О моделировании сенсорных сетей средствами высокого уровня

Дорошенко А.Е.
Институт программных систем НАНУ, Киев, 03187, просп. Глушкова, 40/5, тел. 526-15-38,
dor@isofts.kiev.ua
Жереб К.А
Киевское отделение Московского физико-технического института,
03650 Киев. просп. Глушкова, 40/1
cotan@mail.ru
Шевченко Р.С.
ООО «Градсофт», г. Киев, ул. Магнитогорская 1б, к. 408
Ruslan@Shevchenko.kiev.ua

Аннотация: Развитие полупроводниковых технологий сделало возможным создание беспроводных сенсорных сетей. Для таких сетей необходимо создание систем моделирования, позволяющих выбирать оптимальные протоколы для каждой прикладной задачи. В работе предложен подход к моделированию сенсорных сетей, объединяющий возможности среды визуального моделирования AnyLogic и системы символьных вычислений TermWare. Для демонстрации этого подхода разработан прототип системы моделирования. Рассматривается пример декларативного описания протокола – простой синхронный протокол доступа к каналу.

Abstract: Recent technological advances enabled the creation of Wireless Sensor Networks (WSN). There is a need for simulation systems, which should allow users to choose the best protocols for each applied problem. We present an approach to sensor network simulation, based on AnyLogic – visual simulation environment and TermWare – symbolic computation system. A prototype simulation system implementing this approach is described. As an example of declarative protocol description we use a simple synchronous media access protocol.

Введение

В настоящее время появляется новое перспективное направление, связанное с применением беспроводных сенсорных сетей. Развитие полупроводниковых технологий позволяет создавать малогабаритные устройства, совмещающие функции измерения определенных параметров окружающей среды, предварительной обработки результатов и пересылки данных для дальнейшей обработки. Такие устройства могут иметь низкую стоимость и работают длительное время (до 10 лет) от встроенного источника питания. Эти особенности позволяют использовать сенсорные сети для различных научных и прикладных целей. Исследовательские проекты показывают возможность применения сенсорных сетей для исследования движения ледников [2], обнаружения радиоактивных веществ [3], локализации снайперов в городских условиях [4].

Реализованные и проектируемые сенсорные сети имеют многочисленные различия, как в области применения, так и в использованных технических решениях. Тем не менее, можно выделить основные характеристики, свойственные большинству систем [1]. Типичная сенсорная сеть состоит из большого количества простейших устройств для сбора информации (мотов) и нескольких более сложных устройств для обработки информации (базовых станций). Каждый мот производит периодические измерения параметров окружающей среды, осуществляет первичную обработку полученной информации и пересылает ее на базовую станцию. На базовой станции производится дальнейшая обработка информации, промежуточное хранение и передача ее для дальнейшей обработки. Моты используют маломощные передатчики, поэтому обычно передача происходит не напрямую, а с использованием промежуточных мотов.

Типичные сценарии использования сенсорных сетей накладывают определенные ограничения на устройство и функционирование мотов. В частности, устройства должны быть малогабаритными, достаточно дешевыми и должны работать в течение длительного времени с использованием автономного источника питания. Поэтому для мотов характерны существенные ограничения вычислительных ресурсов (процессор, память) и необходимость очень экономно расходовать энергию. Для базовых станций эти ограничения не столь актуальны.

Особые сложности возникают при разработке протоколов сетевого взаимодействия мотов. Работа приемника и передатчика требует наибольших затрат энергии, поэтому эти устройства должны включаться достаточно редко. Это делает невозможным применение стандартных схем функционирования беспроводных сетей (например, постоянное прослушивание эфира с целью предотвращения коллизий). Кроме того, специфика использования сенсорных сетей определяет и другие отличия. Обычно нет необходимости в передаче информации от одного мота другому, все данные должны доставляться на ближайшую базовую станцию. Надежность при передаче достигается за счет подтверждения каждого пакета данных на каждом этапе передачи, а не путем установления надежного канала между конечными узлами при передаче. Наконец, моты, находящиеся ближе к базовой станции, могут не просто отсылать пакет данных дальше, а агрегировать данные с результатами собственных измерений. Поэтому для сенсорных сетей необходима разработка специфичных эффективных сетевых протоколов (в особенности это касается доступа к каналу и маршрутизации).

На сегодняшний день разработано достаточно много протоколов маршрутизации, учитывающих специфику работы сенсорных сетей. В работе [5] предложена классификация протоколов маршрутизации, основанная на

выделении следующих групп протоколов: 1) протоколы, основанные на данных, 2) иерархические протоколы и 3) протоколы, основанные на положении мотов. Для разных прикладных задач необходимо использование разных подходов к маршрутизации, то есть должен производиться выбор наиболее подходящего протокола. В некоторых случаях может быть оправдана совместная разработка программной и аппаратной частей мота для достижения максимальной эффективности работы сети [6].

Среди всех исследований, проводимых в области беспроводных сенсорных сетей, особое внимание привлекает проект TinyOS [7], [8]. В рамках этого проекта реализован аналог операционной системы для мотов. TinyOS реализует основные низкоуровневые функции, в том числе управление сенсором и передачу данных по сети. Для программирования приложений используется высокоуровневый язык nesC [9] – расширение языка C. Этот язык позволяет использовать многие возможности, например разбиение приложения на компоненты и событийный подход к программированию. Кроме того, TinyOS содержит средства для разработки стандартных приложений без программирования TASK (Tiny Application Sensor Kit) [10].

При разработке сенсорных сетей большое значение имеют системы моделирования [11]. Системы моделирования сенсорных сетей позволяют разрабатывать аппаратное и программное обеспечение для мотов со значительно меньшими затратами, чем в случае использования реальных устройств. Также они позволяют выбирать оптимальные решения для конкретной прикладной задачи, определять параметры работы сети (задержки при доставке пакета, расход энергии, надежность). Но для получения корректных результатов такие системы должны моделировать низкоуровневые детали работы сети, что усложняет их реализацию.

В проект TinyOS входит система моделирования TOSSIM [12]. Она позволяет моделировать работу произвольной программы для сенсорной сети, написанной на языке nesC. Это позволяет разработать приложение с использованием TOSSIM и потом использовать его в реальной сети. В системе TOSSIM поведение мотов моделируется достаточно подробно – например, моделируется передача каждого бита по радиоканалу. Основным недостатком этой системы моделирования является тесная привязка к TinyOS, что не позволяет моделировать сенсорные сети, не основанные на этой платформе.

Существуют также другие системы моделирования сенсорных сетей, в которых уделяется внимание различным аспектам моделирование. Некоторые системы (например, ATEMU [13]) основаны на детальном воспроизведении особенностей функционирования аппаратной части мотов. В других системах учитываются требования масштабируемости (например, SWAN [14] и TOSSF [15] – масштабируемая система моделирования для TinyOS) и компонентной организации, позволяющей легко модифицировать систему (примером может служить SENS [16], а также SWAN). Проект Em* [17] предлагает единую платформу для моделирования, разработки и сопровождения сенсорных сетей. Существуют модификации универсальных систем моделирования сетевых протоколов, адаптированные для учета особенностей сенсорных сетей (NS-2 [18], J-Sim[19]).

В данной работе предложен подход к моделированию сенсорных сетей, объединяющий возможности среды визуального моделирования AnyLogic [20], [21] и системы символьных вычислений TermWare [22]. Большая часть функциональности, необходимой для моделирования, реализовывается в среде AnyLogic, с использованием визуальных средств. Система TermWare используется для декларативного описания протоколов. Декларативное описание протоколов позволяет не только упростить задание протоколов для системы моделирования, но и использовать дополнительные средства для анализа протокола. Например, возможно формальное доказательство корректности протокола (существуют верификаторы для различных протоколов, например, SPIN [23]). Для демонстрации данного подхода разработан прототип системы моделирования, состоящий из базовой модели в AnyLogic, описания протоколов на языке TermWare и компонента связи, реализованного на Java. В качестве примера декларативного описания протокола рассмотрен простейший синхронный протокол доступа к каналу.

1. Среда AnyLogic

Среда AnyLogic [20], [21] позволяет использовать различные парадигмы моделирования (непрерывное и дискретно-событийное моделирование, многоагентный подход), а также комбинировать их. Важными преимуществами системы являются:

- Объектно-ориентированный подход к моделированию
- Визуальная среда разработки с богатой функциональностью
- Возможность использования языка Java

В системе AnyLogic введено понятие *активного объекта (АО)* – расширение классических объектов, которые используются в объектно-ориентированных языках программирования. Как и обычные объекты, активные объекты могут иметь свойства и методы (хотя методы редко используются для связи между объектами). В AnyLogic активный объект может включать также другие элементы:

- Параметры – особые свойства, определяющие параметры модели
- Внешние переменные – используются для связи двух АО, когда изменение переменной в одном объекте автоматически меняет ее значение в другом
- Порты – используются для обмена сообщениями между АО
- Стейтчарты (диаграммы состояний) – задают переходы между состояниями АО

- Таймеры – позволяют совершать периодические действия, задавать временные задержки
- Математические функции – задают функцию с использованием математических формул

Название «активный объект» оправдано тем, что каждый объект в AnyLogic является автономной сущностью, полностью определяющей свое поведение. Так, каждому активному объекту соответствует свой поток управления (это автоматически обеспечивается средой AnyLogic). Связь между активными объектами производится не через вызов методов, а с помощью обмена сообщениями (и внешних переменных). В рамках многоагентной парадигмы, каждый активный объект можно рассматривать как независимого агента. Поэтому система AnyLogic хорошо подходит для реализации моделей, использующих многоагентный подход.

Каждый активный объект может содержать другие активные объекты, при этом определяя связи между ними. Можно включать как один экземпляр активного объекта, так и реплицированный объект – фактически массив объектов. Модель в AnyLogic представлена иерархией активных объектов. Существует один корневой объект, содержащий (прямо или косвенно) все активные объекты, используемые в модели.

Разработка модели в AnyLogic производится в визуальной среде. Создание активных объектов, их элементов, связей между активными объектами не требует написания кода – все эти операции имеют наглядное графическое представление. Кроме того, есть возможность создания анимаций, демонстрирующих состояние объекта. Можно создать анимацию для каждого АО, при этом для сложных объектов анимация строится на основании анимаций их компонентов.

Если стандартных средств AnyLogic не хватает для построения модели (или их использование неудобно), есть возможность использования языка Java. В простейшем случае, это сводится к описанию действий, совершаемых при переходе в другое состояние, срабатывании таймера или приходе сообщения. Кроме того, можно добавлять собственный код на Java к активному объекту, а также использовать сторонние библиотеки. Это делает систему AnyLogic легко расширяемой.

Возможности системы AnyLogic облегчают реализацию модели сенсорной сети. Так, поведение сети определяется поведением отдельных мотов, поэтому есть смысл применять многоагентный подход к моделированию, который поддерживается в AnyLogic. Система сообщений и таймеров позволяет моделировать взаимодействие мотов с учетом временных задержек. Кроме того, возможность расширения модели с помощью внешних библиотек Java также оказывается очень удобной.

2. Система TermWare

Система TermWare [22] нацелена на разработку высокодинамичных прикладных систем, к которым предъявляются повышенные требования как к встроенному интеллекту для обеспечения интерактивности разрабатываемых систем, так и удешевлению разработки, сокращению сроков проектирования и улучшения характеристик повторного их использования (реинжиниринга) и сопровождаемости. Она отличается от большинства других систем этого класса как назначением и семантикой используемых средств, так и технологией их реализации. Язык TermWare не является универсальным языком программирования в том смысле, что он не предназначен для написания полнофункциональных программных систем. Это координационный язык-оболочка, предназначенный для написания предметно-ориентированных частей приложений, встраиваемых в прикладную систему для реализации функций взаимодействия этой системы с программным окружением.

Для системы TermWare характерны следующие особенности:

- Использование декларативного описания
- Реализация только требуемой части приложения, а не всего приложения
- Ориентация на тесное взаимодействие с другими частями приложения

TermWare оформлена как библиотека языка Java, встраиваемая в программное приложение. Для разработчика система термов выглядит как совокупность экземпляров класса ITermSystem, с возможностью управления набором правил и редуцирования термов. База фактов также может содержать императивные элементы, описанные как Java классы. Таким образом, имеется возможность использовать декларативную модель программирования в программных комплексах, основанных на Java-платформе в тех случаях, когда это необходимо. Разработчик может использовать термальную систему как программный агент, встраиваемый в общую инфраструктуру программной системы.

Сами наборы правил описаны на языке TermWare и хранятся в отдельных файлах. Основой языка являются термы, то есть выражения вида $f(x_1,...,x_n)$. В качестве атомарных термов используются переменные (которые записываются в виде $identifier), а также константы определенных типов данных (числовых, логических, строковых и атомарного – неизменяемые строки). Для упрощения записи и восприятия используются сокращения для многих термов, например *x+y* – для *plus(x,y); x ? y: z* – для *ifelse(x,y,z)* и другие. Весь набор правил является термом, записанным с использованием этих сокращений.

Набор правил содержит сами правила, а также дополнительную информацию (название системы правил, используемая база фактов, применяемая стратегия). Типичное правило в TermWare записывается следующим образом:

source [condition] → destination [action]

Здесь используются 4 терма:

- source – входной образец
- destination – выходной образец
- condition – условие, определяющее применимость правила
- action –действие, выполняемое при срабатывании правила

Выполняемые действия и проверяемые условия в основном являются вызовами методов в БД фактов. Таким образом, осуществляется связь между правилами на языке TermWare и кодом на Java. Кроме того, возможно написание собственной стратегии, определяющей порядок применения правил.

В модели сенсорной сети, система TermWare может быть использована для декларативного описания протоколов взаимодействия. При этом описывается только сам протокол, вся функциональность, необходимая для моделирования, содержится в дополнительных компонентах, написанных на Java.

3. Общее описание.

Реализованный прототип системы моделирования состоит из трех основных частей:

- Базовая модель, реализованная средствами AnyLogic
- Протоколы, записанные в виде правил TermWare
- Компонент связи, реализованный на Java

Базовая модель реализует основные элементы сенсорных сетей (моты, базовая станция, их аппаратные и программные компоненты, радиоканал, окружающая среда). Все элементы представлены в виде активных объектов (АО) среды AnyLogic.

Для некоторых элементов, например, сетевого и канального уровня сетевого стека, необходимо использовать различные протоколы. Вместо реализации их методами AnyLogic, используется запись протокола в виде системы правил на языке TermWare. При этом становится возможным независимое изменение базовой модели и протоколов, причем изменение протоколов возможно даже во время проведения экспериментов с моделью, при запущенной модели в AnyLogic.

Для связи базовой модели и протоколов реализован компонент связи (КС). Базовая модель не работает с правилами напрямую – весь обмен информацией происходит через компонент связи. В частности, модель в AnyLogic не содержит непосредственных ссылок на объекты TermWare, вызовов методов и других средств взаимодействия. Это упрощает модель, скрывая особенности реализации TermWare, и позволяет использовать различные версии TermWare без изменения в модели AnyLogic.

Связь между базовой моделью и протоколами (системой правил) реализована на уровне активных объектов (а не всей модели). Активные объекты, использующие поддержку TermWare, связаны с компонентами связи, которые работают с правилами. Каждый активный объект связан с отдельным экземпляром КС, который использует соответствующую протоколу систему правил. В данном прототипе один АО может использовать только один набор правил; однако это ограничение не является существенным.

Общая архитектура прототипа системы моделирования показана на рис. 1 .

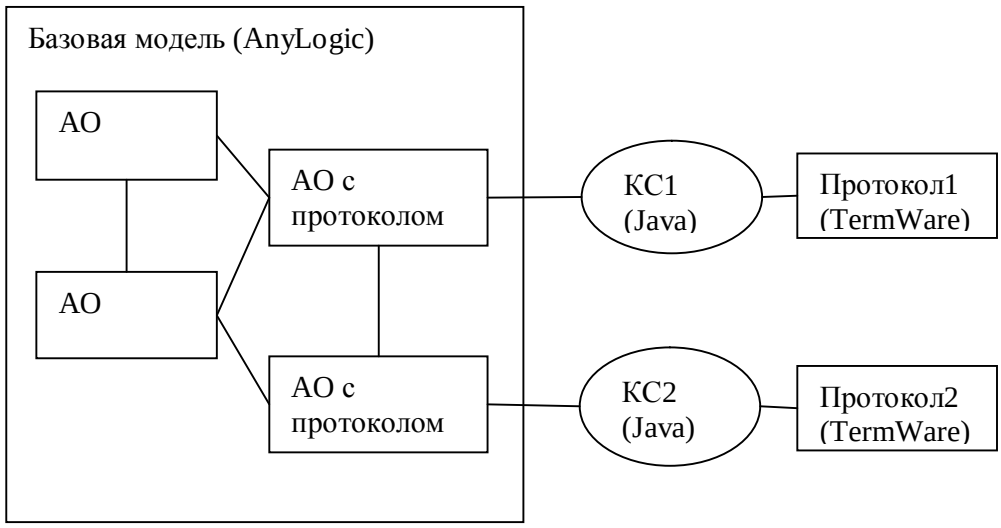

Рис. 1. Общая архитектура системы моделирования

4. Базовая модель.

Базовая модель реализует абстракции основных элементов моделируемой сенсорной сети. Каждый такой элемент представлен в виде активного объекта AnyLogic. Модель имеет иерархическую структуру, что позволяет рассматривать ее на разных уровнях.

Наиболее общее представление модели содержит множество сенсоров, а также среду, с которой они взаимодействуют (рис. 2). Среда представлена двумя независимыми активными объектами:

- Radio – моделирует радиоканал (содержащий все сигналы, посылаемые мотами)
- Phenomena – моделирует множество явлений, которые должны фиксироваться сенсорами

Сами сенсоры представлены реплицированным активным объектом mot*, содержащим произвольное количество экземпляров АО Mot. Количество сенсоров является параметром, задаваемым в свойствах модели. Базовая станция представлена тем же активным объектом, при этом отличия от обычного мота задаются изменением параметров.

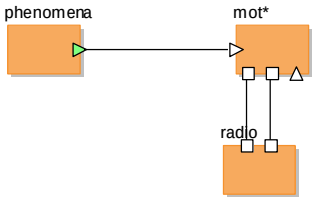

Рис. 2. Наиболее общее представление системы

Активные объекты Radio и Phenomena имеют достаточно простую структуру, поскольку большая часть функциональности модели реализована в активном объекте Mot. На рис. 3 показаны основные компоненты мота, как аппаратные, так и программные.

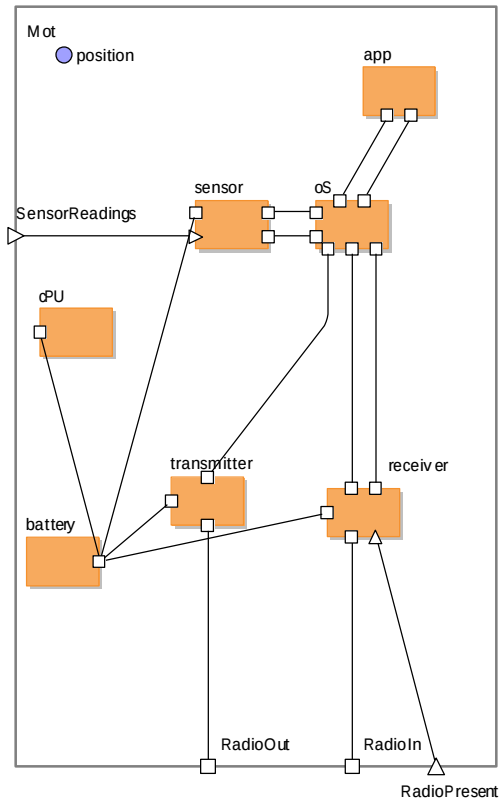

Рис. 3. Компоненты мота.

В модели рассматриваются следующие компоненты:

- Аппаратные:
- CPU – процессор мота, необходимый для выполнения всех программных компонент
- Sensor – блок измерительных устройств, установленных на моте. С помощью параметров этого активн ого объекта можно задавать, какие именно величины могут быть измерены мотом (из числа представленных в АО Phenomena)
- Transmitter – передатчик мота, используемый для передачи сообщений через радиоканал.
- Receiver – приемник мота, позволяющий слушать канал

• Battery – батарея, определяющая запас энергии мота. При работе всех остальных устройств расходуется энергия. В прототипе системы моделирования реализован простейший механизм затрат энергии (для каждого устройства постоянные затраты в единицу времени, энергия расходуется только во включенном состоянии).

• Программные

• App – прикладная программа, выполняемая на моте. В прототипе системы моделирования реализована программа, которая считывает показания сенсоров и пересылает их на базовую станцию, если произошли изменения.

• OS – операционная система мота, представляющая различные службы прикладной программе. В реализованном прототипе доступны две службы: считывание показания сенсоров и посылка сообщений на базовую станцию. Также этот компонент управляет работой аппаратных устройств.

Основные компоненты мота определяют представление системы, удобное для исследования высокоуровневых свойств. Например, на этом уровне возможно наблюдение за расходом энергии, временем доставки пакета на базовую станцию и другими параметрами, определяющими стабильность системы. При этом низкоуровневые детали функционирования сенсорной сети (такие, как алгоритм маршрутизации) остаются скрытыми от пользователя.

Самый глубокий уровень детализации представлен внутренней структурой компонента OS (рис. 4). Данный компонент содержит два встроенных активных объекта, представляющих программную часть сетевого стека:

• NETLayer – сетевой уровень. Определяет маршрутизацию пакетов

• MACLayer – уровень доступа к каналу. Определяет синхронизацию доступа к радиоканалу

В обоих компонентах используются протоколы, реализованные в виде набора правил TermWare.

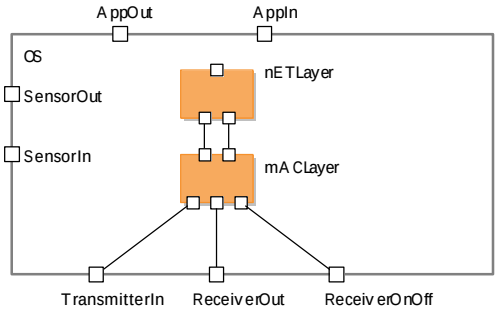

Рис. 4. Компонент OS

Связь между активными объектами в рамках модели осуществляется с помощью сообщений. При этом используется средства AnyLogic для графического представления связей (направления передачи сообщений). Основные сообщения, используемые в системе:

• OSServiceMessage – сообщение, используемое для предоставления сервисов операционной системы

• NetworkMessage – сообщение, используемое при описании перемещение пакета данных между мотами или компонентами в пределах мота

• BatteryMessage – сообщение о расходе энергии устройством.

Также используется другая возможность, предоставляемая AnyLogic для связи между активными объектами – связанные переменные. Например, связь между активными объектами Phenomena и Sensor реализована с помощью связанной переменной SensorReadings. При этом значения соответствующей переменной в блоке измерителей всегда совпадает со значением, определяемым явлением.

Основные параметры системы задаются изменением параметров соответствующих активных объектов. Так, число мотов есть параметр AO Main, потребление энергии устройством – параметр активного объекта, соответствующего устройству (CPU, Sensor, Transmitter, Receiver), периодичность измерений – параметр AO App. Некоторые важные параметры, используемые сразу в нескольких активных объектах, вынесены в корневой объект Main. К таким параметрам относится, например, пропускная способность канала – используется как в Transmitter, так и в Radio. В качестве одного из параметров для активных объектов, поддерживающих протоколы, используется название файла, содержащего систему правил.

5. Описание протоколов – общий подход

Описание протоколов работы мотов в сети (маршрутизация, доступ к каналу) можно реализовать средствами AnyLogic в соответствующих активных объектах. Среда AnyLogic не предоставляет средств для непосредственного описания протоколов, хотя можно использовать стандартные средства – стейтчарты, таймеры и обмен сообщениями.

В реализованном прототипе системы моделирования использован другой подход. Описания протоколов вынесены из соответствующих объектов и реализованы в виде систем правил на языке TermWare. Такой подход имеет определенные преимущества:

• Протокол описан на декларативном языке. По сравнению с использованием Java это позволяет сократить объем записи протокола

• Все описание протокола содержится в одном месте (в файле с системой правил). При реализации протокола средствами AnyLogic части описания будут реализованы в виде переходов в стейтчарте, операций таймеров, методов обработки сообщений, а также дополнительного кода активного объекта. При этом затруднено понимание протокола и его модификация

• Возможно изменение протокола (системы правил), даже во время выполнения модели.

• Описание протокола на TermWare можно использовать для его анализа различными средствами. В частности, возможно автоматическое формальное доказательство корректности протокола. Также можно генерировать код на C для выполнения на устройствах.

К недостаткам такого подхода следует отнести большую трудоемкость при реализации (из-за необходимости реализации дополнительного компонента связи), а также текстовое представление информации (в отличие от среды AnyLogic). Но компоненты связи для разных протоколов практически идентичны, что позволяет реализовать такой компонент один раз и в дальнейшем использовать его с незначительными изменениями. Реализация самих правил при этом не очень сложна, в силу описанных преимуществ. Возможно также создание графического редактора правил, как для произвольных правил в TermWare, так и специализированного, для описаний протоколов.

В реализованном прототипе системы моделирования каждый протокол (система правил) связан с одним активным объектом и фактически определяет его поведение (взаимодействие с другими активными объектами). Поэтому протокол должен определять реакцию на сообщения от других активных объектов (реактивное поведение), а также необходимость самостоятельной посылки сообщений (проактивное поведение). Для реализации проактивного поведения используются динамические таймеры – еще одно средство, предоставляемое средой AnyLogic.

Общую структуру протокола можно представить в следующем виде: протокол является системой правил, описывающих реакцию на определенные события. Возможны два вида событий:

• Приход сообщения

• Срабатывание таймера

Реакция на эти события является комбинацией следующих элементарных действий:

• Посылка сообщения

• Установление таймера

• Изменение внутреннего состояния

Все эти действия несложно записываются в виде правил TermWare. При этом используется БД фактов, реализованная в компоненте связи.

Компонент связи является обычной библиотекой классов на Java. Он содержит два основных класса для обеспечения взаимодействия с базовой моделью и системой правил:

• ActiveObjectMediator – обеспечивает связь с соответствующим активным объектом, использует шаблон проектирования singleton, поэтому доступен из любого класса

• ActiveObjectFacts – выступает в роли БД фактов, что определяет связь с системой правил

Также этот компонент содержит вспомогательные классы:

• Message – представляет общий класс для описания сообщений

• Timer – динамический таймер

• State – внутреннее состояние протокола

• Parameter – суперкласс для Timer и Message, позволяет хранить произвольные параметры (с помощью хэш-таблицы). Кроме того, определяет тип сообщения/таймера (строковая константа)

Кроме того, в компоненте связи определен интерфейс IActiveObject, который должен быть реализован активным объектом.

С технической точки зрения, связь между активным объектом и системой правил реализована следующим образом. Активный объект при инициализации регистрирует себя в ActiveObjectMediator. Когда АО получает сообщение (класс сообщений AnyLogic), он преобразует его в класс Message и передает ActiveObjectMediator. В этом классе на основании переданного сообщения создается терм, и вызывается системы правил для его редуцирования. Аналогично обрабатывается срабатывание таймера.

В обратном направлении взаимодействие происходит через объект ActiveObjectFacts. С помощью стандартных конструкций TermWare из правил могут быть вызваны методы этого объекта для осуществления элементарных действий. Изменение состояния обрабатывается внутри этого объекта, а посылка сообщения и установка таймера перенаправляются ActiveObjectMediator, который через интерфейс IActiveObject вызывает соответствующие методы активного объекта.

Большинство классов компонента связи одинаковы для всех протоколов. В идеальном случае необходимо изменить только класс State, который должен содержать данные и методы, специфичные для данного протокола. Кроме того, если можно обойтись без использования методов (вся обработка данных производится из правил), есть возможность использовать один класс State, унаследованный от Parameter. Если же требуются существенные изменения во взаимодействии правил и модели, возможно изменение ActiveObjectFacts.

6. Описание протоколов – пример

В качестве примера описания протокола с использованием TermWare рассмотрим синхронный протокол доступа к каналу, основанный на выделении временных слотов для всех мотов. Смысл работы протокола заключается в определении точного времени приема и передачи каждого мота, что позволяет существенно сократить затраты энергии на функционирование приемника и передатчика. Работа мотов разбита на фреймы, в начале которых производится передача данных, а дальше радио выключается. В пределах фрейма, есть несколько фаз. В первую фазу, моты определяют необходимость рассылки сообщений, и только в эту фазу необходимо прослушивание эфира всеми мотами. После окончания этой фазы определяется, кто будет передавать и принимать сообщение, и в течении следующих фаз происходит передача сообщения и подтверждение. Структура протокола показана на рис. 5

Рис. 5. Структура протокола доступа к каналу.

Для этого протокола характерны следующие параметры:

- Активный объект: MACLayer
- Состояние:
  - Текущая фаза
  - Номер текущего слота
  - Число слотов
  - Номер слота для мота
  - Длины фаз и кадров
  - Очередь сообщений – очередь сообщений (с сетевого уровня), ожидающих доставки
  - Следующее сообщение – сообщение, которое будет отправлено в начале следующей фазы (особенность синхронного протокола – сообщения отправляются не сразу, а по расписанию)
  - Список желающих послать сообщение – определяет список мотов, желающих послать сообщение в данном фрейме.
  - Текущие приоритеты мотов – определяют, кто будет посылать сообщение, меняются после каждой посылки, чтобы каждый мот мог дождаться своей очереди.
- Типы сообщений (в скобках указаны параметры):
  - Network.SendMessage (Message Payload) – исходное сообщение с сетевого уровня, определяющее необходимость посылки сообщения
  - Receiver.On/Receiver.Off – сообщения о включении/выключении приемника
  - MAC.RequestSlots (int Mot) – служебное сообщение первой фазы
  - MAC.TransmissionRequest – служебное сообщение второй фазы
  - MAC.TransmissionAcknowledge (Boolean ACK) – служебное сообщение(подтверждение) третьей фазы
  - MAC.DataTransmission (Message payload) – передача пакета данных
  - MAC.DataAcknowledge (Boolean ACK) – служебное сообщение(подтверждение) пятой фазы
- Таймеры
  - FrameStart – начало фрейма
  - MySlot – наступление временного слота данного мота
  - PhaseStart (int PhaseNo) – начало фазы

Описание протокола состоит из действий, которые нужно произвести при поступлении соответствующих сообщений или срабатывании таймеров. Описание приведено в табл. 1

Сообщение/таймер	Параметр	Действие
Network.SendMessage	Payload	Добавить сообщение (Payload) в очередь сообщений
MAC.RequestSlots	Mot	Добавить отправителя (Mot) в список желающих послать сообщение
MAC.TransmissionRequest		Выставить следующее сообщение MAC.TransmissionAcknowledge
MAC.TransmissionAcknowledge	ACK=true	Выставить следующее сообщение MAC. DataTransmission
	ACK=false	Послать Receiver.Off
MAC.DataTransmission	Payload	Передать сообщение (Payload) на сетевой уровень, выставить следующее сообщение MAC.TransmissionAcknowledge
MAC.DataAcknowledge	ACK=true	Послать Receiver.Off, убрать доставленное сообщение из очереди

	ACK=false	Послать Receiver.Off
FrameStart		Установить таймер FrameStart, перейти к обработке PhaseStart 1
PhaseStart	1	Послать Receiver.On, модифицировать состояние, установить таймеры MySlot и PhaseStart 2
	2	Проанализировать список мотов, желающих послать сообщение; на основании этого принять решение о роли данного мота в данном фрейме Если мот посылает: послать сообщение MAC.TransmissionRequest и установить таймер PhaseStart 3 Если мот принимает: установить таймер PhaseStart 3 Если мот не участвует в передаче в данном фрейме: послать сообщение Receiver.Off
	3-5	Послать текущее сообщение и установить следующий таймер PhaseStart
MySlot		Если очередь непустая, послать сообщение MAC.RequestSlots

Табл.1 Описание протокола

Эти действия записываются с помощью стандартных средств TermWare. Например, сообщение Network.SendMessage обрабатывается следующим образом:

MessageNetworkSendMessage->PutInQueue($msg) [assign($msg,getLastMesage().getMessage("Payload"))],
PutInQueue($msg)->success[getState().putMessageInQueue($msg)]

Здесь MessageNetworkSendMessage – исходный терм, который редуцируется с использованием системы правил. Этот терм создается объектом ActiveObjectMediator на основании типа сообщения. Редуцирование происходит в два этапа. Сначала исходный терм переходит в PutInQueue($msg), где в переменной $msg сохраняется переданное сообщение (от активного объекта NETLayer), которое получается вызовом соответствующих методов ActiveObjectFacts. После этого обрабатывается терм PutInQueue($msg), при этом вызовом соответствующего метода осуществляется требуемое действие. В результате работы системы правил исходный терм переходит в конечный терм success, который свидетельствует о том, что обработка сообщения (или таймера) завершена успешно.

Сложные конструкции, например, выражающие перебор вариантов, также достаточно просто записываются на TermWare. Например, рассмотрим обработку таймера PhaseStart 2, в которой определяется роль мота в данном фрейме:

TimerPhaseStart -> Phase($n)[assign($n,getLastTimer().getInt("PhaseNo"))]
Phase(2)->CheckSender($f)[assign($f,getState().isSender())],
CheckSender(false)->CheckReceiver($f)[assign($f,getState().isReceiver())],
CheckReceiver(false)->FrameEnded,
CheckReceiver(true)->NextPhase(2),
CheckSender(true)->NextPhase(2)[sendMessage("MACTransmissionRequest","PHY")]

В начале обработки из общего терма TimerPhaseStart строятся термы Phase($n), соответствующие требуемой фазе протокола. В частности, при обработке начала второй фазы вызываются методы isSender() и isReceiver(), определяющие роль данного мота на основании информации, занесенной ранее. При этом происходит переход либо к терму FrameEnded (если мот больше не участвует во взаимодействии), либо к терму NextPhase($n).Эти термы преобразуются в соответствии с другими правилами:

FrameEnded->success[[sendMessage("ReceiverOff","Receiver"), getState().setCurrentState("None")]],
NextPhase($n)->success[setTimer("PhaseStart",getState().getPhaseLength($n-1))]

Рассмотрим также пример использования терма-условия:

Phase($n)[$n>2]->NextPhase($n)[sendMessage(getState().getNextMessage())]

Здесь преобразование происходит одинаково для всех фаз, кроме первой и второй. Это задается условием применения правила $n>2. При этом условие записывается в естественном виде, с использованием сокращения x>y вместо терма greater(x,y).

Более сложный пример условия используется при обработке подтверждения приема:

MessageMACTransmissionAcknowledge [getLastMessage().getBoolean("ACK")]
->FrameEnded [getState().removeMessageFromQueue()]
!->FrameEnded,

Здесь в зависимости от значения булевского поля ACK в сообщении выполняется или не выполняется метод removeMessageFromQueue(), который убирает сообщение из очереди неотправленных сообщений.

Заключение

В данной работе продемонстрирована возможность совместного использования визуальной среды моделирования AnyLogic и системы символьных вычислений TermWare для построения системы моделирования сенсорных сетей. Такое совместное использование обеспечивает многие преимущества по сравнению с использованием только средств AnyLogic. Реализованный прототип системы моделирования показывает обоснованность выбранного подхода. В дальнейшем планируется расширять и совершенствовать эту систему, как базовую модель, так и систему описания протоколов. Важным дополнением к системе может

быть графический редактор протоколов, который позволит создавать протоколы без знаний синтаксиса языка TermWare.

Описанный подход к описанию протоколов позволяет моделировать различные протоколы взаимодействия в сенсорной сети, сравнивать их эффективность и надежность. Тем самым появляется возможность подбора наиболее эффективного протокола для каждой задачи. Это позволяет в полной мере использовать ограниченные ресурсы сенсорных сетей в разнообразных приложениях, как в науке, так и в прикладных задачах.

1. D. Culler, D. Estrin, M. Srivastava. Guest Editors' Introduction: Overview of Sensor Networks, Computer, 37(8):41-49, 2004
2. K. Martinez, J. K. Hart, R. Ong. Environmental Sensor Networks, Computer, 37(8):50-56, 2004
3. S. M. Brennan, A. M. Mielke, D. C. Torney, A. B. Maccabe. Radiation Detection with Distributed Sensor Networks, Computer, 37(8):57-59, 2004
4. M. Maroti, G. Simon, A. Ledeczi, J. Sztipanovits. Shooter Localization in Urban Terrain, Computer, 37(8):60-61, 2004
5. K. Akkaya, M. Younis. A Survey on Routing Protocols for Wireless Sensor Networks, Ad Hoc Networks, 3:325-349, 2005
6. C.C. Enz, A. El-Hoiydi, J.-D. Decotignie, V. Peiris. WiseNET: An Ultralow-Power Wireless Sensor Network Solution, Computer, 37(8):62-70, 2004
7. TinyOS project, http://www.tinyos.net/
8. P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer and D. Culler. The Emergence of Networking Abstractions and Techniques in TinyOS, Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)
9. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems, Proceedings of Programming Language Design and Implementation (PLDI) 2003
10. Tiny Application Sensor Kit (TASK), www.tinyos.net/tinyos-1.x/doc/task.pdf
11. A.Kröller, D.Pfisterer, C.Buschmann, S.P. Fekete, S.Fischer. Shawn: A new approach to simulating wireless sensor networks, Design, Analysis, and Simulation of Distributed Systems 2005, Part of the SpringSim 2005
12. P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications, Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)
13. Manish Karir, et al. ATEMU: A Fine-grained Sensor Network Simulator, Proceedings of First IEEE International Conference on Sensor and Ad Hoc Communication Networks (SECON'04), Santa Clara, CA, October 2004
14. Jason Liu, L. Felipe Perrone, David M. Nicol, Michael Liljenstam, Chip Elliott, and David Pearson, Simulation Modeling of Large-Scale Ad-hoc Sensor Networks, In Proceedings of European Simulation Interoperability Workshop, 2001.
15. Luiz Felipe Perrone and David Nicol, A Scalable Simulator for TinyOS Applications, In Proceedings of the Winter Simulation Conference, 2002
16. Sameer Sundresh, Wooyoung Kim, and Gul Agha, SENS: A Sensor, Environment and Network Simulator , In Proceedings of 37th Annual Simulation Symposium, pages 221-230, 2004.
17. Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin, EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks In Proceedings of the USENIX Technical Conference, 2004.
18. The Network Simulator - ns-2 http://www.isi.edu/nsnam/ns/
19. Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Honghai Zhang, ``J-Sim: a simulation and emulation environment for wireless sensor networks," Proc. 38th Annual Simulation Symposium, April 2005, San Diego, CA.
20. Среда моделирования AnyLogic, http://www.xjtek.ru/anylogic/
21. Карпов Ю.Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. Издательство «BHV», 2005
22. Doroshenko A.E., Shevchenko, R. TermWare: A Rewriting Framework for Rule-Based Programming Dynamic Applications Proc. Int. Workshop "Concurrency: Specification and Programming, Sept., 2005, Ruciane Nida, Poland, Warsaw University, pp. 100-111.
23. Gerard J. Holzmann. The Model Checker Spin, IEEE Trans. on Software Engineering, 23(5): 279-295, 1997