимента.

Шаг 1. Создать фрагмент разделяемой памяти для получения данных от своего дочернего процесса.

Шаг 2. Если не установлен признак завершения, то в текущей пространственной ячейке сканировать все номера портов, соответствующие орбитам с 8-й по 1-ю, с целью отыскания сервера, с которым возможно соединиться. Если не удается найти сервер, то перейти к шагу 6. Если признак завершения установлен, перейти к шагу 7.

Шаг 3. Если удалось соединиться с сервером, то породить дочерний процесс, реализующий алгоритм child_hydrogen2. Зафиксировать идентификатор дочернего процесса.

Шаг 4. Разыграть случайное значение от 0 до 1. Если оно больше порогового значения, то передать свои данные и данные, полученные от дочернего процесса из разделяемой памяти, серверу, к которому произведено подключение, повторить шаг 4. Если случайное число меньше порогового значения, перейти к шагу 5.

Шаг 5. Если получен сигнал завершения или разыгранное случайное число меньше порогового значения, то разослать оповещение о завершении дочернему процессу по его идентификатору.

Шаг 6. Вычислить новые пространственные координаты ячейки для перемещения. Перейти к шагу 2.

Шаг 7. Освободить фрагмент разделяемой памяти. Конец.

Алгоритм child_hydrogen2 реализации процесса обработки входящего подключения для искусственного атома водорода для второй части эксперимента.

Шаг 1. Подключиться к фрагменту разделяемой памяти родительского процесса.

Шаг 2. Создать сервер для приема входящих подключений, используя порт, указанный родительским процессом.

Шаг 3. Если не установлен признак завершения, ожидать входящих подключений. В противном случае перейти к шагу 7.

Шаг 4. При поступлении входящего подключения ответвить клиентский сокет для проведения сеанса.

Шаг 5. Пока не закончился поток входных данных, поступающих через клиентский сокет, принимать их. Передать данные родительскому процессу через разделяемую память.

Шаг 6. Если поток входных данных закончился, закрыть клиентский сокет. Перейти к шагу 3.

Шаг 7. Закрыть серверный сокет. Отсоединиться от разделяемой памяти. Конец.

Отличительная особенность данного варианта реализации в том, что искусственный атом водорода может не только предоставлять свободные электроны (то есть быть клиентом в ходе взаимодействий), но и притягивать их от других атомов водорода (то есть выполнять роль сервера). Это обусловливает более сложную программную структуру искусственного атома водорода для данной реализации.

Первичная и наиболее существенная роль искусственного атома водорода сохраняется: это летучий модельный атом, который призван установить связь с атомом кислорода из текущей пространственной ячейки. Данная роль реализуется главным процессом hydrogen2.

Однако искусственный атом водорода должен позволять присоединяться к себе другим аналогичным атомам искусственного водорода. Это обеспечивается дочерним процессом child_hydrogen2, который выполняет роль сервера для присоединения других атомов искусственного водорода. Подобная схема обеспечивает возможность формирования водородных цепочек.

Как и в случае искусственных атомов кислорода, полезная функция, выполняемая искусственными атомами водорода, сведена к простейшей, а именно к перекачиванию данных от своего клиента к своему процессу-серверу. При этом главный процесс присоединяет к перекачиваемым данным и свои собственные. Таким образом, модельные химические соединения из [23] реализуют конвейерную прокачку данных с их последующей журнализацией.

Важную роль в успешности реализации приведенной схемы играет принцип нумерации портов, которые используются искусственными атомами для соединения [23].

Результаты и их обсуждение

Как было показано в [23], в результате применения предложенных средств взаимодействия базовых программных единиц (искусственных атомов) были получены сложные программы со структурами предсказанных типов, напоминающие разнообразные соединения атомов водорода и кислорода в реальном мире.

Эти программные образования способны выполнять простейшие функции, например, журналировать поступающие данные в файлах на диске или прокачивать данные через себя. Однако дело не в уровне сложности функций, реализуемых формирующимися программами, а в сложности структуры и топологии графов, описывающих взаимосвязи между базовыми программными единицами. Как показывают эксперименты, формирующиеся структуры могут состоять из множества разнородных элементов и быть протяженными, напоминая тем самым молекулы-полимеры [23]. Потенциально могут быть получены и более сложные структуры, например, древовидные или циклические.

Предложенные в [23] средства обладают таким преимуществом, как возможность обеспечить взаимосвязь между динамическими сущностями, исходный код которых недоступен для изучения и модификации, а именно выполняющимися процессами, возможно, соответ-

ствующими прикладным программам. Установление взаимосвязей приводит к формированию программных структур, имеющих более высокий, чем обычные прикладные программы, уровень. При этом такие структурные образования могут формироваться спонтанно, без целенаправленного воздействия.

Следовательно, рассмотренные средства, алгоритмы реализации которых представлены в настоящей статье, вполне могут служить частью механизма, обеспечивающего самоорганизацию программных систем.

Необходимо отметить и трудности, связанные с реализацией предложенного механизма формирования сложных программных структур.

Прежде всего, сложность самого предложенного механизма, который предполагает наличие двух несимметричных компонентов, обеспечивающих установление взаимосвязи между программными единицами.

Большинство из таких компонентов, а именно сервер и клиент-сервер, сами по себе сложносоставные и являются совокупностью отдельных процессов. Эти процессы должны действовать слаженно и обмениваться данными, что предполагает использование таких средств межпроцессного взаимодействия (IPC UNIX/Linux), как сокеты UNIX и разделяемая память. Применение средств IPC уже само по себе повышает сложность алгоритмов и программ, а также приводит к существенным затратам системных ресурсов. Это отрицательно сказывается на производительности вычислений и ограничивает число программных единиц, которые могут использовать предложенный механизм взаимодействия.

Однако, как отмечалось ранее, предложенный механизм взаимодействия программных единиц является наиболее явным, а потому понятным и контролируемым методом взаимодействия, обеспечивающим спонтанное формирование программных структур более высоких порядков на основе множества независимых программ в процессе их выполнения.

Поскольку производительность современных вычислительных систем постоянно повышается, можно надеяться, что отмеченные затруднения со временем потеряют свою остроту. В этом смысле стоит возлагать надежды на дальнейшее развитие моделей мелкозернистых вычислений, например туманных [26]. Такие подходы позволят сопоставить каждому искусственному атому собственное простое процессорное устройство, реальное либо виртуальное, что даст возможность повысить производительность вычислений.

Перспективным направлением дальнейших исследований также представляется интеграция предложенного механизма самосборки программ не только со средствами императивного программирования, но и с концепциями, возникшими в рамках направления искусственной химии на основе средств функционального и логического программирования, а также моделей программирования на основе потоков данных. К таким концепциям относится, например, концепция химического программирования [14].

Заключение

В настоящее время одной из наиболее затратных составляющих производственных расходов является человеческий труд. В связи с необходимостью обеспечения конкурентных преимуществ конечного продукта, в том числе достижения низкой стоимости, данная составляющая постепенно сокращается и вытесняется переходом от частичной автоматизации к полностью автоматическим производствам. Не является исключением и сфера разработки ПО. Уже сейчас широко применяются средства САПР и автоматизации программирования. Однако полной автоматизации труда программиста можно достичь только тогда, когда станет возможной самосборка образующих программу элементов, происходящая в соответствии с заданными правилами и требованиями и под воздействием внешних условий, задаваемых вычислительным окружением. Под элементами можно понимать как отдельные команды, так и более сложные единицы, в частности, искусственные атомы.

Первые результаты реализации подобной концепции в исследованиях автора статьи показывают ее состоятельность, что важно с точки зрения развития информатики и программной инженерии как наук. На практике это означает возможность перехода к полной автоматизации процессов разработки и сопровождения ПО, которые должны будут проходить под контролем человека, но без существенных затрат ручного и умственного труда.

Список литературы

1. Кольчугина Е.А. Программные системы с самоорганизацией континуального типа // Программная инженерия. 2013. № 3. С. 15–20.

2. Banzhaf W., Yamamoto L. Artificial Chemistries. London, The MIT Press Publ., 2015, 576 p.
3. Rainford P.F., Sebald A., Stepney S. MetaChem: an algebraic framework for artificial chemistries. Artificial Life, 2020, vol. 26, no. 2, pp. 153–195. doi: 10.1162/artl_a_00315.
4. Kol'chugina E.A. Model reproduction of non-equilibrium thermodynamics principles as a means to provide software self-development. IOP Conf. Ser.: Materials Sci. and Eng., 2021, vol. 1155, art. 012054. doi: 10.1088/1757-899X/1155/1/012054.
5. Fontana W. Algorithmic chemistry. In: Artificial Life II, pp. 159–209.
6. Fontana W., Baumgaertner B., Beslon G., Doursat R., Foster J.A. et al. Defining and simulating open-ended novelty: requirements, guidelines, and challenges. Theory in Biosci., 2016, vol. 135, no. 3, pp. 131–161. doi: 10.1007/s12064-016-0229-7.
7. Hutton T.J. Evolvable self-reproducing cells in a two-dimensional artificial chemistry. Artificial Life, 2007, vol. 13, no. 1, pp. 11–30. doi: 10.1162/artl.2007.13.1.11.
8. Ray T.S. An evolutionary approach to synthetic biology: Zen and the art of creating life. In: Natural Computing Ser., 2003, pp. 179–209. doi: 10.1007/978-3-642-18965-4_19.
9. Pargellis A.N. Self-organizing genetic codes and the emergence of digital life. Complexity, 2003, vol. 8, no. 4, pp. 69–78. doi: 10.1002/cplx.10095.
10. Pawlak R., Cuesta C.E., Younessi H. Recombinant programming. INRIA, 2004, 44 p. URL: https://hal.inria.fr/inria-00070623/document (дата обращения: 04.12.2022).
11. Hickinbotham S., Clark E., Stepney S. et al. Specification of the Stringmol chemical programming language version 0.2. Tech. Report Number YCS-2010-458. URL: http://www-users.cs.york.ac.uk/susan/bib/ss/nonstd/tr458.pdf (дата обращения: 04.12.2022).
12. Clark E.B., Hickinbotham S.J., Stepney S. Semantic closure demonstrated by the evolution of a universal constructor architecture in an artificial chemistry. J. of the Royal Society Interface, 2017, vol. 14, no. 130, art. 20161033. doi: 10.1098/rsif.2016.1033.
13. Kol'chugina E.A. Spontaneous emergence of programs from ″primordial soup″ of functions in distributed computer systems. Automatic Control and Comput. Sci., 2018, vol. 52, no. 1, pp. 40–48. doi: 10.3103/S0146411618010054.
14. Cudennec L., Goubier T. A short overview of executing chemical reactions over the and dataflow programming models. Procedia Comput. Sci., 2015, vol. 51, no. C, pp. 1413–1422. doi: 10.1016/j.procs.2015.05.349.
15. Hordijk W., Steel M., Dittrich P. Autocatalytic sets and chemical organizations: Modeling self-sustaining reaction networks at the origin of life. New J. of Phys., 2018, vol. 20, art. 015011. doi: 10.1088/1367-2630/aa9fcd.
16. Kruszewski G., Mikolov T. Emergence of self-reproducing metabolisms as recursive algorithms in an artificial chemistry. Artificial Life, 2022, vol. 27, no. 3-4, pp. 1–23. doi: 10.1162/artl_a_00355.
17. Atkinson T., Plump D., Stepney S. Horizontal gene transfer for recombining graphs. Genetic Programming and Evolvable Machines, 2020, vol. 21, no. 3, pp. 321–347. doi: 10.1007/s10710-020-09378-1.
18. Varela D., Santos J. Evolving cellular automata schemes for protein folding modeling using the Rosetta atomic representation. Genetic Programming and Evolvable Machines, 2022, vol. 23, no. 2, pp. 225–252. doi: 10.1007/s10710-022-09427-x.
19. Tabareau N., Tanter É. Chemical foundations of distributed aspects. Distributed Computing, 2019, vol. 32, no. 3, pp. 193–216. doi: 10.1007/s00446-018-0334-6.
20. O'Neill M., Spector L. Automatic programming: The open issue? Genetic Programming and Evolvable Machines, 2020, vol. 21, no. 1-2, pp. 251–262. doi: 10.1007/s10710-019-09364-2.
21. Medvet E., Bartoli A., De Lorenzo A. et al. Designing automatically a representation for grammatical evolution. Genetic Programming and Evolvable Machines, 2019, vol. 20, no. 1, pp. 37–65. doi: 10.1007/s10710-018-9327-2.
22. Ruberto S., Terragni V., Moore J.H. A semantic genetic programming framework based on dynamic targets. Genetic Programming and Evolvable Machines, 2021, vol. 22, no. 4, pp. 463–493. doi: 10.1007/s10710-021-09419-3.
23. Kolchugina E.A. Self-synthesis of programs based on artificial chemistry model // Программная инженерия. 2022. Т. 13. № 9. С. 440–448. doi: 10.17587/prin.13.440-448.
24. Кольчугина Е.А. Программа для моделирования спонтанного формирования программных структур при взаимодействии через сокеты Water 29.04.2022: Свид. о регистр. ПрЭВМ № 2022680600. Рос. Федерация, 2022.
25. Christiansen T., Torkington N. Perl Cookbook. Beijing-Cambridge, O'Reilly & Associates Publ., 1998, 794 p.
26. Matt C. Fog computing. BISE, 2018, vol. 60, no. 4, pp. 351–355. doi: 10.1007/s12599-018-0540-6.

Software & Systems doi: 10.15827/0236-235X.142.202-211 2023, vol. 36, no. 2, pp. 202–211

Developing a program self-assembly mechanism based on sockets

Elena A. Kol'chugina

For citation
Kol'chugina, E.A. (2023) 'Developing a program self-assembly mechanism based on sockets', *Software & Systems*, 36(2), pp. 202–211 (in Russ.). doi: 10.15827/0236-235X.142.202-211
Article info
Received: 30.01.2023 After revision: 06.03.2023 Accepted: 06.03.2023

Abstract. The paper focuses on methods and algorithms of spontaneous self-assembly and self-organization of software systems. Among the artificial chemistry models, there are some methods allowing program self-formation. But these meth-

ods are very specific and problematic for integration with conventional widespread and well-known imperative programming tools. Thus, it is necessary to offer other types of tools that enable dynamically establishing relations between programs or executing processes. The method previously proposed by the author is based on using Internet sockets connecting program units of different types. Some of these units are servers, some are clients, and some are of a hybrid client-server type. The units are generally considered as artificial atoms that react with each other and form complex substances (i.e. programs of different structures). This paper proposes the algorithms of such program units. Being implemented, these algorithms allow creating collectives of independent interacting program units capable to form different computing configurations. The designed algorithms are the basis for implementing the concept that allows spontaneous formation of the software in accordance with the specified rules under specified conditions. The experiments resulted in computational structures similar to real-world polymers and capable of pumping data through themselves. The obtained results are necessary for organizing a fully automated software development process based on the simulation of spontaneity. The program development process will require less human involvement and will therefore become more efficient and economically profitable.

Keywords: self-organization and self-assembly of programs, chemical reactions and charged particles simulation, sockets

Reference List

1. Kol'chugina, E.A. (2013) 'Software systems with self-organization of continual type', *Software Eng.*, (3), pp. 15–20 (in Russ.).

2. Banzhaf, W., Yamamoto, L. (2015) *Artificial Chemistries*, London, The MIT Press Publ., 576 p.

3. Rainford, P.F., Sebald, A., Stepney, S. (2020) 'MetaChem: an algebraic framework for artificial chemistries', *Artificial Life*, 26(2), pp. 153–195. doi: 10.1162/artl_a_00315.

4. Kol'chugina, E.A. (2021) 'Model reproduction of non-equilibrium thermodynamics principles as a means to provide software self-development', *IOP Conf. Ser.: Materials Sci. and Eng*., 1155, art. 012054. doi: 10.1088/1757-899X/1155/1/012054.

5. Fontana, W. (1991) 'Algorithmic chemistry', in *Artificial Life II*, pp. 159–209.

6. Fontana, W., Baumgaertner, B., Beslon, G., Doursat, R., Foster, J.A. et al. (2016) 'Defining and simulating open-ended novelty: requirements, guidelines, and challenges', *Theory in Biosci*., 135(3), pp. 131–161. doi: 10.1007/s12064-016-0229-7.

7. Hutton, T.J. (2007) 'Evolvable self-reproducing cells in a two-dimensional artificial chemistry', *Artificial Life*, 13(1), pp. 11–30. doi: 10.1162/artl.2007.13.1.11.

8. Ray, T.S. (2003) 'An evolutionary approach to synthetic biology: Zen and the art of creating life', in *Natural Computing Ser.*, pp. 179–209. doi: 10.1007/978-3-642-18965-4_19.

9. Pargellis, A.N. (2003) 'Self-organizing genetic codes and the emergence of digital life', *Complexity*, 8(4), pp. 69–78. doi: 10.1002/cplx.10095.

10. Pawlak, R., Cuesta, C.E., Younessi, H. (2004) 'Recombinant programming', *INRIA*, 44 p., available at: https://hal.inria.fr/inria-00070623/document (accessed December 04, 2022).

11. Hickinbotham, S., Clark, E., Stepney, S. et al. 'Specification of the Stringmol chemical programming language version 0.2', *Tech. Report Number YCS-2010-458*, available at: http://www-users.cs.york.ac.uk/susan/bib/ss/non-std/tr458.pdf (accessed December 04, 2022).

12. Clark, E.B., Hickinbotham, S.J., Stepney, S. (2017) 'Semantic closure demonstrated by the evolution of a universal constructor architecture in an artificial chemistry', *J. of the Royal Society Interface*, 14(130), art. 20161033. doi: 10.1098/rsif.2016.1033.

13. Kol'chugina, E.A. (2018) 'Spontaneous emergence of programs from ″primordial soup″ of functions in distributed computer systems', *Automatic Control and Comput. Sci*., 52(1), pp. 40–48. doi: 10.3103/S0146411618010054.

14. Cudennec, L., Goubier, T. (2015) 'A short overview of executing chemical reactions over the and dataflow programming models', *Procedia Comput. Sci*., 51(C), pp. 1413–1422. doi: 10.1016/j.procs.2015.05.349.

15. Hordijk, W., Steel, M., Dittrich, P. (2018) 'Autocatalytic sets and chemical organizations: Modeling self-sustaining reaction networks at the origin of life', *New J. of Phys*., 20, art. 015011. doi: 10.1088/1367-2630/aa9fcd.

16. Kruszewski, G., Mikolov, T. (2022) 'Emergence of self-reproducing metabolisms as recursive algorithms in an artificial chemistry', *Artificial Life*, 27(3-4), pp. 1–23. doi: 10.1162/artl_a_00355.

17. Atkinson, T., Plump, D., Stepney, S. (2020) 'Horizontal gene transfer for recombining graphs', *Genetic Programming and Evolvable Machines*, 21(3), pp. 321–347. doi: 10.1007/s10710-020-09378-1.

18. Varela, D., Santos, J. (2022) 'Evolving cellular automata schemes for protein folding modeling using the Rosetta atomic representation', *Genetic Programming and Evolvable Machines*, 23(2), pp. 225–252. doi: 10.1007/s10710-022-09427-x.

19. Tabareau, N., Tanter, É. (2019) 'Chemical foundations of distributed aspects', *Distributed Computing*, 32(3), pp. 193–216. doi: 10.1007/s00446-018-0334-6.

20. O'Neill, M., Spector, L. (2020) 'Automatic programming: The open issue?', *Genetic Programming and Evolvable Machines*, 21(1-2), pp. 251–262. doi: 10.1007/s10710-019-09364-2.

21. Medvet, E., Bartoli, A., De Lorenzo, A. et al. (2019) 'Designing automatically a representation for grammatical evolution', *Genetic Programming and Evolvable Machines*, 20(1), pp. 37–65. doi: 10.1007/s10710-018-9327-2.

22. Ruberto, S., Terragni, V., Moore, J.H. (2021) 'A semantic genetic programming framework based on dynamic targets', *Genetic Programming and Evolvable Machines*, 22(4), pp. 463–493. doi: 10.1007/s10710-021-09419-3.

23. Kolchugina, E.A. (2022) ‘Self-Synthesis of programs based on artificial chemistry model’, *Software Eng*, 13(9), pp. 440–448. doi: 10.17587/prin.13.440-448.

24. Kol’chugina, E.A. (2022) *Program for Modeling the Spontaneous Formation of Program Structures through Interaction via Sockets*, Pat. RF, № 2022680600.

25. Christiansen, T., Torkington, N. (1998) *Perl Cookbook*, Beijing-Cambridge: O’Reilly & Associates Publ., 794 p.

26. Matt C. (2018) ‘Fog computing’, *BISE*, 60(4), pp. 351–355. doi: 10.1007/s12599-018-0540-6.

Авторы

Кольчугина Елена Анатольевна [1], д.т.н., профессор кафедры математического обеспечения и применения ЭВМ, kea_sci@list.ru

[1] Пензенский государственный университет, г. Пенза, 440026, Россия

Authors

Elena A. Kol’chugina [1], Dr.Sc. (Engineering), Professor of the Department of Mathematical Support and Computer Application, kea_sci@list.ru

[1] Penza State University, Penza, 440026, Russian Federation