Лабораторная работа № 6
Описание взаимодействия при помощи Sequence Diagram

Назначение диаграммы

Продолжением создания гидропонной системы будет диаграмма, взаимодействия между объектами.

Кроме сценария поведения каждого объекта системы необходимо точно представлять взаимодействие этих объектов между собой, определение клиентов и серверов и порядка обмена сообщений между ними.

Обмен сообщениями происходит в определенной последовательности, и Sequence Diagram позволяют получить отражение этого обмена во времени.

В течение работы сложной системы объекты, являющиеся клиентами, посылают друг другу различные сообщения, а объекты, являющиеся серверами, обрабатывают их. В простейшем случае можно рассматривать сообщение как вызов метода какого-либо класса, в более сложных случаях сервер имеет обработчик очереди сообщений, и сообщения обрабатываются им асинхронно, т.е. сервер накапливает несколько сообщений в очереди, если не может обработать их сразу.

На основе приема-передачи сообщений основана многозадачность Windows, а в нашем случае для простоты демонстрации создания приложения будем считать, что сообщения обрабатываются немедленно в той последовательности, в которой они выдаются клиентами.

Создание диаграммы

Создадим Interaction Diagram при помощи Menu:Browse=>Interaction Diagram=>New или значка в строке инструментов.

При этом активизируется окно, представленное на рис. 40.

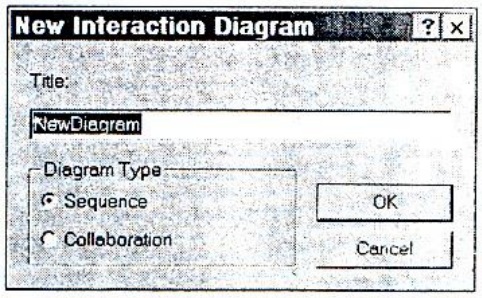

Рис.40 Создание Interaction Diagram

Присвоим полю Title значение Process. Здесь можно выбрать из двух типов диаграмм. Эти диаграммы показывают взаимодействие объектов с разных сторон. Collaboration показывает взаимодействие объектов независимо от времени. Sequence в свою очередь, представляет собой разворот взаимодействия во времени и отражает последовательность выдачи сообщений клиентами. Создадим Sequence диаграмму.

Строка инструментов диаграммы

При переходе в эту диаграмму панель инструментов сменилась на новую, в которой теперь доступны значки Text Box, Note, Anchor To Item, Object, Message, Message to Self. Первые три были рассмотрены для предыдущих диаграмм и здесь выполняют те же функции, поэтому не будем на них останавливаться.

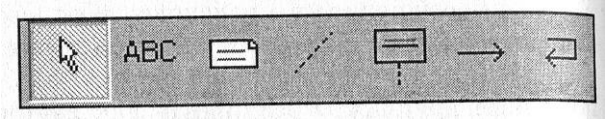

Рис.41.Строка инструментов Sequence Diagram

Object (объект)
Значок Object позволяет включить новый объект в диаграмму. Каждый объект является реализацией класса, поэтому в нем можно указать класс, на основе которого он создан.
Message(сообщение)

Значок Message позволяет создать сообщение, передаваемое от одного объекта к другому. Так как все взаимодействие в объектно-ориентированных системах осуществляется при помощи сообщений между объектами, то классы должны позволять отправку или прием сообщений.
Message to Self (сообщение самому себе)

Данный значок позволяет показать, что отправитель сообщения является одновременно и его получателем.

При обмене сообщениями одни классы являются клиентами в принимают сообщения, а другие — серверами и отправляют сообщения. В том случае, когда объект отправляет сообщение самому себе, он одновременно является и сервером и клиентом, что и отражает данный значок.

Настройка времени жизни объекта

Добавим в диаграмму два новых объекта. Первый назовем Timer, второй — Controller. Для задания названий необходимо сначала выделить значок, а затем просто указать мышкой в середину значка и ввести название.

По двойному нажатию мышки на значке иди через RClick=> Open Specification активизируется диалоговое окно параметров объекта. Для объекта Controller оно показано на рис. 42.

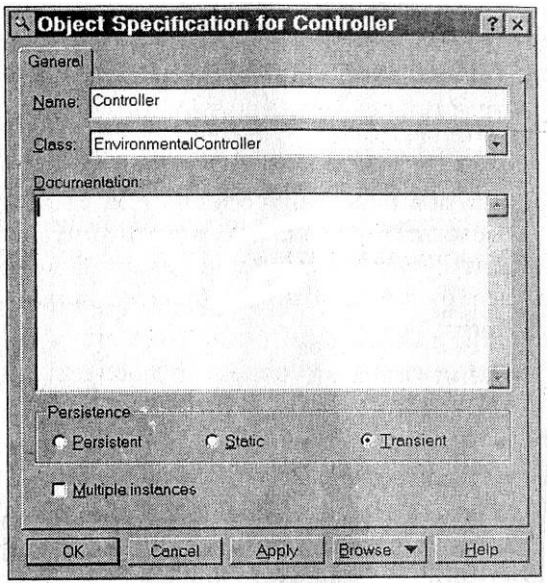

Рис.42. Настройка параметров объекта

Здесь также можно ввести название объекта и класс, реализацией которого является данный объект. Для контроллера мы уже создали класс EnvironmentalController, поэтому его и выберем. Для объекта Timer класс еще не создан, что является нашим упущением, поэтому пока оставим поле Class пустым. В дальнейшем наилучшим решением для нашей диаграммы будет добавление класса Timer в диаграмму классов.

Для реализации объектов в окне параметров доступна опция Persistence, отражающая время жизни объекта. Время жизни объекта — это время от его создания до уничтожения. Обычно объекты существуют в пределах их области видимости и автоматически уничтожаются системой, когда выходят за эту область. Это происходит в тех случаях, когда, например, в подпрограмме определяется объект класса, который автоматически уничтожается при завершении подпрограммы.

В языке C++ можно создать и удалить объект, не дожидаясь, когда это сделает система, при помощи команд new и delete. Обычно к такому варианту прибегают еще и для уменьшения исполняемого файла. Известно, что при создании программы на Visual C++ такое определение:

```
char s[10000];
```

приведет к увеличению исполняемого файла на 10000 байт, так как для этой переменной будет зарезервировано место в области данных программы.

Однако создание массива при помощи операторов:

```
char *s;
s=new char[10000];
 // работа с массивом
```

практически не приведет к увеличению размера исполняемого файла, хотя подразумевает, что программист освободит выделенную память до завершения программы при помощи оператора:

```
delete s[];
```

иначе она будет недоступна для системы в целом.

Обычным решением, для того чтобы не забыть освободить выделенную память, будет выделение памяти в конструкторе класса и освобождение ее в деструкторе. Таким образом, можно настроить следующие параметры жизни объекта:

- Persistent означает, что область видимости объекта превышает время жизни.
- Static данный элемент существует на всем протяжении работы программы.
- Transient означает, что время жизни объекта и область видимости совпадают. Этот вариант присваивается по умолчанию и нам он подходит для обоих случаев.

Для того чтобы показать, что эти свойства описывают состояние всех объектов данного класса, можно указать Multiple instances, но для нашего случая это не обязательно.

Создание класса из окна настройки параметров

Теперь вспомним, что для нашего объекта Timer не указан класс, на основе которого этот объект создается. Зайдем в окно свойств объекта, перейдем в поле Class и нажмем клавишу «стрелка вверх». При этом сразу открывается окно свойств класса (рис. 43). Здесь нужно ввести только имя plantTimer (это будет класс, отвечающий за расчет текущего времени роста растений) и нажать ОК.

Теперь, по своему усмотрению, можно скрыть или показать наименование класса на значке объекта, как показано на рис. 44.

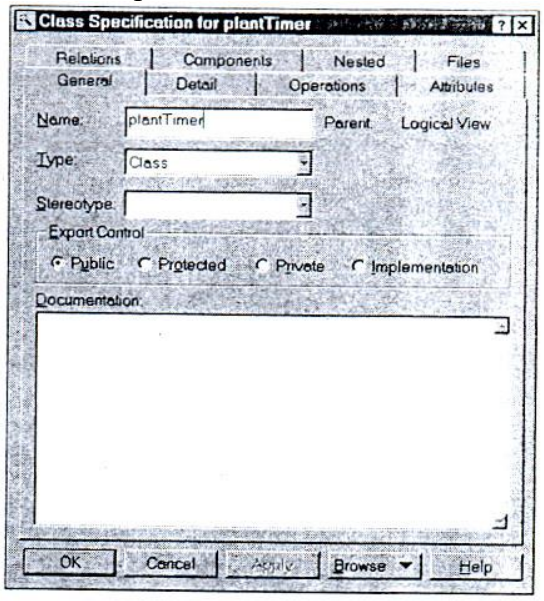

Рис. 43. Свойства класса

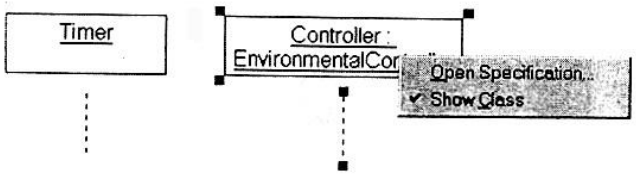

Рис.44. Показ и скрытие названия класса

Создание сообщений

Перейдем непосредственно к сообщениям. Выбираем стрелку Object Message и соединяем пунктирные линии контроллера и таймера. Затем еще раз, но от таймера к контроллеру ниже первого сообщения. В данном случае мы подразумеваем, что самым первым сообщением будет указание таймеру времени, через которое происходит активизация и передача адреса обратного вызова таймером. Вторым сообщением будет сообщение от таймера, что указанный промежуток времени истек.

В PC совместимых компьютерах системные часы дают возможность устанавливать минимальное время таймера 1/60 секунды. Но для нашей системы такая точность совершенно не нужна. Растения растут медленно, а температура или pH падает или увеличивается на ощутимую величину, по крайней мере, за несколько минут. Поэтому для нашего таймера приемлемой величиной будет активизация раз в минуту. Однако, позволяя устанавливать промежуток времени произвольно, мы, во-первых, придаем нашей системе большую гибкость и, во-вторых, что не маловажно, оставляем себе лазейку для отладки системы. Можем установить заведомо большее или меньшее время для ускоренного тестирования процесса.

Выделим первое сообщение и вызовем контекстное меню (RClick). Здесь мы увидим уже знакомый пункт Open Specification и пункт New Operation.

При помощи последнего можно добавить новую операцию (метод) к классу, из которого создан объект. Выберем этот пункт и введем название операции SetTimer (рис.45).

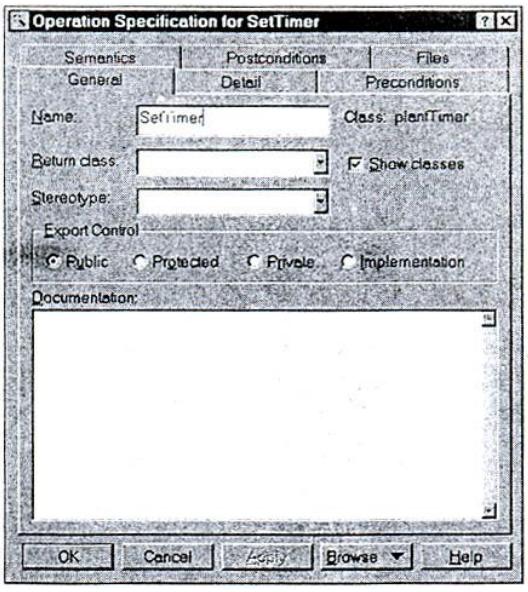

Рис. 45.Редактирование названия операции

И нажмем ОК. Вопреки ожиданиям, мы попали обратно в нашу диаграмму, и на ней ничего не изменилось. Но это только на первый взгляд. Если еще раз вызвать контекстное меню, то в нем, кроме уже описанных пунктов будет присутствовать пункт SetTimer(), после выбора которого над стрелкой сообщения появляется название SetTimer.

Что же сейчас было проделано? Не выходя в диаграмму классов, мы добавили новый класс Timer и метод к нему SetTimer. В этом и проявляется мощь Rational Rose, что

диаграммы связаны. Классы и методы к ним добавляются в том месте, где это наиболее логично и понятно. Причем, если внести изменения в описание класса на любой диаграмме то эти изменения тут же появятся на остальных диаграммах модели.

Свойства сообщений

Теперь разберемся со свойствами только что введенного общения. При вызове SetTimer()=>RClick=>Detial мы попадаем в диалоговое окно, показанное на рис.46.

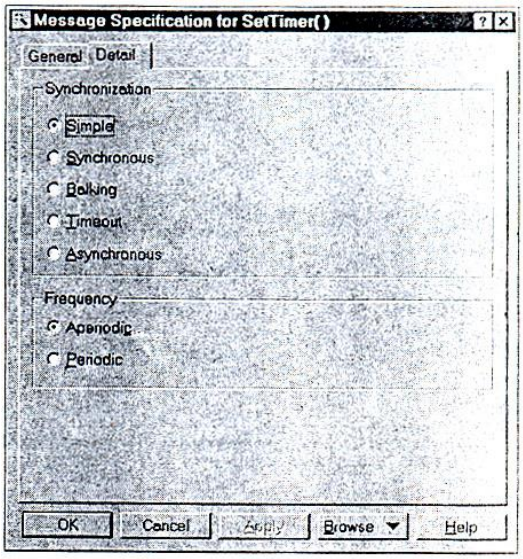

Рис. 46. Установка свойств сообщения

Здесь мы видим две группы радио-кнопок (кнопок с зависимой фиксацией):

1. Synchroniztion — определяет порядок обмена сообщениями и может быть выбрана из следующих вариантов:

- Simple — простая посылка сообщения;
- Synchronous — операция происходит только в том случае, когда клиент посылает сообщение, в сервер может принять сообщение клиента;
- Balking — операция происходит только в том случае, когда сервер готов немедленно принять сообщение, если сервер не готов к приему, клиент не выдает сообщение;
- Timeout — клиент отказывается от выдачи сообщения, если сервер в течение определенного времени не может его принять;
- Asynchronous — клиент выдает сообщение, и, не ожидая ответа сервера, продолжает выполнение своего программного кода.

Для каждого вида операций стрелка сообщения изменяется в соответствии с рис.47.

2.Frequency — определяет частоту обмена сообщениями:

- Periodic — сообщения поступают от клиента с заданной периодичностью;
- Aperiodic — сообщения поступают от клиента нерегулярно.

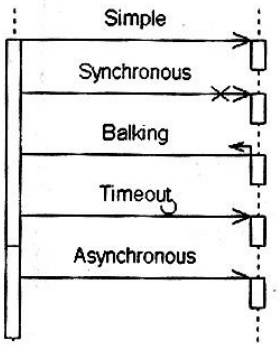

Рис.47. Различные виды сообщений

Для операции OnTimer() подходит установка Asynchronous и Periodic, потому что классу таймера все равно, что будет делать с его сообщением другой класс, причем в нашем случае Timer обслуживает только класс контроллера, а в реальной системе, возможно, понадобится вызывать прерывание для нескольких объектов.

Добавление класса plantDayHour

Если посмотреть на созданную ранее диаграмму состояний, то после получения сообщения от таймера происходит анализ времени, прошедшего от начала посадки растений. Этот анализ неплохо было бы переложить на плечи объекта специального класса, например класса plantDayHour. Этот класс будет отвечать за отсчет текущего дня и часа. Добавим объект m_plantDayHour и создадим для него новый класс plantDayHour. Соединим новыми сообщениями UpdateTime() и GetCurrentTime() объект контроллера и m_plantDayHour и получим рис. 48.

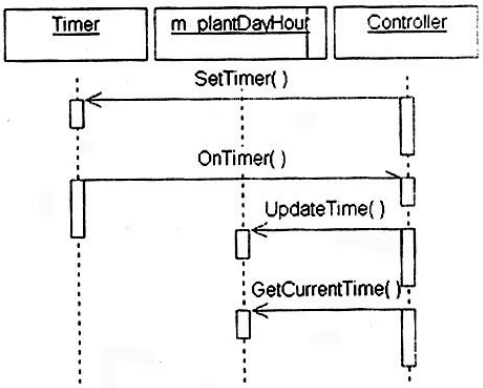

Рис.48. Добавленный объект m_plantDayHour

Окончательный вариант диаграммы

В этом и заключается процесс визуального проектирования, что система получает все большую детализацию в процессе добавления диаграмм и детализации взаимодействия между ними.

Теперь, когда все необходимые возможности изучены, можно добавить сообщения между датчиками и исполнительными устройствами. Начало этого процесса показано на рис.49.

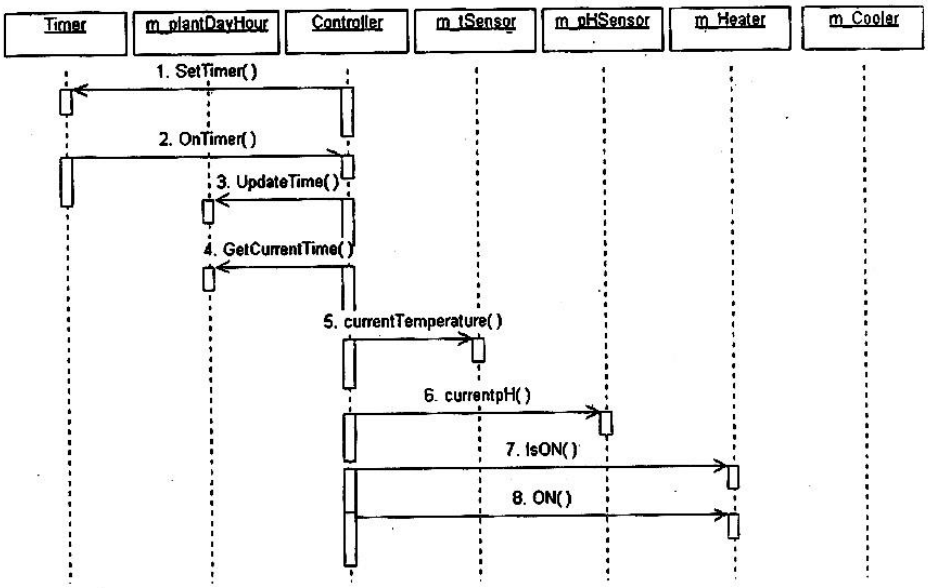

Рис. 49. Добавление в диаграмму датчиков и исполнительных устройств

На этом рисунке добавлены датчики, нагреватель и вентилятор. Добавить WaterTank, NutrientTank и Light вы можете самостоятельно.