Лабораторная работа № 5

Создание значка анализа времени

На диаграмме Statechart, созданной нами ранее, следующим значком был анализ времени, который здесь можно расписать подробнее.

Добавление значка получения времени

Из состояния ожидания контроллер должен быть выведен таймером, после чего происходит проверка текущего времени выращивания на предмет его окончания. Допустим, что таймер может выдать время, которое прошло от момента начала посадок, и контроллер должен получить это время для анализа, не пора ли заканчивать работу.

Для отображения этого процесса создадим новый значок Activity и назовем его GetCurrentTime, который соединен со значком Idle событием Timer. Добавим значок GetCurrentTime для отражения обращения к таймеру за текущим временем. Создадим значок ReturnTime для объекта Timer и введем значок решения для контроллера (рис. 33).

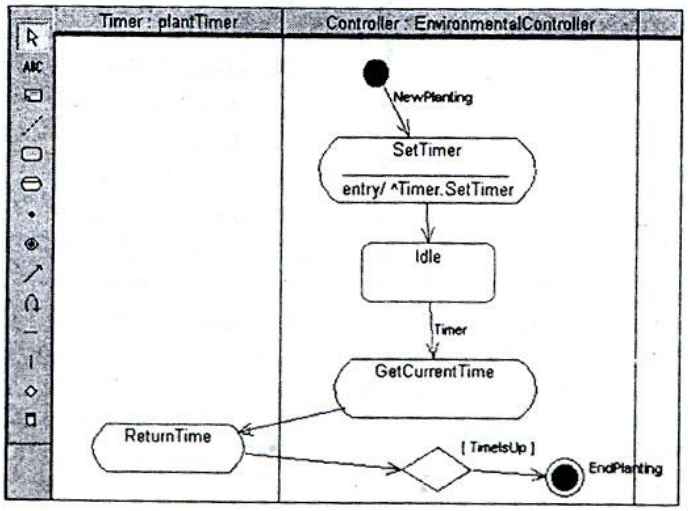

Рис.33. Диаграмма активности после ввода значка решения

Для того чтобы отразить, что работа контроллера должна быть закончена, добавим значок EndState и соединим его стрелкой с условием что переход должен осуществляться только а случае, когда закончилось время выращивания растений На рис. 34 показано, как необходимо заполнить свойства для StateTransition в этом случае.

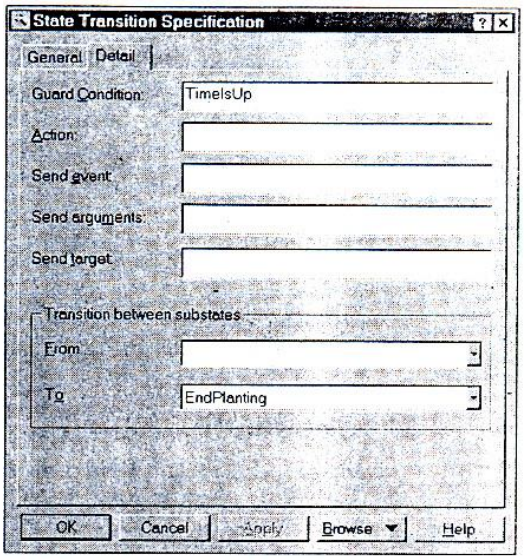

Рис.34. Установка свойств для StateTransition

Добавление решения

Значок решение стал достаточно неплохим добавлением. Теперь есть возможность показать ветвление по определенным условиям и хотя данный тип диаграмм никак не отражается на получаемом программном коде, но он позволяет точно описать последовательность действий при выполнении определенных задач, для которых показ ветвления может оказаться просто необходимым.

Если вы программ то попробуйте вспомнить, когда вы последний раз рисовали блок-схемы алгоритмов. Сейчас это происходит все реже и реже именно по причине ограничения времен на выполнение проектов и доступности вычислительных машин.

В то время, когда доступ к ЭВМ осуществлялся по предварительной записи, никто не мог позволить себе такую роскошь, как изобретать алгоритм сидя за клавиатурой терминал. Сначала блок-схема, затем — код, написанный вручную на листах перфорированной бумаги, и только затем клавиатура терминала.

Сейчас многое изменилось, программы стали создаваться быстрее, но их поддержка становится все более трудоемким делом. В связи с появлением объектов, обменивающихся сообщениями, нарисовав блок-схему таких программ стало значительно труднее.

Но теперь вместо блок-схем мы имеем программу Rational Rose, которая позволяет значительно проще и нагляднее описать алгоритм любой сложности, и немалую часть этой простоты и наглядности привносят такие значки как Decision (решение) и Swimlane (разделитель).

В нашем случае значок решения будет обозначать проверку на окончание плана выращивания растения. Для большего упрощения задачи будем использовать специализированный контроллер, который запрограммирован на выращивание определенного типа растений. Если вспомнить, что за хранение плана выращивания отвечает специализированный класс GrowingPlan, то необходимо обращение контроллера к этому классу для проверки, завершилось ли время выращивания нашего растения.

Синхронизация процессов

Для того чтобы обозначить действия, совершаемые объектом GrowingPlan (план выращивания), добавим в диаграмму новую линию Swimlane и назовем ее GrowingPlan, после чего добавим в поле GrowingPlan новое действие ReturnAllTime. Теперь для отражения того, что для дальнейшей работы необходимо получить оба времени, и текущее, и полное, введем линию горизонтальной синхронизации.

Конечно, в данном случае можно было бы обойтись и без синхронизации, если создать последовательность действий, отражающую последовательное получение указанных данных. В данном случае мы использовали синхронизацию для показа примера ее применения.

После добавления синхронизации диаграмма получит следующий вид (рис. 35).

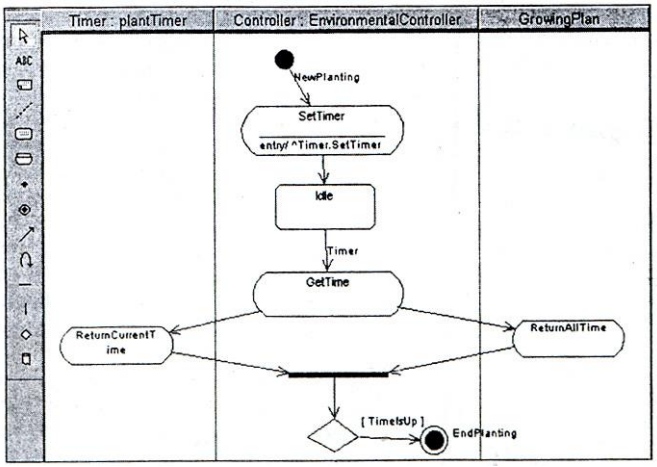

Рис.35 Диаграмма после добавления значка синхронизации

Линия синхронизации обычно включается в том случае, когда имеются независимые процессоры, которые выполняют задачи параллельно. Если бы у GrowingPlan был свой процессор, который позволял бы выполнять задачи параллельно, то синхронизация была бы жизненно необходима. Но в нашем случае она только показывает, что дальнейший переход не может быть осуществлен без обработки данных как таймером, так и планом выращивания, и только после того, как оба этих объекта возвратят запрашиваемую информацию, контроллер может произвести ее анализ и принять решение о дальнейших действиях.

Обычно линии синхронизации используются при проектировании процессов клиент-сервер, когда клиенты выдают запросы сразу нескольким серверам и ожидают от них ответа, или при проектировании бизнес-процессов, происходящих в нескольких структурных подразделениях одного офиса, где, например, различные документы обрабатываются в подразделениях, а затем, по завершении создания всех документов, происходит их группировка в один отчет.

Добавление опроса датчиков

Далее для получения картины работы контроллера введем значок активности TestingEnvironment, который, как и в Statechart диаграммы показывает опрос датчиков теплицы.

Алгоритмы опроса датчиков могут быть различны. Выбор алгоритма зависит от проектировщика системы, его опыта и предпочтений. Здесь задача состоит в том, что необходимо в первую очередь создать работоспособную систему и на ее примере научиться пользоваться инструментом. Разберем различные варианты создания датчиков:

1. Создать активные датчики, которые будут сами посылать сообщения контроллеру при изменении параметров, а контроллер будет активизировать исполнительные устройства.
2. Опрашивать датчики по одному с немедленной реакцией на отклонения.
3. Опросить сначала все датчики, а затем, сравнив с необходимыми значениями параметров, заложенными в плане выращивания, активизировать исполнительные устройства для приведения этих параметров в должное состояние.

Первый вариант подразумевает создание нескольких контроллеров определенных параметров среды, которые не связаны друг с другом. С точки зрения объектно-ориентированного проектирования этот вариант предпочтительней. Он позволит на основе класса контроллера создавать контроллеры различных параметров и, главное, позволит в дальнейшем расширять систему, добавляя новые контроллеры дополнительных параметров среды практически без затрагивания уже работающих контроллеров.

Второй вариант тоже может быть реализован, но он недалеко уходит от последнего, который, опять же, для простоты создания нашей учебной системы мы и примем к реализации.

По принятому варианту необходим опрос датчиков температуры и pH, затем сравнение с необходимыми значениями и настройка среды путем включения или выключения исполнительных устройств и занесение в протокол данных при изменении параметров.

Окончательный вариант диаграммы

Для отражения этих действий добавим на диаграмму значки действий TestingEnvironment, Writing, AdjustingEnvironment. Мы не будем расшифровывать содержание этих действий, чтобы не загромождать диаграмму, но добавим значки решений, которые направляют действия при изменении (IsChanged) среды и при необходимости настройки (NeedAdjust) среды. При этом должна получиться диаграмма, представленная на рис. 36.

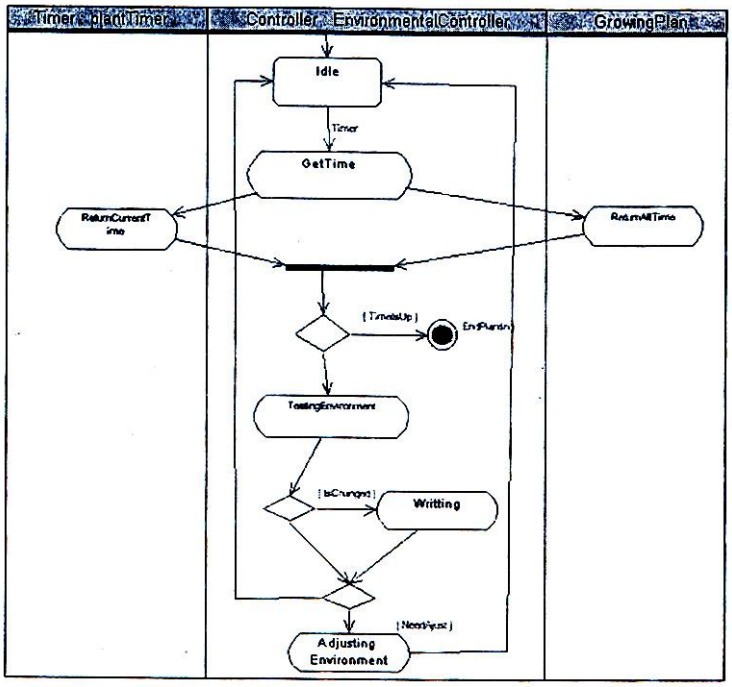

Рис.36. Диаграмма после добавления всех действий.

В окончательном варианте на диаграмме получен полный алгоритм работы контроллера от тестирования параметров среды до их настройки с протоколированием.

В отличие от Statechart Diagram здесь мы не стали подробно расписывать действия TestingEnvironment, Writing, AdjustingEnvironment, которые далее распишем при помощи вложенных диаграмм. Вложенные диаграммы в версии 2000 можно создать для каждого значка состояния или активности.

Эта возможность позволяет не загромождать саму диаграмму, а создавать сколько угодно вложенных, что не только скрывает ненужные детали, но и позволяет легко разделить проект для работы команды программистов и также легко собрать проект обратно.

Если вы посмотрите на полученную диаграмму, то заметите, что по алгоритму запись в протокол происходит до проверки на необходимость настройки среды. Это сделано потому, что возможно изменение среды без ее настройки. Например, согласно плану выращивания необходимо скорректировать показатели, а показатели уже изменились в нужную сторону благодаря изменению внешних условий. В этом случае необходимо записать данные об изменениях, но настраивать, т.е. включать исполнительные устройства, нет необходимости.

Создание алгоритма — работа творческая и у каждого проектировщика алгоритм может несколько отличаться в деталях. Но самое главное, Rational Rose позволяет создать такие диаграммы, которые позволяют сделать алгоритм работы программы четким, легко читаемым, понятным даже тому, кто видит его впервые.

Создание вложенной диаграммы

Для создания вложенной диаграммы выберем значок TestingEnvironment и проделаем RClick=>SubDiagram=>New Activity Diagram.

При этом откроется новая диаграмма, которая в окне Browser будет выглядеть так, как представлено на рис.37. Естественно, здесь название новой диаграммы с присвоенного при создании программой изменено на TestingEnvironmentDiagram.

Таким образом, разработчик сам должен следить за тем, чтобы эти диаграммы не входили в противоречие между собой по входным и выходным переходам состояний и действий. Такой подход позволяет не уделять много внимания синтаксически правильной стыковке диаграмм различных уровней, а сосредоточиться на логике процесса. Но все же, необходима дополнительная проверку, которая позволит избежать трудно

обнаруживаемых впоследствии логических ошибок, но с возможностью включения и выключения такой проверки, когда это необходимо.

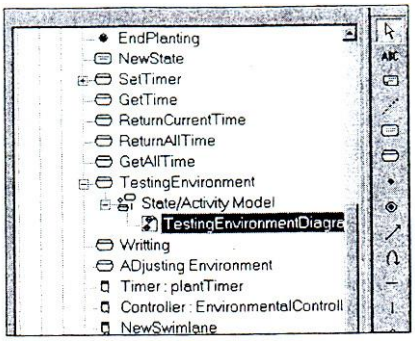

Рис.37. SubDiagram для TestingEnvironment в окне Browse

Добавим во вновь созданную диаграмму значки начала и конца действия, значки действия получения значения температуры и получения значения pH и, конечно же, используем для помещения этих значков соответствующие Swimlane. Таким образом, должна получиться диаграмма, показанная на рис. 38. Если теперь активизировать контекстное меню на значке TestingEnvironment, то в нем появится пункт для быстрого перехода на вновь созданную диаграмму TestingEnvironmentDiagram (рис.39).

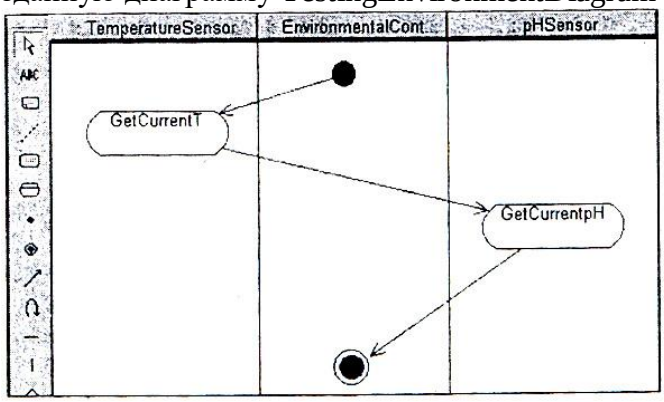

Рис.38. Диаграмма TestingEnvironmentDiagram

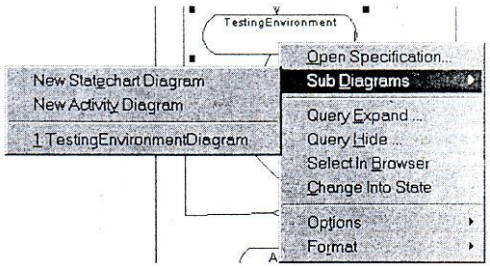

Рис.39. Измененное контекстное меню значка TestingEnvironment

Мы получили вложенную диаграмму, которая связана с основной. Конечно, в нашем случае вложенная диаграмма проста, однако, в других системах такие диаграммы могут быть сложны настолько, что потребуется создание еще более глубокой вложенности. А ее создание в Rational Rose не составит никакого труда. Таким образом, можно детализировать алгоритм программы, начиная с выполнения крупных задач и постепенно разбивая их на более мелкие, вплоть до выполнения отдельных операторов выбранного языка.

Теперь в качестве тренировки вы можете самостоятельно создать вложенные диаграммы для значков Writing и AdjustingEnvironment. Диаграммы создаются аналогично рассмотренной.