УДК 681.3.06:62-507

РЕШЕНИЕ ЗАДАЧ С ПОМОЩЬЮ КЛЕТОЧНЫХ АВТОМАТОВ ПОСРЕДСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ CAME&L (Часть I)

Л. А. Наумов,
аспирант
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Работа представляет собой введение в программирование для пакета CAME&L посредством библиотеки CADLib. Описываются основы решения задач с помощью рассматриваемого программного обеспечения, а именно – создание пользовательских правил для клеточных автоматов. На примере игры «Жизнь» демонстрируется разработка простейших решений, использование зональной оптимизации, поддержка многопроцессорной и кластерной вычислительных систем, а также – обобщенных координат. Кроме того, приводится пример решения физической задачи, а именно уравнения теплопроводности.

This work represents the introduction into programming for CAME&L software by means of CADLib library. Bases of building tasks' solutions with the help of considered software, namely the creation of user rules for cellular automata, are described. Here five variants of "Game of Life" are introduced: plain one, variant with zonal optimization, implementations for multiprocessor and cluster computing systems and for generalized coordinates. Moreover the solution of physical problem, thermal conductivity equation, is described.

Введение

Большинство актуальных в настоящее время вычислительных задач требует для своего решения мощностей, превосходящих возможности однопроцессорного персонального компьютера. Это обстоятельство послужило причиной чрезвычайно бурного развития теории, а также программных и аппаратных средств параллельных и распределенных вычислений.

Особой популярностью пользуются реализации стандарта MPI (Message Passing Interface) [1] и библиотека PVM (Parallel Virtual Machine) [2]. Их изучение включено в образовательные программы университетов мира.

Возникла идеология Grid-вычислений [3–5]. Наиболее популярным средством для их организации является пакет Globus Toolkit [6].

Настоящая работа посвящена еще одному средству для параллельных и распределенных вычислений – программному обеспечению CAME&L (Cellular Automata Modeling Environment & Library) [7–9], среде выполнения и библиотеке разработчика клеточных автоматов.

Клеточные автоматы [10] – дискретные динамические системы, поведение которых может быть полностью описано в терминах локальных зависимостей [11]. Спектр их применения чрезвычайно широк [12–14]. Например, они могут быть использованы для моделирования распределенных систем (пространственно-протяженных, распределенных во времени или по другому параметру).

Клеточные автоматы удобны для решения дифференциальных уравнений и уравнений в частных производных, так как эти уравнения описывают непрерывные динамические системы, а, следовательно, рассматриваемые автоматы являются их дискретными аналогами.

Можно считать, что клеточный автомат представляет собой дискретный эквивалент понятия «поле».

Кроме того, рассматриваемые автоматы образуют общую парадигму параллельных вычислений подобно тому, как машины Тьюринга образуют парадигму последовательных вычислений. В результате они могут быть использованы для реализации параллельных алгоритмов, как модели, обладающие естественным параллелизмом.

Было решено разрабатывать пакет CAME&L как средство, предоставляющее развитый инструментарий для решения научных и образовательных задач на основе клеточных автоматов.

При работе на однопроцессорном компьютере это программное обеспечение имеет образовательное значение, как инструмент для выполнения, изучения и визуализации параллельных алгоритмов. Многопроцессорная или кластерная платформы позволят использовать естественный параллелизм клеточных автоматов для осуществления трудоемких вычислений.

Анализ более чем двадцати пяти существующих средств решения задач с помощью клеточных автоматов подтвердил адекватность и необходимость разработки нового программного обеспечения в силу приведенных ниже обстоятельств.

1. Большинство из существующих средств решения задач с помощью клеточных автоматов не удовлетворяют современным требованиям к пользовательскому интерфейсу и не предоставляют возможности, которые стали обычными и неотъемлемыми и на поддержку которых пользователь имеет основания рассчитывать. В настоящее время отсутствуют программные продукты со столь удобным интерфейсом, каким обладает среда, входящая в рассматриваемый пакет. В ней, в частности, реализованы такие возможности, как поддержка буфера обмена; поддержка отмены/повторения последней операции; поддержка печати; возможность сохранения состояния решетки в виде рисунков для иллюстрирования публикаций.

2. Все известные продукты накладывают жесткие ограничения на используемый тип клеточных автоматов. Разрабатываемая среда универсальна и позволяет использовать модели, принадлежащие даже более широкому классу, чем класс «клеточные автоматы».

3. Почти все средства используют различные языки для описания автомата, причем иногда эти языки весьма сложны. Таким образом, появляется еще один навык, которым должен овладеть исследователь прежде, чем приступить к решению задачи.

4. Большинство известных средств моделирования клеточных автоматов разработаны для операционных систем семейства UNIX/Linux. По крайней мере, хорошие экземпляры существуют только для них. Предлагаемое программное обеспечение разработано для операционных систем семейства Windows NT[1]. Однако части, не связанные с пользовательским интерфейсом, не зависят от платформы.

5. Подавляющее большинство существующих продуктов не «моделируют» клеточные автоматы, а только «имитируют» их, так как не используют естественного параллелизма этих моделей, не поддерживают средств для осуществления параллельных или распределенных вычислений. При решении задач на кластерной платформе программное обеспечение CAME&L использует специализированный сетевой протокол CTP (Commands Transfer Protocol) [15, 16], разработанный автором. Поддержка многопроцессорной платформы также предусмотрена.

Проект CAME&L получил высокую оценку экспертов на международной конференции по компьютерным наукам (ICCS-2003), проходившей в Мельбурне и Санкт-Петербурге в 2003 году, а также на шестой международной конференции «Клеточные автоматы в науке и промышленности» (ACRI-2004), проходившей в Амстердаме в 2004 году.

В настоящей работе приводятся примеры решения различных задач с помощью пакета CAME&L.

Основные понятия и свойства пакета CAME&L

Программное обеспечение CAME&L можно разделить на три основные части.

1. Среда выполнения – приложение с богатым и дружественным пользовательским интерфейсом. Базовой абстракцией в ней является понятие «эксперимент», вычислительная задача, описанная в терминах клеточных автоматов. В данном контексте это понятие – синоним термина «документ».

Среда может работать на однопроцессорном компьютере, многопроцессорной системе или в вычислительном кластере (в том числе и через Internet). При этом продукт не требует никаких дополнительных инструментов для организации кластера, так как все необходимые возможности реализованы внутри него.

Среда имеет многодокументный интерфейс для того, чтобы управлять несколькими экспериментами одновременно, а также реализовывать межавтоматные взаимодействия в процессе моделирования.

Помимо этого она обеспечивает сохранение документов в формате XML (файлы с расширением «cml»). Для уменьшения размера этих файлов информация о состоянии решетки автомата может быть сжата внутри документа с помощью алгоритма BZip2.

2. Библиотека разработчика клеточных автоматов CADLib (Cellular Automata Development Library) – богатый набор C++-классов [17], которые служат для создания так называемых «компонентов», элементов решения задач посредством программного обеспечения CAME&L. Библиотека предоставляется для использования и расширения пользователями при разработке своих решений.

Помимо классов, она содержит функции, макроопределения и константы, делающие разработку компонентов настолько простой, насколько это возможно.

Настоящая работа представляет введение в решение задач с помощью библиотеки CADLib, при

[1] Имеются в виду такие операционные системы, как Windows NT 4, Windows 2000, Windows XP и т. д.

этом вопросы, касающиеся работы в среде, рассматриваться не будут.

3. Стандартные компоненты – базовый набор «кирпичиков», из которых могут быть построены решения пользовательских задач.

Каждый клеточный автомат представляется набором компонентов следующих четырех типов.

1. Решетка (grid) – осуществляет визуализацию решетки автомата и навигацию по клеткам.

2. Метрика (metrics) – сопоставляет клеткам координаты, вводит отношение соседства и функции вычисления расстояний между клетками.

3. Хранилище данных (datum) – обеспечивает хранение, обмен и преобразования состояний клеток, а также некоторые аспекты визуализации (соответствие между состоянием клетки и цветами, используемыми для ее отображения).

4. Правила (rules) – описывают итерацию, не только функцию переходов, но и метод разделения задачи на подзадачи, использование оптимизации вычислений, а также многое другое.

Идеология компонентов позволяет «собирать» автоматы с различной функциональностью, посредством незначительных изменений переходить от одной задачи к другой, пробовать решать одну и ту же задачу на разных решетках, с различными метриками и т. п. Кроме того, она предоставляет возможность распределять разработку программного обеспечения эксперимента между различными исполнителями.

Каждый компонент декларирует список собственных параметров, изменение значений которых обеспечивает настройку компонента.

Компонент, реализующий правила автомата, декларирует также список анализируемых параметров, характеризующих работу системы. Среда позволяет наблюдать изменение их значений в динамике, строить графики, файлы отчетов и т. п.

Для анализа таких параметров предусмотрен пятый тип компонентов – анализатор (analyzer).

В стандартный набор компонентов входит специализированный анализатор для построения графиков динамики производительности вычислений, что позволяет настраивать систему для достижения оптимальных результатов.

Реализация метрики в виде отдельного компонента позволяет использовать нестандартные системы координат, как, например, обобщенные координаты [7, 18], четыре вида которых входят в набор стандартных компонентов.

При постановке эксперимента пользователь может воспользоваться компонентами, входящими в базовый набор или разработать свои собственные.

С точки зрения разработчика каждый класс, реализующий компонент – потомок некого класса библиотеки CADLib или класса другого компонента. Иерархия классов компонентов восходит к классу `CAComponent`. От него наследуются классы, соответствующие всем пяти типам компонентов. Базовый класс решеток носит имя `CAGrid`, метрик – `CAMetrics`, хранилищ данных – `CADatum`, правил `CARules`, а анализаторов – `CAAnalyzer`.

Для того чтобы быть использованным в среде, компонент, а также три функции для его аутентификации, создания и удаления помещаются в динамическую библиотеку DLL (Dynamic-Link Library).

Аналогично, иерархия классов, реализующих параметры компонентов, восходит к классу `Parameter`. Анализируемые параметры компонента, реализующего правила автомата, с точки зрения реализации ничем не отличаются от обычных параметров.

Полная диаграмма классов библиотеки CADLib и стандартных компонентов приведена на рис. 1.

Параллелограммы с пунктирными границами обозначают шаблоны классов [17]. Серые фигуры показывают классы стандартных компонентов. Они распространяются вместе с исходным кодом, но формально в библиотеку CADLib не входят. Остальные параллелограммы обозначают обычные классы.

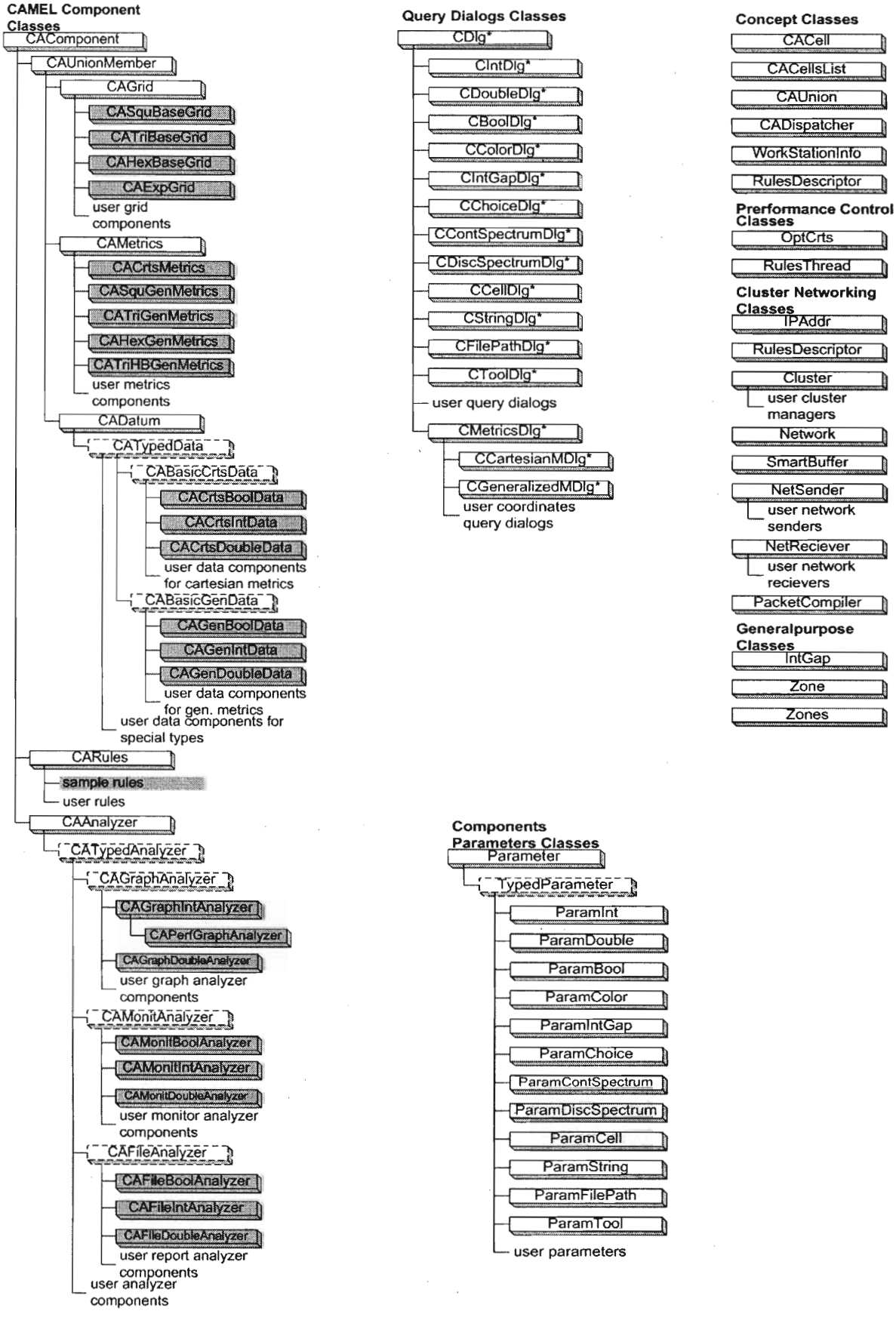
Большинство классов разрабатывалось так, чтобы они не зависели от платформы на уровне исходного кода. В них используются только стандартные библиотеки и библиотека STL (Standard Template Library) [17]. Однако некоторые классы, преимущественно, имеющие отношение к пользовательскому интерфейсу, имеют связи с библиотекой MFC (Microsoft Foundation Classes) [19]. Эти классы отмечены на диаграмме знаком «*». При переносе библиотеки CADLib на другую платформу их необходимо изменить для работы с соответствующей библиотекой.

Кроме того, на диаграмме специально показаны места, которые могут занять пользовательские классы в зависимости от их назначения.

Одним из ключевых типов данных является тип `CACell`, служащий для представления идентификаторов клеток решетки, как 64-разрядных целых чисел.

Необходимо отметить также чрезвычайно востребованный класс `Zone`, описывающий некоторую область на трехмерной решетке автомата. Экземпляр класса хранит шесть значений типа `CACell`, определяющих три отрезка вдоль трех осей пространства `[a1, b1]`, `[a2, b2]` и `[a3, b3]`. В случае необходимости описания зоны на двумерной решетке `a3` полагается равным `b3`. Для одномерного случая, помимо этого, `a2` присваивается `b2`. Кроме того, класс `Zone` реализует большой набор функций зональной алгебры.

Так как в задачи настоящей работы не входит подробный обзор классов библиотеки CADLib, а лишь демонстрация примеров разработки пользовательских компонентов правил клеточных автоматов, рассмотрим лишь необходимые для дальнейшего изложения классы `CAComponent` и `CARules`, а также приведем некоторые начальные сведения о параметрах и компиляции библиотек компонентов.



■ *Рис. 1. Диаграмма библиотеки CADLib*

Основы использования библиотеки CADLib

Класс CAComponent. Как отмечалось выше, данный класс является базовым для всех компонентов. Опишем его члены.

- public CAComponent() – конструктор;
- public ~CAComponent() – деструктор;
- protected int m_iType – переменная, хранящая тип компонента. Может принимать одно из следующих значений:

CT_COMPONENT – абстрактный тип, среда игнорирует такой компонент;
CT_GRID – решетка;
CT_DATUM – хранилище данных;
CT_METRICS – метрика;
CT_RULES – правила;
CT_ANALYZER – анализатор.
Значение присваивается в потомках.

- public inline UINT GetType() – функция возвращает тип компонента.

Далее описываются функции, предоставляющие информацию о компоненте. Их можно или нужно переопределять в наследниках.

- public virtual inline STRING GetName() – функция возвращает имя компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос COMPONENT_NAME(<имя>).
- public virtual inline STRING GetInfo() – функция возвращает описание компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос COMPONENT_INFO(<информация>).
- public virtual inline LPTSTR GetIconID() – функция возвращает идентификатор ресурса-иконки данного компонента. Если она возвращает NULL, то среда будет использовать иконку по умолчанию. Функция может быть переопределена в потомке, по умолчанию возвращает NULL. Для переопределения предусмотрен макрос COMPONENT_ICON(<идентификатор>).
- public virtual inline STRING GetRequire() – функция возвращает выражение, описывающее набор свойств, которыми должны обладать компоненты для того, чтобы быть совместимыми с данным (так называемые «условия совместимости»). Выражение принадлежит языку, определяемому следующей грамматикой:

```
<EXPRESSION> ::= (<EXPRESSION>)
<EXPRESSION> ::= !<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>^<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>|<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>&<EXPRESSION>
<EXPRESSION> ::= <TOKEN>
```

Каждая лексема <TOKEN> может представлять собой:

конкретное свойство, набор слов, разделенных точкой (например, «Data.bool» или «Metrics.2D.cartesian»). Каждое следующее слово уточняет предыдущее или, иными словами, является «подсвойством» предшествующего;

начало свойства и групповой символ «*» (например, «Metrics.2D.*»), обозначающий любое, в том числе и нулевое количество подсвойств;

начало свойства и групповой символ «?», обозначающий любое ненулевое количество подсвойств.

В качестве имен свойств и подсвойств могут быть использованы произвольные слова. При их проверке на совпадение регистр не учитывается.

Функция может быть переопределена в потомке, по умолчанию возвращает «*», что соответствует совместимости со всеми компонентами. Для переопределения предусмотрен макрос COMPONENT_REQUIRES(<выражение>).

- public virtual inline STRING GetRealize() – функция возвращает набор конкретных свойств, которые реализует настоящий компонент. Если свойств несколько, то они разделяются точкой с запятой. Функция может быть переопределена в потомке, по умолчанию возвращает пустую строку, что приводит к тому, что данный компонент будет совместим со всеми другими. Для переопределения предусмотрен макрос COMPONENT_REALIZES(<выражение>).

Далее описываются функции для поддержки работы с параметрами, наследниками класса Parameter. Средой обеспечивается возможность изменения их значений для настройки компонента.

- public virtual inline unsigned short GetParametersCount() – функция возвращает число параметров данного компонента. Она может быть переопределена в потомке, по умолчанию возвращает ноль.
- public virtual inline Parameter* GetParameter(unsigned short n) – возвращает указатель на *n*-й параметр компонента, где *n* принимает значение от нуля до значения, которое вернет функция GetParametersCount, уменьшенного на единицу. Функция может быть переопределена в потомке, по умолчанию возвращает NULL.

Для удобства реализации двух последних функций предусмотрены следующие макроопределения:

PARAMETERS_COUNT(<число_параметров>) для задания числа параметров;

BEGIN_PARAMETERS для того, чтобы начать так называемую «карту параметров»;

PARAMETER(n,<указатель_на_n-ый_параметр>) для добавления *n*-го параметра в карту;

END_PARAMETERS для того, чтобы закончить карту.

Например, если у компонента *N* параметров, носящих имена p_Par1, p_Par2, ..., p_ParN, то карта параметров будет иметь вид:

```
PARAMETERS_COUNT(N)
BEGIN_PARAMETERS
    PARAMETER(0,&p_Par1)
```

```
    PARAMETER(1,&p_Par2)
            ...
    PARAMETER(N-1,&p_ParN)
END_PARAMETERS
```

• public virtual void ParameterChanged(Parameter* pPar) – функция, вызываемая ядром, когда в среде произошло изменение значения некого параметра данного компонента. При этом указатель на изменившийся параметр передается в функцию. При инициализации компонента ей передается значение NULL. Функция может быть переопределена в потомке, по умолчанию ничего не делает.

Класс CARules. Данный класс является базовым для всех компонентов, реализующих правила клеточного автомата. Класс обладает широким спектром возможностей. Он включает в себя, в частности, средства для межавтоматных взаимодействий, кластерных вычислений и т. д. Опишем здесь только основные его члены:

• public CARules() – конструктор.

• protected CAUnion* m_pUnion – переменная хранит указатель на «союз компонентов», с которыми должны работать данные правила автомата. Союзом компонентов называется набор из совместимых друг с другом решетки, метрики и хранилища данных.

• public void SetUnion(CAUnion* pUnion) – функция устанавливает значение указателя на союз компонентов.

• public CAUnion* GetUnion() – функция возвращает значение указателя на союз компонентов.

• public EnvInfo* m_pEnvInfo – переменная хранит объект класса EnvInfo, служащего для описания платформы и окружения, в котором выполняются вычисления. Эта информация может быть проанализирована и учтена функциями класса CARules.

• public void SetEnvInfo(EnvInfo* pEnvInfo) – функция устанавливает описания платформы и окружения.

• public bool m_bFinalizeIfChanged – если значение этой переменной равно true, то после любого изменения состояния решетки пользователем будет произведена финализация эксперимента и перед следующим шагом потребуется повторная инициализация. Эта возможность оказывается полезной при использовании средств оптимизации производительности, кластерных вычислениях и т. д.

Далее описываются функции для управления итерациями, которые могут быть переопределены в потомках.

• public virtual bool SubCompute(Zone& z) – функция выполняет шаг автомата в области решетки, ограниченной значением первой координаты от z.a1 до z.b1, второй – от z.a2 до z.b2 и третьей – от z.a3 до z.b3. Эта функция является вспомогательной. Она вызывается функцией Compute при распараллеливании вычислений на многопроцессорной системе, например, с помощью вспомогательного класса RulesThread, а также при использовании кластеров.

Необходимо отметить, что идеологически правильно применять ее и в тех случаях, когда вычисления не распределяются.

Клеточные автоматы характеризуются тем, что результирующее состояние решетки не зависит от порядка обхода клеток или областей. При выполнении этого свойства вызов данной функции для множества непересекающихся зон, объединение которых покрывает всю решетку, приведет к тому же результату, что и вызов ее для единственной области, охватывающей всю решетку целиком.

Функция может быть переопределена в потомке, по умолчанию возвращает true.

Необходимо отметить, что, например, при использовании обобщенных координат [7, 18] даже для двумерных и трехмерных решеток область ограничивается только вдоль одной оси.

• public bool SubCompute() – функция вызывает предыдущую функцию, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных.

• public virtual bool Initialize() – функция выполняет инициализацию эксперимента перед его стартом или выполнением шага после финализации. Если функция возвращает true, то инициализация считается прошедшей успешно и эксперимент может быть начат. Она может быть переопределена в потомке, по умолчанию возвращает true.

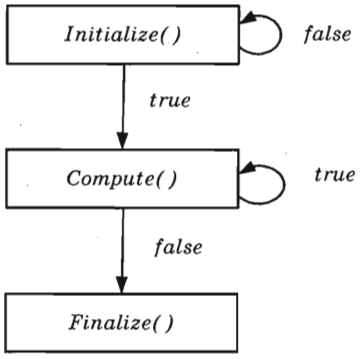
• public virtual bool Compute() – функция выполняет шаг автомата. Если функция возвращает true, то эксперимент может быть продолжен после данного шага. В противном случае он будет закончен и финализирован. Таким образом, можно задавать критерии окончания эксперимента. Функция может быть переопределена в потомке, по умолчанию вызывает функцию SubCompute, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных.

• public virtual void Finalize() – функция выполняет финализацию эксперимента. Она может быть переопределена в потомке, по умолчанию не делает ничего.

Порядок вызова последних трех функций при проведении эксперимента иллюстрирует рис. 2.

Кроме того, для управления состоянием решетки в классе предусмотрены следующие функции, которые могут быть переопределены в потомке.

• public virtual void Tool1(), public virtual void Tool2(), public virtual void Tool3() – функции реализуют три «пользовательских инструмента», которые могут быть вызваны из среды. Эти инструменты позволяют преобразовывать или анализировать состояние решетки. По



■ *Рис. 2. Схема вызова функций класса* CARules *при проведении эксперимента*

умолчанию функции отображают сообщения о том, что они не были переопределены.

В случае, если пользователю потребуется больше трех инструментов, можно использовать параметры-инструменты (объекты класса ParamTool), реализуя их внутри функции ParameterChanged.

Для определения анализируемых параметров, характеризующих эксперимент, предусмотрены две функции:

• public virtual inline unsigned short GetAnalyzableParametersCount() – функция возвращает число анализируемых параметров данного компонента правил. Она может быть переопределена в потомке, по умолчанию возвращает ноль.

• public virtual inline Parameter* GetAnalyzableParameter(unsigned short n) – возвращает указатель на n-й анализируемый параметр компонента, где n принимает значение от нуля до значения, которое вернет функция GetAnalyzableParametersCount, уменьшенного на единицу. Функция может быть переопределена в потомке, по умолчанию возвращает NULL.

Для удобства реализации двух последних функций, как и в случае обычных параметров, предусмотрены макроопределения:

APARAMETERS_COUNT(<число_параметров>) для задания числа анализируемых параметров;

BEGIN_APARAMETERS для того, чтобы начать карту анализируемых параметров;

END_APARAMETERS для того, чтобы закончить карту.

Параметры в карту добавляются с помощью описанного выше макроопределения PARAMETER.

Остальные функции и переменные, члены класса CARules, останутся за пределами рассмотрения в настоящей работе, так их общее количество слишком велико. Отметим лишь, что некоторые из них будут обсуждаться в контексте организации кластерных вычислений.

Базовые сведения о параметрах. В библиотеку CADLib входит набор классов, реализующих параметры различных типов. Как отмечалось выше, все они являются потомками класса Parameter. Однако у них есть еще один общий предок, шаблон TypedParameter, непосредственный наследник класса Parameter. Этот шаблон, помимо прочего, реализует две следующие функции (будем считать type аргументом шаблона, определяющим тип параметра):

• virtual inline type Get() – функция возвращает значение параметра;

• virtual inline bool Set(type value, bool notify=true) – функция устанавливает значение параметра равным value. Если флаг notify равен true, то при этом будет вызвана функция ParameterChanged компонента, владеющего данным параметром, иначе – не будет. Функция возвращает true, если новое значение корректно, и оно было присвоено. В противном случае она возвращает false. Функция должна быть переопределена в потомке.

В библиотеку CADLib также входит набор классов, реализующих окна-диалоги для запроса значений параметров всех предусмотренных типов из среды.

В таблице приведены имена классов, реализующих стандартные параметры, имена классов соответствующих им диалогов, а также словесные описания типов параметров.

Необходимо также отметить, что у класса каждого параметра имеется конструктор, который в качестве аргументов принимает имя параметра, его описание, начальное значение и указатель на компонент, которому принадлежит данный параметр. При этом конструктора по умолчанию у параметров нет, поэтому они должны создаваться в конструкторе владеющего ими компонента.

Разработка библиотеки, содержащей компонент. Как отмечалось выше, компонент должен быть помещен в динамическую библиотеку DLL (Dynamic-Link Library). Помимо класса, реализующего компонент, в ней должны находиться три функции.

1. EXPORTIT[1] STRING[2] GetCompatibilityInfo() – функция аутентификации библиотеки и компонента. Возвращает строку вида «CAMEL Component| <версия_ядра>|<тип_компонента>», где

<версия_ядра> – версия ядра программного обеспечения, с которой совместим данный компонент, например «1.0». Совместимость «вниз», по возможности, должна обеспечиваться;

<тип_компонента> – строка «Grids», «Data», «Metrics», «Rules» или «Analyzers», в зависимости от типа компонента, содержащегося в библиотеке.

[1] Использование макроопределения EXPORTIT обеспечивает экспортирование функции из библиотеки.

[2] Макроопределение STRING обозначает указатель на первый символ строки с завершающим нулем.

■ *Стандартные классы параметров, соответствующие им классы диалогов и описания их типов*

Класс параметра	Класс диалога	Описание типа параметра
ParamInt	CIntDlg	Целое число
ParamDouble	CDoubleDlg	Число с плавающей точкой двойной точности
ParamBool	CBoolDlg	Булева величина
ParamColor	CColorDlg	Дескриптор цвета
ParamIntGap	CIntGapDlg	Отрезок с целочисленными границами
ParamChoice	CChoiceDlg	Значение из конечного множества вариантов
ParamContSpectrum	CContSpectrumDlg	Непрерывный цветовой спектр
ParamDiscSpectrum	CDiscSpectrumDlg	Дискретный цветовой спектр
ParamCell	CCellDlg	Идентификатор клетки решетки
ParamString	CStringDlg	Строка
ParamFilePath	CFilePathDlg	Путь к файлу
ParamTool	CToolDlg	Инструмент

Для удобства определения этой функции предусмотрены макроопределения COMPATIBLE_GRID(<версия_ядра>), COMPATIBLE_METRICS(<версия_ ядра>), COMPATIBLE_DATUM(<версия_ядра>), COMPATIBLE_RULES(<версия_ ядра>) и COMPATIBLE_ANALYZER(<версия_ядра>).

2. Функция создания компонента, возвращающая указатель на новый экземпляр. В зависимости от типа она может иметь вид:

```
EXPORTIT CAGrid* CreateGrid();
EXPORTIT CAMetrics* CreateMetrics();
EXPORTIT CADatum* CreateDatum();
EXPORTIT CARules* CreateRules();
EXPORTIT CAAnalyzer* CreateAnalyzer().
```

3. Функция удаления компонента, освобождающая память, занимаемую им. В зависимости от типа функция удаления может иметь вид:

```
EXPORTIT DestroyGrid(CAGrid* pGrid);
EXPORTIT DestroyMetrics(CAMetrics* pMetrics);
EXPORTIT DestroyDatum(CADatum* pDatum);
EXPORTIT DestroyRules(CARules* pRules);
EXPORTIT DestroyAnalyzer(CAAnalyzer* pAnalyzer).
```

Для удобства реализации двух последних функций предусмотрены макроопределения GRID_COMPONENT(<имя_класса>), METRICS_COMPONENT(<имя_класса>), DATUM_COMPONENT(<имя_класса>), RULES_COMPONENT(<имя_класса>) и ANALYZER_COMPONENT(<имя_класса>).

В результате, например, при написании библиотеки, содержащей класс компонента решетки CAMyGrid, совместимого с ядром версии 1.0, помимо реализации самого класса достаточно добавить две строки:

```
COMPATIBLE_GRID(1.0)
GRID_COMPONENT(CAMyGrid)
```

Для написания компонентов рекомендуется использовать компилятор и среду разработки Microsoft Visual C++ 6, так как пакет CAME&L создавался с помощью этого средства. Таким образом, весь набор примеров, стандартных компонентов, также ориентирован на него.

Для реализации нового компонента следует создать в среде разработки проект, представляющий собой динамическую библиотеку (выбрать в меню «File» | «New...» | «Projects» | «Win32 Dynamic-Link Library», указать имя и директорию). В появившемся после этого мастере выбрать вариант «An empty DLL project». После этого необходимо подключить к проекту библиотеку CADLib (выбрать в меню «Project» | «Settings» | «Link» и в категории «General» в поле «Object/library modules» добавить «CADLib.lib», а в категории «Input» в поле «Additional library path» добавить «%CAMEL_PATH%[1]\Lib»). Теперь можно приступать к разработке, создавать новые файлы, реализующие компонент, а также его иконку.

Однако для того, чтобы избавить пользователя от необходимости настраивать среду для каждого нового компонента, был разработан скрипт ccomp.sh, служащий для копирования существующих компонентов. Под операционной системой

[1] Будем обозначать через %CAMEL_PATH% путь к установленному пакету CAME&L.

семейства Windows он может быть выполнен с помощью пакета Cygwin [20], средства эмуляции оболочки операционных систем семейства UNIX/Linux.

Данный скрипт доступен с сайта [7]. Перед использованием его необходимо поместить в директорию «%CAMEL_PATH%\CompsSrc\ <тип_компонента>», в которой обычно размещаются исходные коды компонентов.

Скрипт поддерживает следующие параметры из командной строки:

```
ccomp.sh <директория_из_которой_копировать> <директория_в_которую_ копировать> <опции>
```

Поддерживаются следующие опции (в алфавитном порядке):

`-c <имя_класса>` – установить соответствующее имя класса для копии компонента;

`-i <описание>` – установить соответствующее описание для копии компонента;

`-n <имя>` – установить соответствующее имя[1] для копии компонента.

Для получения информации о параметрах командной строки можно также воспользоваться встроенной в скрипт помощью, вызвав

```
ccomp.sh -h
```

Например, если в директории %CAMEL_PATH%\CompsSrc\Rules\MyRules1 находится компонент, который можно взять за основу нового компонента правил, то следует воспользоваться описанным выше скриптом, поместив его в директорию %CAMEL_PATH%\CompsSrc\Rules и вызвав

```
ccomp.sh MyRules1/ MyRules2/ -c CAMyRules2 -i "This is my second rules component" -n MyRules2
```

В результате компонент будет скопирован в директорию %CAMEL_PATH%\CompsSrc\Rules\MyRules2. В копии будет установлено имя класса `CAMyRules2`, описание «`This is my second rules component`» и имя компонента «`MyRules2`».

Описанный выше скрипт для копирования компонентов позволяет существенно увеличить скорость разработки новых решений.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту № 05-07-90086 «Разработка среды и библиотеки "CAME&L" для организации параллельных и распределенных вычислений на основе клеточных автоматов».

Литература

1. **MacDonald N., Minty E., Harding T., Brown S.** Writing Message Passing Parallel Programs with MPI – Edinburgh Parallel Computing Center. – The University of Edinburgh, 1996.
2. **Geist A., Beguelin A., Dongarra J.** and oth. PVM: Parallel Virtual Machine. A User Guide and Tutorial for Networked Parallel Computing. – MIT Press, Cambridge, 1994.
3. **Foster I., Kesselman C., Tuecke S.** The Anatomy of Grid // http://www.globus.org, 2001.
4. **Foster I., Kesselman C., Nick J. M., Tuecke S.** The Physiology of Grid // http://www.globus.org, 2002.
5. **Foster I.** What is a Grid? A Three Point Checklist // http://www.globus.org, 2002.
6. **Foster I., Kesselman C.** Globus: A Metacomputing Infrastructure Toolkit // http://www.globus.org, 2002.
7. **Сайт** «CAMEL Laboratory» – http://camellab.spb.ru.
8. **Naumov L.** CAME&L – Cellular Automata Modeling Environment & Library // Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. – Springer–Verlag, 2004.
9. **Наумов Л.** CAME&L – Среда моделирования и библиотека разработчика клеточных автоматов // Труды XI Всероссийск. научно-метод. Конф. Телематика'2004. – Т. 1. – СПб.: СПбГУ; ИТМО, 2004.
10. **фон Нейман Дж.** Теория самовоспроизводящихся автоматов. – М.: Мир, 1971.
11. **Тоффоли Т., Марголус Н.** Машины клеточных автоматов. – М.: Мир, 1991.
12. **Wolfram S.** A New Kind of Science. – IL: Wolfram Media, 2002.
13. **Sloot P.M.A., Chopard B., Hoekstra A.** Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. – Springer-Verlagю, 2004.
14. **Sarkar P.** A Brief History of Cellular Automata // ACM Computing Surveys. – Vol. 32. –№ 1. – 2000.
15. **Наумов Л.** CTP (Commands Transfer Protocol) – Сетевой протокол для высокопроизводительных вычислений // Труды XI Всероссийск. научно-метод. конф. Телематика'2004. – Т. 1. – СПб.: СПбГУ; ИТМО, 2004.
16. **Naumov L.** Commands Transfer Protocol (CTP) – New Networking Protocol for Parallel or Distributed Computations // http://www.codeproject.com/internet/ctp.asp, 2004.
17. **Страуструп Б.** Язык программирования C++. – 3-е изд. СПб.: Невский диалект, 1999.
18. **Naumov L.** Generalized Coordinates for Cellular Automata Grids // Computational Science – ICCS 2003. – P. 2. – Springer-Verlag, 2003.
19. **Мешков А., Тихомиров Ю.** Visual C++ и MFC. – 2-е изд. – СПб.: БХВ–Санкт-Петербург, 2000.
20. http://www.cygwin.com.

[1] Здесь имеется в виду имя, под которым компонент виден в среде, возвращаемое его функцией `GetName()`, а не имя класса.