UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONALES Y DE POSGRADO

SISTEMA DE ADQUISICIÓN DE DATOS APLICADO A LA ACUACULTURA (UN ENFOQUE ORIENTADO A OBJETOS)

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

BERNABE ORTIZ Y HERBERT

DIRECTOR DE TESIS:
DRA. HANNA OKTABA

OCTUBRE DE 2000

AGRADECIMIENTOS

Quiero agradecer a las profesoras Hanna Oktaba y a Lupita Ibarguengoitia sus valiosos consejos y el gran interés en este trabajo.

Deseo agradecer también a mi equipo de trabajo durante los estudios de maestría Alberto León Manilla y Daniel de la Cruz Estrada.

Además gracias especiales a Ricardo Calderón de la Barca y a Jerry Dueck por sus valiosos consejos.

CONTENIDO

Introducción.

El objetivo del presente trabajo es aplicar la metodología orientada a objetos a un sistema de adquisición de datos, empleando el método de Booch.

Como caso de estudio, se eligió un sistema acuicola, en donde se llevan a cabo los estudios básicos que permiten el desarrollo de técnicas de acuacultura.

En las actividades del diseño orientado a objetos se emplea el método de Booch, como un medio de estructurar y sistematizar los componentes de software del sistema de adquisición de datos aplicado a la acuacultura.

Un sistema acuícola automatizado en su fase mas general se puede plantear como un proceso de llevar a cabo eficientemente con mínima intervención humana, el cultivo de peces y moluscos para la producción correcta de múltiples cosechas.

De esta manera, la automatización de la planta acuícola se plantea como un problema de adquisición de datos: a partir del comportamiento de un sistema real (como un cultivo de moluscos) se obtienen datos físicos a través de sensores que los convierten en datos eléctricos. Los datos eléctricos son trasmitidos a un circuito llamado tarjeta de adquisición de datos la cual convierte estos datos a un formato aceptable por la computadora. Finalmente, la computadora utilizando el software apropiado recibe los datos, los analiza, los almacena, los muestra en pantalla, produce reportes y ofrece control y comunicación. En una palabra, el sistema de adquisición de datos nos facilita el estudio y nos permite controlar el comportamiento del sistema a partir de datos recabados del mismo, en el momento en que se presentan.

Para llevar a cabo eficientemente los procesos acuícola, se propone un sistema de control automatizado a través del cual se faciliten las diversas tareas sobre cultivos marinos.

Este trabajo se desarrolla en cuatro capítulos y un apéndice.

- En el primer capítulo se fundamenta este trabajo. Se habla de los aspectos esenciales de la acuacultura, el diseño orientado a objetos, el método de Booch, la adquisición de datos y el lenguaje de programación Visual C++.

- En el segundo capítulo se identifica el propósito del sistema y se define el dominio del problema. Se examina con el cliente, en este caso el experto en el dominio del problema metas, objetivos y elementos del sistema y se decide que parte del mismo se desea automatizar.

- En el tercer capítulo, se plantea el análisis y diseño del sistema. Tradicionalmente estas etapas se desarrollaban por separado. Sin embargo, ponemos en práctica la recomendación del método de Booch, primero avanzar con un poco de análisis y luego continuar en forma incremental e iterativamente con un poco de análisis y un poco de diseño. En esta etapa se deben resolver dos problemas centrales ¿Cuál es el comportamiento deseado del sistema? ¿Y cuáles son los papeles y responsabilidades de los objetos que producen dicho comportamiento? La notación de Booch usa los diagramas de clase para representar los roles y responsabilidades entre las clases de objetos, es decir, cómo son construidos los elementos del diseño y las relaciones y atributos de estos elementos, esto se conoce como la estructura estática del sistema. Como parte del análisis y a fin de expresar la semántica dinámica del sistema se usan los diagramas de interacción y los diagramas de objetos con los cuales se modelan e ilustran las diferentes trazas de la ejecución de escenarios del sistema.

- El cuarto capítulo, trata de la instrumentación y codificación de las principales clases y la forma en que se relacionan y cooperan para cumplir con los requisitos inicialmente planteados y, como pueden evolucionar para adaptarse a situaciones cambiantes del sistema. Este capítulo incluye un prototipo, desarrollado en Visual C++, con algunas pruebas del sistema de adquisición de datos en funcionamiento.

Al final se incluye un apéndice del método de Booch y una selección de citas de autores muy importantes de la metodología Orientada a Objetos.

Capítulo 1.

Fundamentación

> ... mi propósito no es el de enseñar aquí el método que cada cual debe seguir para guiar acertadamente su razón, sino solamente el de mostrar de que manera he tratado de guiar la mía.
>
> Descartes (Discurso del método).

1.1 Marco teórico.

Existe una frase muy pegajosa que dice que en el planteamiento del problema está su solución. No hay duda de que para inventar la solución de algún problema se requiere una disciplina de diseño que tome en cuenta la complejidad del dominio del problema.

En este capítulo se hace una breve presentación de los fundamentos de este trabajo: Los sistemas acuícolas, los sistemas de adquisición de datos, el diseño orientado a objetos, el método de diseño de Booch y el ambiente de programación Visual C++.

1.2 Acuacultura.

El problema que se plantea es el diseño de un sistema automatizado para el estudio de moluscos y peces en un medio controlado. De manera general, la acuicultura es un conjunto de actividades destinadas al aprovechamiento y mejora de los recursos naturales mediante la cría de animales y el cultivo de plantas en un medio acuático.

Como se verá mas adelante en la entrevista (Capitulo 2), el cultivo dependerá por completo del alimento y del control que se ejerza sobre el medio acuático (oxigenación del agua, disolución de nutrientes, control del pH, etcétera). El problema que se plantea es lograr el máximo rendimiento. Para ello los investigadores, experimentan con diversos planes de cultivo utilizando abonos naturales y acondicionando las zonas de cultivo.

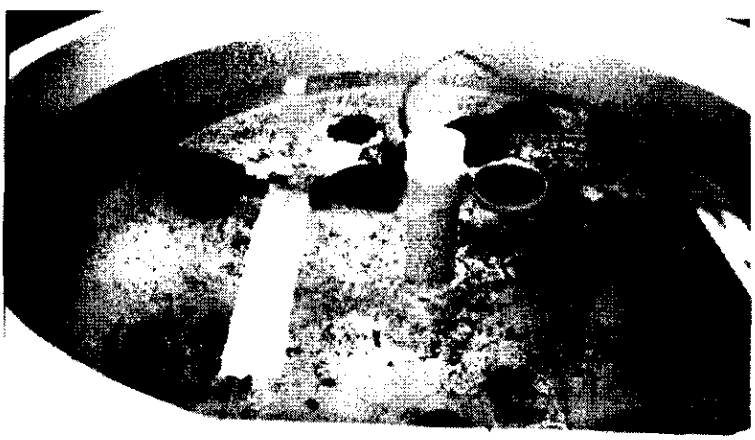


Figura 1.1 Cultivo de moluscos.

La principal preocupación del investigador es contar con un sistema automatizado a través del cual pueda controlar el ambiente (figura 1.1) y estudiar la reproducción y el desarrollo de los peces y moluscos y cultivo del alimento que consumen.

1.3 Adquisición de datos.

Medición y automatización son dos de las componentes claves en nuestro sistema acuícola. Para poder controlar eficientemente el medio ambiente de los cultivos marinos se deben monitorear constantemente sus variables más importantes.

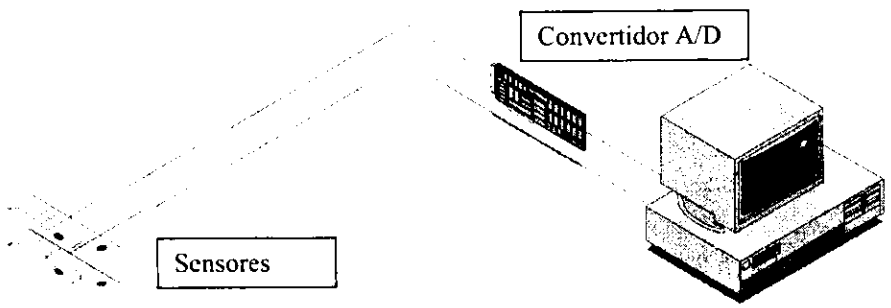


Figura 1.2. Sistema de adquisición de datos.

El primer problema que se presenta al estudiar un fenómeno mediante una computadora es la conversión de datos analógicos a datos digitales o discretos (figura 1.2). Por ejemplo, una representación analógica en una digital de ocho bits tiene únicamente 256 niveles posibles igualmente espaciados. Los valores analógicos entre estos niveles deben ser redondeados al nivel más cercano. Esta representación digital de un valor analógico, produce un error. Los convertidores Analógico/Digital (A/D), esencialmente redondean el dato analógico a su valor digital más cercano, especificando que el error puede ser +/- el dígito menos significativo.

Los transductores o sensores son dispositivos que responden al fenómeno físico a medirse. El fenómeno físico es medido indirectamente, por la respuesta del transductor ante el fenómeno. Los transductores cambian el fenómeno físico en señales eléctricas. Algunos de los transductores más comunes son: termopar, el cual tiene la capacidad de tolerar temperaturas extremas en el rango de los -200° a 1,300° C. Termistores, se trata de otro dispositivo para medir temperaturas, la resistencia (aunque en forma negativa) cambia significativamente con cada grado de aumento en la temperatura y están limitados a un rango entre -50 y 110 grados Celsius. Transductor de fuerza, la fuerza es medida indirectamente por la tensión que produce en un objeto. Un objeto responde a esta tensión experimentando una deformación. Transductores de presión, son un tipo especial de transductor de fuerza, puesto que la presión es definida como una fuerza por unidad de área, el transductor simplemente mide la deformación ocasionada por el efecto de la presión distribuida sobre el área de una membrana elástica.

Los sensores requieren algún tipo de acondicionamiento de la señal: amplificar señales de bajo nivel, aislar, filtrar o excitar. Por ejemplo, los termopares generan voltaje por si mismos que debe amplificarse y requiere una aleación o compensación para que el voltaje sea correctamente interpretado. En otros casos, como con el termistor de resistencia, la resistencia debe convertirse en voltaje.

La calibración es el acto de determinar el grado de incertidumbre asociada a un dispositivo de medición. El proceso se hace en dos pasos. El primer paso, consiste en verificar, que su capacidad éste dentro de su especificación. El segundo paso, es ajustar el dispositivo para reducir la incertidumbre.

El muestreo de un sensor establece que tan frecuentemente deben adquirirse datos en un tiempo dado (figura 1.3). Obviamente, si la señal cambia más rápidamente que la velocidad de conversión del convertidor analógico/digital, se introducen errores en los datos ponderados. Cuando se muestrean datos desde varios canales, un multiplexor analógico conecta cada señal al convertidor analógico/digital a un promedio constante.

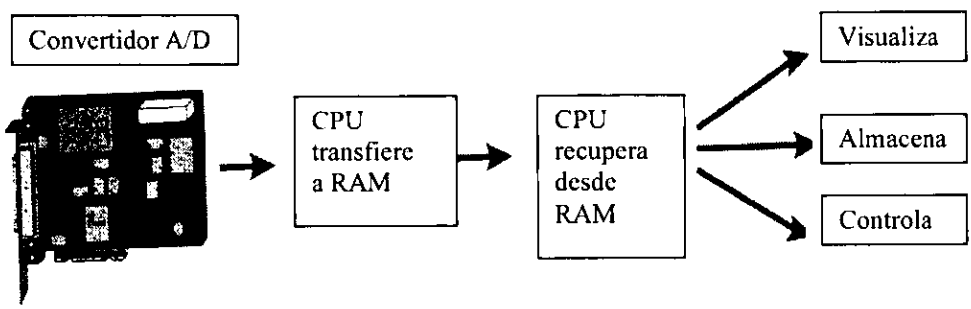


Figura 1.3. Interfaces digitales de Entrada/Salida usadas internamente por la PC.

1.4 Diseño Orientado a Objetos.

El diseño orientado a objetos es una disciplina de ingeniería de software. Su objetivo es organizar las dependencias dentro de un programa, para lograrlo, divide el programa en objetos de tamaño adecuado y a continuación envuelve estos objetos con sus interfaces y canaliza las dependencias entre los objetos a sus interfaces. Por ejemplo, si se tiene un cliente en un objeto y un servidor en otro, ninguna parte del cliente depende del servidor. En vez de ello, el cliente depende de las interfaces que ocultan al servidor. Aún mas, las interfaces no dependen del servidor, sino que las dependencias se han invertido, para que el servidor dependa de sus interfaces. Evitando dependencias transitivas entre el cliente y el servidor.

La importancia de la inversión de dependencias es que se puede reusar tanto el cliente como el servidor en otros contextos. Un cliente que no tenga dependencias sobre un servidor especifico no requiere que el servidor viaje con él, cuando es reusado en otros programas. El cliente únicamente requiere que las interfaces que utiliza sean totalmente implementadas, es decir, alguna clase de servidor debe estar oculto dentro de esas interfaces. Para reusarlo se deben implementar esas interfaces por algún medio.

Otro aspecto importante de la inversión de las dependencias, es que los cambios hechos en alguna parte del programa no se propaguen al resto del programa.

El diseño orientado a objetos es un paradigma de abstracción que permite representar entidades del dominio de un problema como objetos de software con un estado y un comportamiento.

1.5 Método de Booch.

El principal problema al que se enfrenta el desarrollador de software es el de dominar la complejidad. El diseño de software involucra tantos requisitos que compiten entre sí, que plantean problemas incluso para comprenderlos. Por ejemplo, para un sistema de acuacultura, como el que nos ocupa, primero considérense los requisitos electrónicos de adquisición de datos recabados del medio ambiente donde se encuentran los organismos marinos en estudio y luego los requisitos no funcionales, tales como facilidad de uso, rendimiento, capacidad de supervivencia, fiabilidad, etcétera. Al trabajar para organizar esta complejidad a través de un proceso de diseño, hay que pensar en muchas cosas a la vez. La propuesta de muchos investigadores de software (ver Citas al final) y recogidas en el método de Booch es la descomposición orientada a objetos.

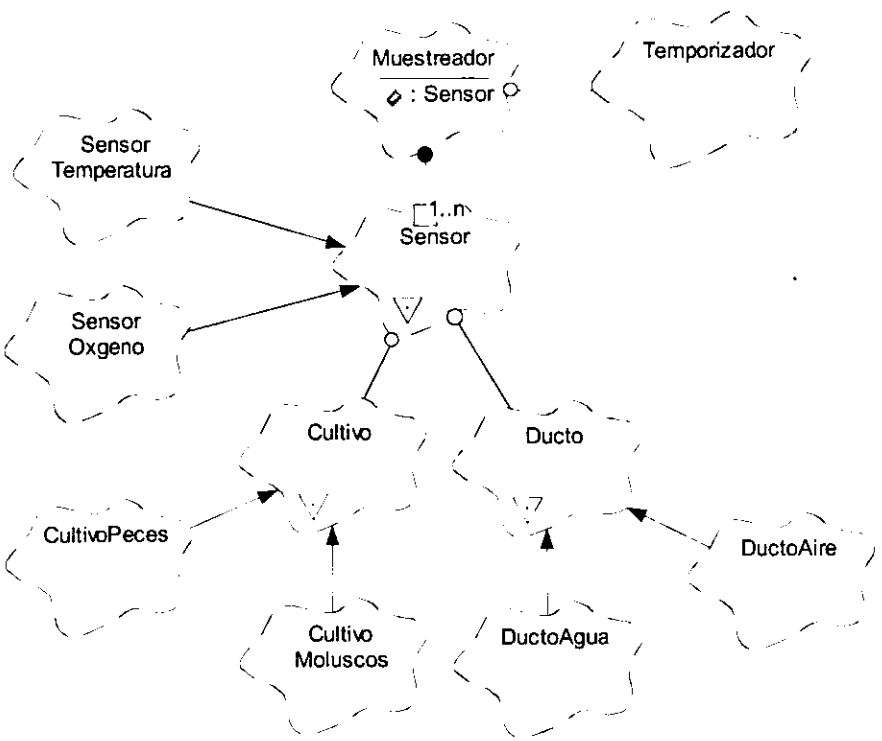


Figura 1.4. Visión lógica del sistema acuícola, mediante el método de Booch.

En la figura 1.4, se muestra la estructura de clases para nuestro sistema acuícola. La lectura de este diagrama nos dice que a través de las responsabilidades de la clase Muestreador, se monitorean n sensores, representados por la clase Sensor, de acuerdo a ciertos intervalos de tiempo proporcionados por la clase Temporizador. Las clases SensorTemperatura y SensorOxigeno heredan de la superclase Sensor su comportamiento, lo mismo ocurre entre

las clases de Cultivo y Ducto. Los sensores están ubicados en los cultivos, de donde toma sus muestras. En este diagrama se muestran otros símbolos, que se explican más adelante.

Además de la complejidad, existen otros problemas no menos importantes como son el ciclo de vida de un sistema, la reutilización de código, la productividad, la flexibilidad, la gestión de desarrollo, la separación de intereses entre componentes y la abstracción por mencionar sólo algunos. Sin embargo, el método de Booch, nos ofrece un rico conjunto de modelos lógicos con los cuales se puede razonar sobre los diferentes aspectos del sistema que se está considerando.

El objetivo es entregar un producto que satisfaga y posiblemente exceda las expectativas del cliente, desarrollado en forma económica ajustado a los tiempos establecidos y flexible al cambio y la adaptación. Booch, en su método propone:

- Una fuerte visión arquitectónica.
- Y la aplicación de un ciclo de vida del desarrollo bien dirigido, iterativo e incremental.

Existen muchas notaciones para el diseño orientado a objetos, sin embargo Booch, nos propone una notación sencilla para representar las decisiones de diseño en una forma concisa y comprensible. Su notación tiene la virtud de capturar muchas clases de decisiones de diseño y presentarlas con una gran fuerza expresiva, porque contiene un amplio repertorio de símbolos para las relaciones, los objetos, las clases etcétera.

1.6 Visual C++.

No importa cual sea la técnica de desarrollo de un sistema de software, ésta debe poder expresarse en algún lenguaje de programación. El desarrollador espera que lenguaje de programación le de soporte a su diseño y le permita ofrecer al usuario un sistema que sea fácil de operar, confiable, bien documentado y que pueda despejar sus dudas en un momento dado, con algún tipo de ayuda.

Para instrumentar un diseño orientado a objetos existen muchos lenguajes disponibles, ¿Porqué Visual C++? En primer lugar, no hay que confundirse con el nombre del producto. Se trata del mismo lenguaje de programación C++. Con las mismas características con las que lo diseño Stroustrup, para que el y sus amigos no tuviesen que programar en C, para hacer más fácil y agradable para el programador individual la escritura de buenos programas.

En nuestro caso muy particular, de un sistema de adquisición de datos, donde es importante la velocidad de proceso, la escritura en disco, el acceso a los puertos serie y que produzca un programa compilado, Visual C++ es una buena elección.

Visual C++ es un ambiente de desarrollo en el que los programas en C++ son parte de la estructura de aplicación de la MFC (biblioteca de clases de base de Microsoft) y que proporciona dos herramientas para el desarrollo: el asistente para aplicaciones y el asistente para clases.

Visual C++ consta además de un compilador para código C o C++, un depurador, un visualizador de código, en el que se escribe la aplicación partiendo de cero, proporciona también ayuda en línea y un compilador de recursos: menús, mapas de bits, barras de herramientas, etcétera.

En una palabra Visual C++, ofrece un ambiente de aplicación adecuado para el desarrollo orientado a objetos.

Capítulo 2.

Requerimientos.

En este capítulo se inicia el caso de estudio del sistema de adquisición de datos aplicado a la acuacultura. La investigación de requerimientos se inicia con la inspección del sitio en donde se desea implantar el nuevo sistema y entrevistando al usuario para conocer sus requerimientos.

El objeto de las dos actividades, tanto de la entrevista como de la inspección es obtener una descripción completa del sistema. En esta etapa únicamente se intenta conocer qué hace el sistema y establecer un vocabulario común extraído del dominio del problema.

2.1 Entrevista e inspección.

La entrevista se da dentro del marco de la Unidad Pichilingue (figura 2.1) de la Universidad Autónoma de Baja California Sur con el Dr. Carlos Cáceres Martínez.

Figura 2.1 Unidad Pichilingue.

Bernabé Ortiz y Herbert (BOH). ¿Qué tipo de acuacultura se desarrolla en la Unidad Pichilingue?
Carlos Cácerez Martínez (CCM). No se desarrolla propiamente acuacultura sino que se trabaja con los estudios básicos que permiten el desarrollo de técnicas de acuacultura.

Dentro de este marco se trabaja en dos líneas, una que es acuacultura de moluscos y otra la acuacultura de peces marinos a cargo de investigadores invitados.
BOH. ¿Qué especie de moluscos se estudian?
CCM. Primero en ambientes controlados se trabaja en el manejo y control de la reproducción de pectínidos como son: la almeja catarina, almeja mano de león, callo de hacha, madre perla y concha nácar (figura 2.2). Y posteriormente la engorda de juveniles en el campo.

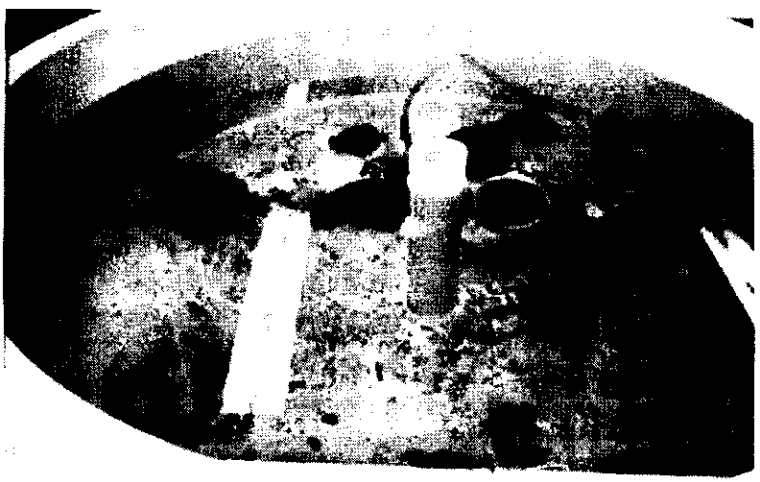


Figura 2.2 Cultivo de moluscos.

BOH. ¿Cómo se llevan a cabo los estudios?
CCM. En el interior del laboratorio se requiere sobre todo de agua, las características más importantes es que estén libres de partículas con tamaño superior a un micrómetro, debido a que este tipo de especie es filtrador, se alimenta de partículas en suspención, para controlar su reproducción se deben alimentar con lo que nosotros queremos, no con lo que venga en el agua. Debemos asegurarnos que el agua venga libre de esas partículas, para nosotros introducir las partículas que queremos. Las partículas que se introducen es materia viva, sobre todo fitoplancton y bacterias.

Se cuenta con bombas para hacer llegar el agua a los laboratorios, el agua se hace pasar previamente por filtros de arena, para eliminar partículas de hasta 60 micrómetros, después, el agua se hace pasar por una serie de filtro de poro decreciente, hasta que se tiene el agua libre de esas partículas.

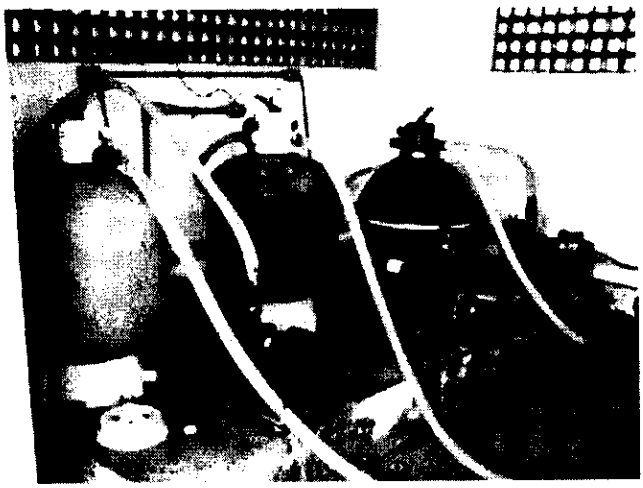


Figura 2.3 Filtros de agua.

Sin embargo, algunas veces es necesario también desinfectar el agua. Es decir, después de eliminar las partículas, el agua se hace pasar a través de una lampara de luz ultravioleta, la cual permite oxidar (destruir) las membranas de los microorganismos que se hayan filtrado e inactivar su capacidad de reproducción.

BOH. ¿En que consiste la irradiación?
CCM. Las condiciones de la irradiación de luz ultravioleta para desinfectar el agua, pueden variar. Las lamparas trabajan internamente con vapor de mercurio, después de un cierto tiempo pierden intensidad y al bajar la capacidad de irradiación de las lámparas, la capacidad bactericida o destructiva de las membranas de las bacterias disminuye.

Otro aspecto importante es la presión del agua. En el sistema decreciente de filtración se aumenta la presión del agua sobre la bomba, afectándola. Es decir, los cartuchos de filtrado, pierden su capacidad de filtración con el uso, situación que da como resultado un aumento en la presión del agua.

BOH. ¿Qué tipo de bombas usan?
CCM. Las bombas que envían el agua a las tinas experimentales son bombas centrífugas, de impelente cerrado para aumentar su potencia, son del tipo del que se usa en las albercas. Figura 2.3.

Sin embargo, la bomba principal, se encuentra sumergida en el mar, la cual bombea el agua durante las 24 horas del día y se cambia cada seis meses (figura 2.4). Esta bomba es del tipo sumergible de impelente abierto. Es de impelente abierto para permitir, la entrada y destrucción de animales, sin embargo, existe la posibilidad de que la bomba se atasque con algún objeto como alguna bolsa de plástico, como ya ha ocurrido.

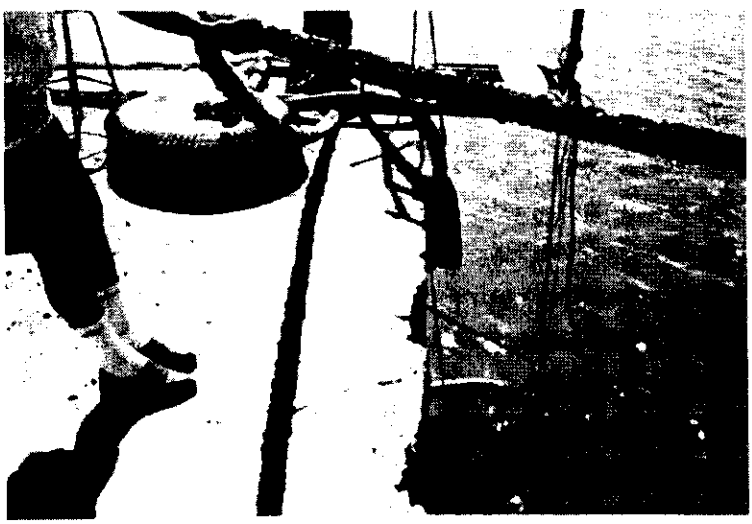


Figura 2.4 Bombeo de agua de mar.

BOH. ¿Además del tratamiento del que hemos hablado sobre el agua, agregan algún tipo de nutrientes?
CCM. Los animales cultivados se alimentan agregando al agua concentraciones conocidas de nutrientes. Los nutrientes también se cultivan (figura 2.5) y se tienen controlados y con el fin de optimizar el alimento no se agregan sino hasta que han sido consumidos en su mayor parte.

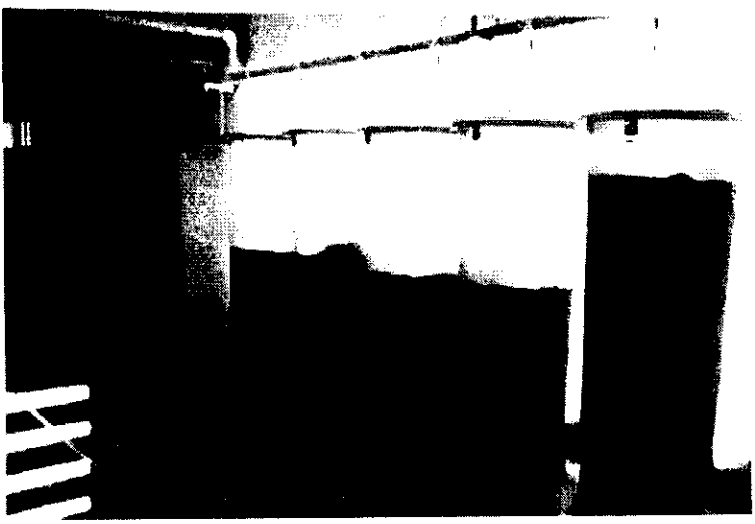


Figura 2.5 Cultivo de microalgas para el alimento de peces y moluscos.

BOH. ¿Existen planes de cultivo establecidos, bases de datos con la información histórica sobre los planes de cultivo?
CCM. Se tiene un programa anual de trabajo, el cual es revisado semestralmente. Este programa está en concordancia con los periodos reproductivos de los animales.

Actualmente la especie que más se domina es la almeja catarina, tanto la reproducción como la crianza larvaria, con el estudio de esta especie se produce información científica y la preparación de los estudiantes de maestría y se tiene un programa anual de siembra de la especie a fin de asegurar contar con ejemplares para el año siguiente y dar seguimiento a los experimentos.

Se han generado bases de datos, que son de hecho algunos trabajos de tesis en los que se tiene información sobre cómo han respondido las generaciones animales en estudio con que se ha contado.

BOH. ¿Los planes de cultivo para larvas son diferentes para el desarrollo que para engorda de las especies?
CCM. Como te había mencionado la engorda de juveniles se hace en el campo y ahí no tenemos ningún control de los parámetros ambientales, sin embargo, en el laboratorio se

tienen varios tipos de control, principalmente sobre el agua. Se tienen dos tipos de agua. Un tipo que se llamamos agua cruda y que solamente fue filtrada dos veces sobre filtros de arena para eliminar partículas de hasta 60 micrómetros de tamaño. El agua cruda está a la temperatura ambiente, se distribuye por gravedad. Es decir, el agua de mar se hace pasar por filtros de arena y es bombeada a un tanque elevado y por gravedad es distribuida directamente en depósitos colocados en los laboratorios, sin embargo antes de llegar a estos depósitos, el agua se filtra a fin de eliminar las partículas que lograron pasar los filtros de arena o los animales que se hayan desarrollado en el transcurso de este proceso. El agua cruda, sin embargo tiene partículas de diversos tamaños por arriba de los 60 micrómetros.

El otro tipo de agua es el agua tratada. Es este caso, el agua cruda es pasada por filtros de diferentes retículas que van desde los 60, 20, 15, 10, 5 y a una micra para eliminar las partículas sólidas en el fluido y después si es necesario se irradian con lámparas de luz ultravioleta para desinfectarla.

BOH. ¿El agua recibe un tratamiento diferente de un cultivo a otro?
CCM. El agua es la misma para todos los estudios, sin embargo, algunas veces es necesario desinfectarla y otras veces no lo es.

Además de la distribución del agua mediante el uso de bombas centrífugas, se tienen controles sobre el aire y la temperatura. Se cuenta con un sistema de distribución de aire mediante el uso de bombas rotatorias de flujo radial, es decir, el aire no se comprime, sino solamente se desplaza.

BOH. ¿El aire, recibe un tratamiento de filtración?
CCM. El aire se usa directamente (figura 2.6), sin embargo, en algunos casos en necesario filtrarlo para eliminar partículas de hasta cinco micrómetros.

Figura 2.6 Cultivo de peces.

BOH. ¿Cómo se controla la temperatura?
CCM. La temperatura, se controla directamente sobre los recipientes de cultivo. Mediante el uso de calentadores, controlados con termostatos de inmersión se mantiene la temperatura constante a un valor establecido.

BOH. ¿Qué otro tipo de parámetros relacionados con el agua desean conocer o controlar?
CCM. Bueno actualmente no tenemos posibilidad de saber si durante el bombeo, se ha producido una sobresaturación de gas, si se sobresatura con gases el agua, envenenamos a los animales. Hasta ahora para prevenir esta situación, se deja caer el agua sobre un tanque distribuidor, para que con el golpe, el agua se aireé y se desgasifique, sin embargo, no se cuenta con algún dispositivo, que nos permita, conocer si efectivamente el agua ha sido desgasificada.

BOH. ¿Lo que deseas medir es la cantidad de oxígeno disuelto en el agua?
CCM. De oxígeno o cualquier otro gas, Porque puede tratarse también de sobresaturación de nitrógeno. Recordemos que al aumentar la presión (hasta de 20 libras) para que el agua pueda pasar por los diferentes niveles de filtración, también aumentamos la presión sobre los gases, produciendo la sobresaturación.

Otro parámetro que deseamos registrar es la temperatura con que el agua ingresa a nuestros laboratorios, con el fin de elaborar un patrón de distribución de temperaturas y conocer cuando se presentan puntos críticos y determinar con anticipación la presencia de algún problema durante la noche o por la mañana.

Observación. Un sensor que pueda monitorear la temperatura del agua y pueda enviar una señal de alarma que anticipe la presencia de algún tipo de problema, derivado de una temperatura fuera de control.

Otra situación importante que se nos presenta es que los equipos destinados para bombear aire o agua se controlan manualmente, en caso de presentarse un problema de cualquier índole, no tenemos posibilidades de anticiparlo, para poder intervenir y evitar daños.

BOH. ¿El desarrollo de técnicas de acuacultura para los peces es similar que para las almejas?
CCM. En el área de peces se presentan otro tipo de necesidades, en este caso se trabaja con un sistema de recirculación del agua. Es decir el agua se reutiliza, para mantenerla caliente.

BOH. ¿Cómo aseguran la calidad del agua?
CCM. Al reciclar el agua se hace pasar por un filtro biológico, que convierte las concentraciones de amonio y otros compuestos excretados por los organismos en nitritos o nitratos de amonio que son formas no tóxicas que pueden estar hasta cierta concentración en el agua.

BOH. ¿Cómo alimentan a los peces?
CCM. Con alimento balanceado. En este caso se producen excretas líquidas y sólidas, que finalmente van a parar a la cama de bacterias, este mismo filtro sirve para retener partículas.

BOH. ¿También agregan oxígeno al agua?
CCM. Agregan aire y cuentan con bombas que recirculan el agua. En este caso de recirculación de agua debe asegurarse que las camas de filtración de bacterias estén bien aireadas para que haya poblaciones de bacteria aeróbicas y no anaeróbicas.

En el caso de los moluscos, el agua, no es recirculada, sino que se envía a un manglar cercano, este tipo de agua, no tiene ningún tipo de elemento tóxico que pudiera producir algún efecto negativo sobre el manglar.

BOH. ¿Se llevan a cabo estudios sobre las sedimentaciones reunidas en los filtros?
CCM. No.

BOH. ¿Se hacen estudios sobre crustáceos?
CCM. No en este momento, quizá a futuro.

Como investigador, me interesa conocer el desempeño de los equipos involucrados en los proyectos de investigación. Por ejemplo, conocer la curva de rendimiento de algún motor, para en caso de que ésta vaya en descenso, estar preparado con un motor extra, a fin de evitar un colapso en la investigación.

BOH. ¿La turbidez, es un factor importante en sus estudios?
CCM. Hay periodos en los que el nivel de turbidez o de sólidos es muy alto, por ejemplo, en época de vientos, los filtros deben limpiarse con mayor frecuencia. Nos ayudaría mucho, tener conocimiento sobre las tendencias de filtrado, a fin de limpiar los filtros en el momento más conveniente.

BOH. ¿Bajo que condiciones opera el área de cultivo del fitoplancton y zooplancton?
CCM. Este laboratorio trabaja bajo ciertas condiciones controladas de temperatura e iluminación. Actualmente utilizamos un termómetro de máximos y mínimos, para registrar cada cierto tiempo la temperatura del laboratorio en una bitácora.

Observación. Un sensor que pueda monitorear, la temperatura del laboratorio con relación al rendimiento del equipo de enfriamiento, a fin de conocer, el desempeño y el gasto del equipo en las diferentes épocas de año: primavera, verano, otoño e invierno.

BOH. ¿En este laboratorio como se distribuye el agua y el aire en los recipientes de cultivo?
CCM. El agua se distribuye a los tanques de cultivo (de material traslúcido, de forma cilíndrica y fondo cónico) a través de una bomba conectada a filtradores diferenciales. Una vez que el tanque se ha llenado de agua, se hace un caldo de cultivo, fertilizando y enriqueciendo el agua con nutrientes.

BOH. ¿Cómo determinan las condiciones de cosecha?
CCM. Existen varias técnicas de cultivo, denominadas: estáticos, continuos o semicontinuos.

En nuestro caso, utilizamos la técnica de cultivo estático, en la que contamos con muchos años de experiencia. Esta técnica de cultivo es manual. No hay un flujo continuo de agua, sino que el ingreso y el filtrado del agua se operan manualmente. Las cosechas se llevan a cabo directamente en forma manual, extrayendo del cultivo los volúmenes adecuados. Por experiencia, sabemos cada cuantos días, el cultivo tiene la densidad adecuada, no es necesario contar la cantidad de cepas por unidad de volumen excepto para experimentos muy precisos. Hasta ahora hemos logrado obtener con esta técnica de cultivo una muy buena calidad sanitaria.

Sin embargo, es necesario conocer la curva de crecimiento de la población que estamos cultivando y saber cuando se han alcanzado las condiciones de densidad que nos interesan.

BOH. ¿Qué tipo de problemas se tienen en este laboratorio?
CCM. Existen básicamente dos tipos de problemas. El primero es el ingreso del aire al laboratorio. Inicialmente filtrábamos aire del exterior, sin embargo en verano, el aire ingresaba muy caliente y se decidió colectar el aire del interior del laboratorio y recircularlo y contar para ello con un sistema de filtración apropiado que evite contaminación. Los sistemas de filtración que se recomiendan para estos casos son: pasar el aire por un ozonizador, para oxidar los animales y matarlos o por una lámpara de luz ultravioleta.

El segundo problema son los altos costos del ambiente controlado. Actualmente estamos analizando una técnica de cultivo en el exterior que se aplica en las Filipinas y que consiste en un sistema de flujo continuo a través de un serpentín de tubos translúcidos e irradiado que después de un recorrido de algunos metros, la población es suficientemente útil. No obstante que las temperaturas a las que operan son inferiores a las nuestras, sería cosa de conseguir alguna especie de alga que se adapte a nuestras temperaturas.

BOH. ¿En que consisten las técnicas de cultivo continuos o semicontinuos?
CCM. Se trata de sistemas automatizados, con un ingreso de agua filtrada y esterilizada automáticamente y a través de un dispositivo se determina la concentración de nutrientes y mediante una válvula controlar las cosechar

El problema en este tipo de técnicas es que si no se tienen los controles necesarios se reduce la calidad sanitaria de los cultivos. En sistema de tipo estático, la población de bacterias crece en igual proporción al de nuestros nutrientes y en un sistema continuo o semicontinuo la población de bacterias se puede estabilizar en un número inapropiado o dar lugar a alguna bacteria indeseable.

BOH. ¿Qué tipo de dietas se tienen programadas para alimentar a los cultivos?
CCM. En el caso de moluscos se están ofreciendo dietas de animales vivos y algas unicelulares no de cereales. La combinación de animales vivos y algas conforman una dieta y se está investigando el impacto de estas dietas en término de sobrevivencia, madurez monádica, crecimiento, etcétera.

BOH. ¿Qué tipo de precauciones se toman para mantener los cultivos sanos?
CCM. En el caso de los moluscos aplicamos el principio de prevenir antes que curar. Para ello lo que hacemos es garantizar que el agua llegue libre de partículas y lo suficientemente tratada para asegurar su calidad sanitaria. Cuando se ha presentado algún problema de tipo patológico, se ha preferido, sacrificar a la población para no propalarlo.

Actualmente se está trabajando con callo de hacha, una especie sumamente frágil en estado larvario y ha sido necesario utilizar antibióticos. Es este caso, se estudia el tipo y la cantidad de antibiótico necesario para asegurar la sobrevivencia larvaria, por otro lado, no se trata de un problema patológico, sino un problema de resistencia a la presencia de bacterias de las larvas, esperamos que con una dieta adecuada, garantizar suficientes defensas en las larvas para que puedan sobrevivir.

Los moluscos en particular no tienen un sistema inmunológico, es decir no tienen memoria de enfermedades, no hay manera de vacunarlos, el sistema de defensa con que cuentan está fundado en su sangre. Lo cierto es que en lugar de sangre tienen molinfa, en la que se encuentran grupos de hemocitos, que se encargan de la defensa. Tratamos de producir una dieta para que su sistema de hemolinfa sea autosuficiente para defenderse.

Actualmente no se trabaja con inmunología. Con respecto a bacteriología, se ha llevado un registro sobre las condiciones sanitarias con relación a la cantidad de bacterias en nuestra agua. Estos registros nos permiten programar medidas sanitarias que tiendan a evitar incubaciones indeseables. Se hacen registros diarios o semanales.

BOH. ¿Qué tipo de parámetros químicos les interesa medir?
CCM. Los básicos son: oxígeno disuelto, salinidad, temperatura y Ph. Y como parámetro físico la turbidez.

BOH. ¿Se cuenta con una base de datos con la información de los resultados de los análisis que se llevan a cabo en los distintos aparatos con que cuenta la unidad?
CCM. Sería muy importante contar con una base de datos compartida, actualmente, cada investigador mantiene su información en su disco.

BOH. ¿Tienes referencias de algún laboratorio que esté automatizado o semiautomatizado?
CCM. Conozco uno totalmente automatizado, específicamente en lo que se refiere al control de la calidad del agua. A través de dispositivos es monitoreada permanentemente la presión o flujo del agua y los parámetros anteriormente mencionados: salinidad, oxigeno, pH, temperatura y algún compuesto tóxico, sobre todo amonio. En un monitor, mediante un sistema de ventanas el usuario puede consultar las tendencias del flujo del agua o de los parámetros, en forma individual o por grupos, por día o por periodo. Hay que considerar, que este tipo de sistemas acuícolas son de carácter comercial y sus riegos e intereses son en cierta forma diferente a una unidad de investigación. En su caso una sobresaturación de amonio, acabaría con todos los peces y adiós negocio. Con nosotros quizá no sea necesario ni prudente tener todo el proceso controlado, pero si es necesario tener un control sobre la calidad de agua antes de utilizarla.

2.2 Documento de requerimientos.

Después de haber obtenido de parte del Dr. Cáceres su declaración sobre lo que hace el sistema, el siguiente paso es identificar los puntos funcionales o comportamiento del nuevo sistema.

En este inciso se describen los requerimientos de software para un nuevo producto que proporcione un mejor control sobre el equipo, los cultivos, el agua, el aire y los diversos elementos que se ven involucrados en los cultivos de peces marinos y moluscos.

El sistema de adquisición de datos trata sobre las diferentes variables a medir. Las funciones detalladas del sistema son las siguientes:

1. Monitorear los cultivos de moluscos para determinar:
   - Las concentraciones de nutrientes de fitoplancton y zooplancton disueltos en el agua.
   - Las concentraciones de sales disueltas en el agua.
   - El pH del agua.
   - La temperatura del agua en los cultivos de moluscos.
   - La intensidad de la luz ultravioleta en el agua.
2. Monitorear los cultivos de peces para determinar:
   - Las concentraciones de amonio disueltas en el agua.
   - Las concentraciones de nitratos disueltas en el agua.
   - La temperatura del agua.
3. Monitorear las tuberias para determinar:
   - La presión del agua de la red de tuberías que abastecen los recipientes de los cultivos. Con ello se busca tener un mejor control sobre los cartuchos de filtrado. Una disminución en la presión, puede significar la necesidad de sustituir un cartucho de filtrado
   - Monitoreo de la presión del agua de la red de tuberías que abastecen los depósitos secundarios. Con ello se busca tener un mejor control sobre el rendimiento de las bombas. Una disminución de la presión, puede significar la necesidad de sustituir una bomba.

Nos encontramos ante una típica lista de requerimiento inicial que contiene las especificaciones que son consideradas esenciales para el nuevo producto, sin intentar describir demasiado la estructura del sistema, ni de que manera opera.

El análisis se inicia considerando el hardware para el nuevo sistema. Se muestra la interfaz de usuario o dicho de otra manera los puntos funcionales del sistema y sus relaciones con sus componentes.

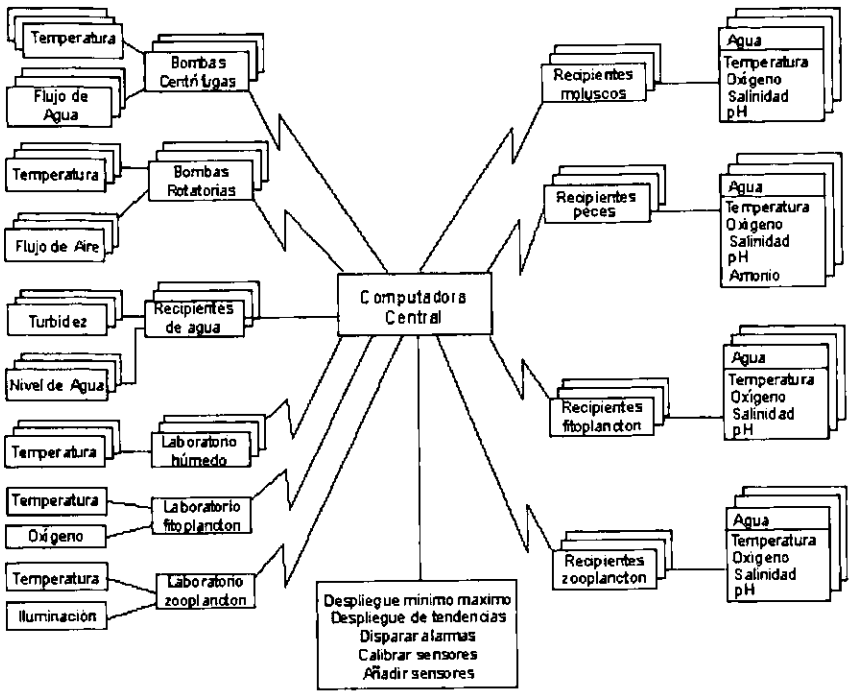


Figura 2.7. Hardware para el sistema de adquisición de datos.

Es muy importante establecer las delimitaciones o fronteras del sistema antes de iniciar las tareas de análisis y diseño del software.

La figura 2.7 muestra la estrategia que se va a seguir para desarrollar el nuevo sistema. El hardware consta de los elementos siguientes:

- Una computadora personal, con las características usuales, la cual albergara la aplicación, un puñado de objetos C++, que se encargaran del monitoreo y control para el sistema acuícola.
- El temporizador del sistema operativo, el cual notifica a una aplicación cuando ha transcurrido un determinado intervalo de tiempo. El muestreo o medición de variables se ejecuta cada cierto intervalo de tiempo, establecido por el usuario. Por ejemplo, la temperatura se puede decidir monitorear cada minuto, luego el sistema cada minuto en forma recurrente, llevará a cabo las mediciones.
- El puerto paralelo es utilizado con frecuencia como una salida de propósito general de 8 bits para manejar una amplia gama de impresoras, discos externos, etcétera. Se pueden usar estas líneas para controlar el hardware que elijamos. En nuestro caso los sensores.

Como puede verse el sistema debe proporcionar monitorización automática en tiempo real, aunque las condiciones cambien lentamente y además debe ofrecer al usuario un ambiente interactivo, el cual se describe a través de la Tabla 2.1.

Inicializar el sistema.
Configurar los sensores de acuerdo al plan de cultivo.
Iniciar el proceso de adquisición.
Visualizar para un sensor seleccionado sus valores.
Visualizar para un sensor seleccionado la tendencia durante un período específico.
Visualizar situaciones de alarma. Cuando una medida se dispara fuera de sus límites normales, por ejemplo, cuando el nivel del agua de un tanque de almacenamiento sale de sus límites o cuando la temperatura se eleva por encima de sus valores tolerados.
Terminar el proceso de adquisición.

Tabla 2.1. Ambiente del usuario.

Capítulo 3.

Análisis de requerimientos.

Por objeto entiendo todo aquello que puede llenar un espacio, de tal manera que cualquier otro objeto quede excluido de él.
Descartes.

3.1 Modelo de objetos: estudio y modelación del dominio del problema.

La ingeniería de software es un balance entre dos fuerzas diametralmente opuestas, debe desarrollarse rápidamente y reportar beneficios desde las primeras etapas de su ciclo de vida. Una de las ventajas de la metodología orientada a objetos, es que es posible hacerlo. Para ello Booch propone un diseño que sea fácil de mantener sobre un largo ciclo de vida y que pueda ser implantado rápidamente.

Muchos métodos tradicionales separan en dos fases distintas el análisis y el diseño del ciclo de vida del software. Frecuentemente la notación y los métodos usados durante el análisis son diferentes de la notación y los métodos usados durante el diseño. Al final del análisis debe hacerse una traslación para convertir los diagramas de análisis convenientes para el diseño. Esta barrera de traslación puede ser difícil de cruzar. Cuando un problema del análisis se encuentra en el diseño, debemos regresar al análisis para resolver éste.

En este capítulo, se considera la noción de que el análisis y el diseño son esfuerzos estrechamente concurrentes, en donde el análisis fluye hacia el diseño y el diseño fluye hacia el análisis y terminando con actividades del diseño, pero sin una clara separación de estas actividades en ninguna parte del proceso.

En nuestro proyecto, una vez que se han establecido las fronteras del problema, a partir de los requerimientos del usuario, el siguiente paso consiste en identificar las clases y objetos del dominio del problema y construir un modelo de objetos.

Iniciamos esta etapa con un diagrama de clases, una idea muy general del sistema de adquisición de datos para el requerimiento 1.

Requerimiento 1. Muestreador, Temporizador, Sensor, Cultivo.

4. Monitorear los cultivos de moluscos para determinar:
   - Las concentraciones de nutrientes de fitoplancton y zooplancton disueltos en el agua.
   - Las concentraciones de sales en los cultivos de moluscos.
   - El pH del agua.
   - La temperatura del agua en los cultivos de moluscos.
   - La intensidad de la luz ultravioleta en el agua en los cultivos de moluscos.

Iniciamos trasladando este requerimiento en su correspondiente caso de uso.

Caso de uso 1. Requisitos de muestreo de nutrientes, concentraciones de sales, pH, temperatura e intensidad de la luz ultravioleta del agua. Cuando un usuario inicia el periodo de muestreo, el sistema monitorea los sensores recurrentemente cada cierto tiempo. Los datos adquiridos del cultivo de moluscos se muestran en pantalla. Si la cantidad de un sensor en particular es menor que un valor establecido, se dispara la alarma correspondiente.

A continuación se crea un simple modelo estático para el muestreo de nutrientes en el cultivo de moluscos. En el caso de uso 1, hallamos que tenemos que modelar los conceptos de muestreador, temporizador, sensor, alarma, vista y cultivo.

El diagrama de la figura 3.1 presenta las relaciones estáticas entre las principales abstracciones del sistema de adquisición de datos. Tenemos que la clase **Vista** posee por referencia a la clase **Muestreador**. La clase **Vista** es la interfaz de usuario, a través de la cual el usuario se comunica con el sistema. A continuación se ve que la clase **Muestreador**, usa la clase **Temporizador**, del que recibe tics de reloj cada cierto tiempo. Por cada interrupción, el Muestreador toma un valor del **SensorNutrientes**. El Muestreador posee un sensor del que recibe los datos de nutrientes adquiridos. El SensorNutrientes usa el cultivo de moluscos del cual toma valores analógicos que representan cantidades de nutrientes del sistema de adquisición de datos. Finalmente vemos que la clase Muestreador usa la clase **Alarma** para enviar mensajes de advertencia cuando un valor sale de sus límites establecidos.

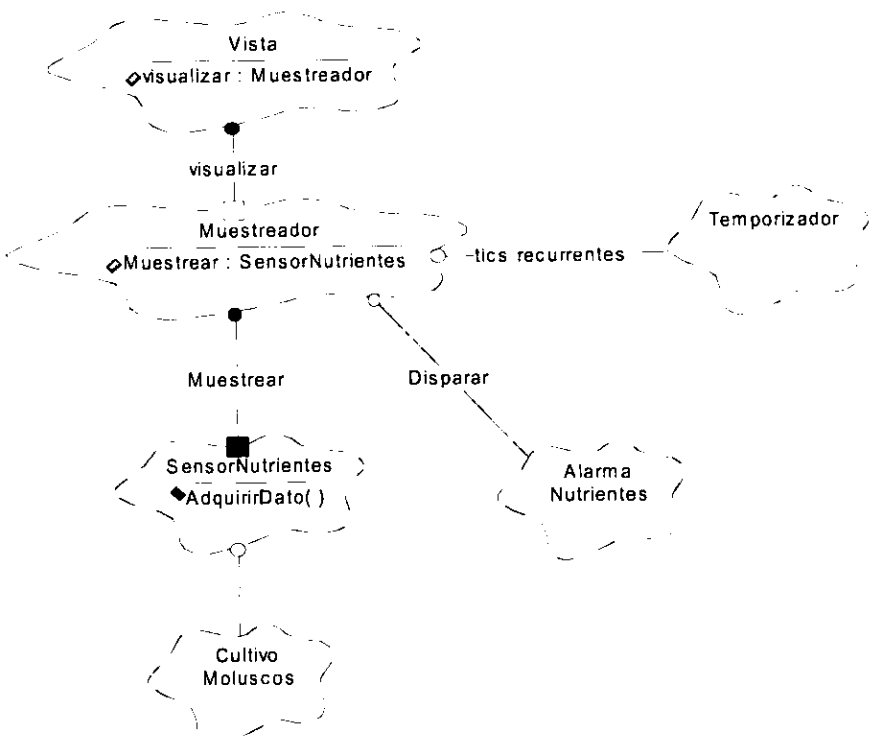


Figura 3.1 Modelo estático para Muestreador, Temporizador, Sensor y Vista.

Si ejecutamos este modelo para ver un poco del comportamiento de este escenario, veremos como trabaja.

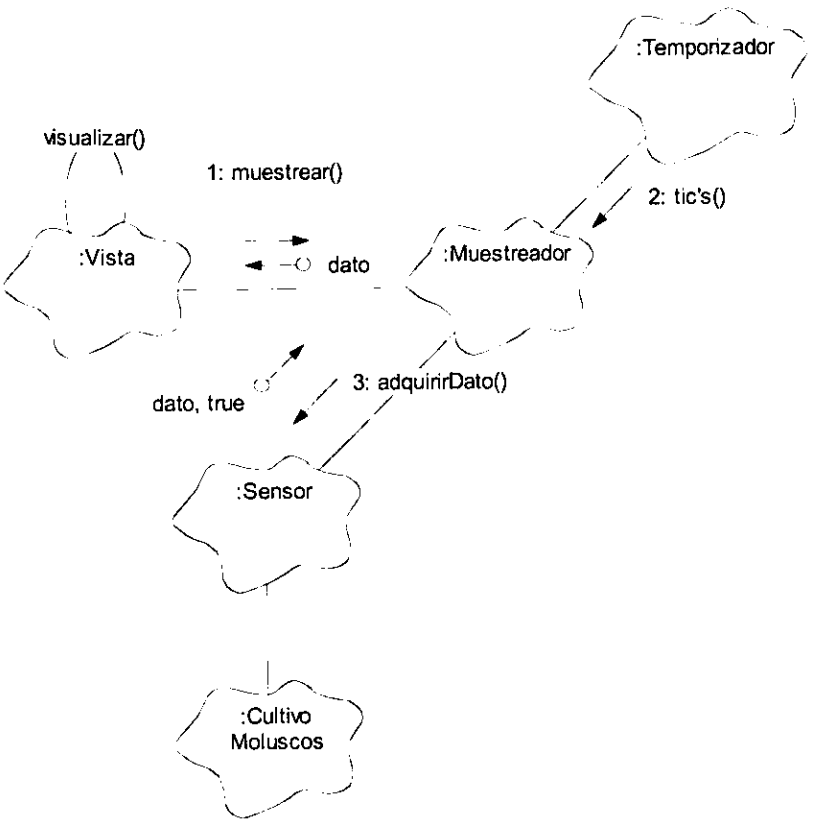


Figura 3.2 Modelo dinámico

La Figura 3.2 muestra lo siguiente. Suponga que un usuario trata de monitorear un sensor de nutrientes ubicado en algún cultivo de moluscos. El objeto **Vista** envía un mensaje a **Muestreador** para que inicie el muestreo. El **Muestreador** entonces por cada tic del **Temporizador** adquiere un dato del **SensorNutrientes**, el cual es mostrado en la **Vista**.

¿Pero que ocurre cuando el valor de un sensor sale del rango de valores permitidos? ¿Cómo advierte el sistema de esta situación? De acuerdo el requerimiento 1, el sistema debe disparar una alarma, lo cual se puede entender como un sonido de chicharra y una luz roja en la pantalla. La figura 3.3 muestra este escenario.

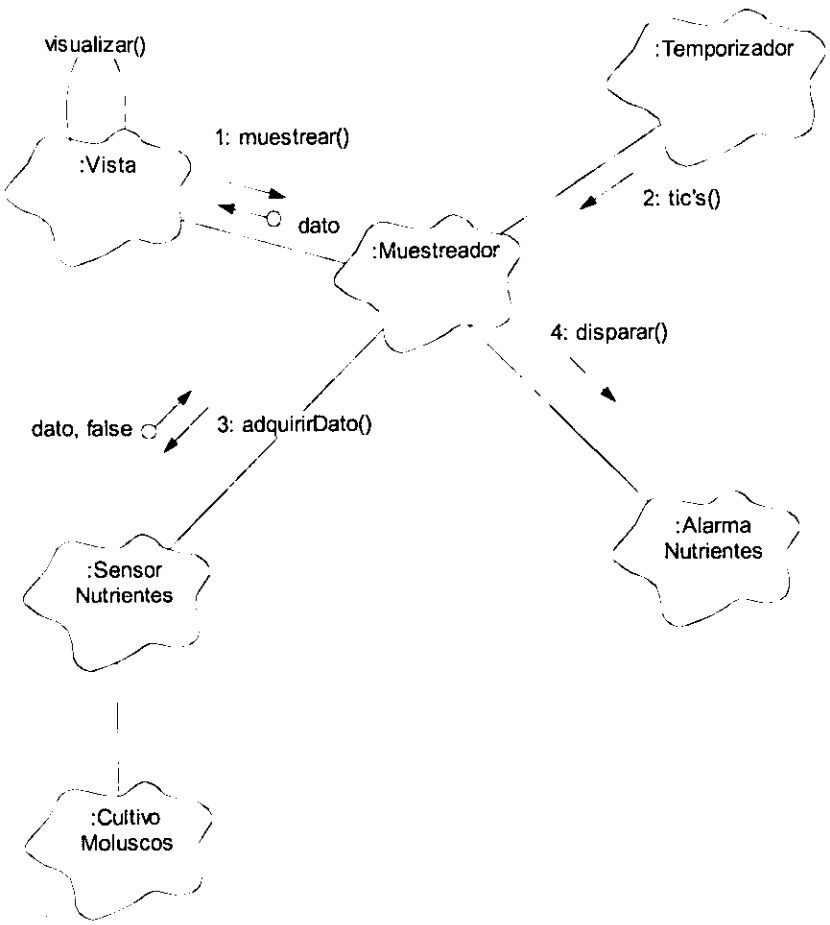


Figura 3.3 Disparo de alarma.

El objeto Muestreador es un concepto abstracto, que tiene varias responsabilidades. Cada cierto tiempo debe enviar un mensaje al objeto SensorNutrientes de donde obtiene una muestra y un valor booleano que indica si la muestra está dentro del rango(true) o no (false), a continuación, el objeto Muestreador, envía un mensaje al objeto Vista para que visualice el valor obtenido. En caso de recibir una muestra fuera de rango, el objeto Muestreador envía un mensaje al objeto AlarmaNutrientes para que dispare la chicharra.

La idea de trasladar los requerimientos en sus correspondientes diagramas estáticos y dinámicos del sistema es ir descubriendo el comportamiento de los objetos tangibles del sistema. Los diagramas de las figuras 3.1, 3.2 y 3.3, anteriores no son mas que un punto inicial.

Cuando se está en la etapa de análisis se le dan muchas vueltas a los problemas. La ventaja del método de Booch es que las ideas llegan como un torrente, el problema sería que impidiera que llegaran, sin embargo, uno puede quedar atrapado y no saber como manejar todas estas ideas.

Continuando con el análisis para el requerimiento 1, notamos que se requieren diversos tipos de muestreo: nutrientes, temperatura, sales, etcétera. Para cada tipo de muestreo se requiere un sensor diferente, sin embargo podemos preguntarnos que tienen en común este tipo de sensores. En la tabla 3.1 mostramos una caracterización genérica para la clase Sensor.

Identidad	NombreSensor
Responsabilidad	Llevar la cuenta de sus mediciones
Operaciones	Medición actual Fijar medición alta Fijar medición baja
Atributos	Tipo de dato

Tabla 3.1 Requisitos para la clase sensor.

De los requerimientos se desprende, que el sistema en un momento dado, estará monitoreando diversos sensores físicos, los cuales estarán ubicados en los cultivos de moluscos. Por ejemplo, La clase SensorTemperatura es una analogía de los sensores físicos de temperatura, su responsabilidad es llevar la cuenta de la temperatura actual. Los sensores como cualquier otro instrumento de medición debe calibrase para acondicionar su señal (ver la sección 1.3 Adquisición de datos, del capitulo I. Fundamentación). La MediciónActual es el valor recientemente adquirido, se considera que cada valor de un sensor de temperatura se representa por un número de punto fijo, pero las operaciones de FijarMediciónAlta y FijarMediciónBaja, se derivan directamente de los requisitos, que obligan a proporcionar un mecanismo para calibrar cada sensor, para adecuarse a valores reales conocidos.

La Figura 3.4 muestra que para un sensor calibrado, un valor de 0.6 corresponde una temperatura de 10° C. En este aspecto todos los sensores guardan un comportamiento similar, todas las clases deben saber como calibrarse a si mismas, esto puede hacerse mediante interpolación lineal entre dos puntos de datos conocidos.

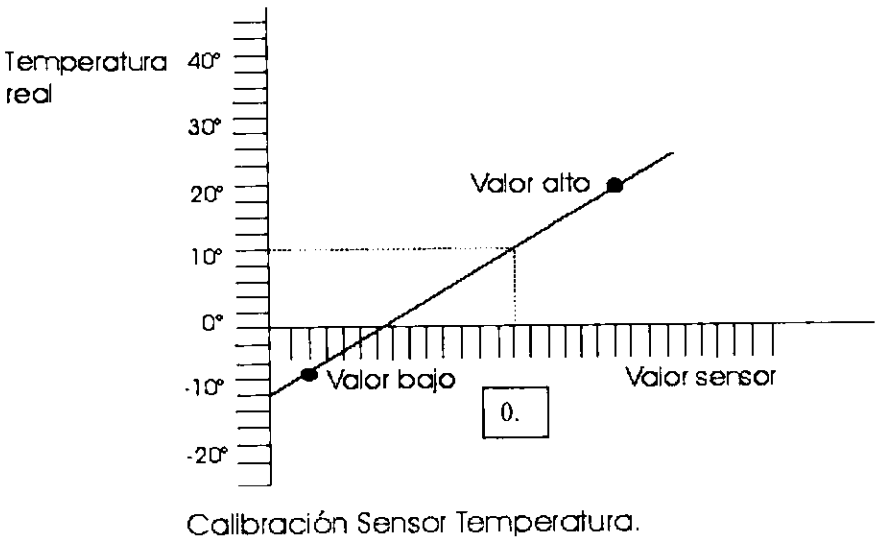


Figura 3.4 Calibración de sensor temperatura.

Un requisito más que deben cumplir los sensores de temperatura y de presión (como veremos más adelante) es el de proporcionar la tendencia a lo largo de un intervalo de tiempo. En este caso, se puede expresar la tendencia como el promedio de temperaturas en un intervalo de tiempo, es decir números en punto flotante entre –1 y 1, que representan valores a lo largo de algún intervalo de tiempo. Un valor 0, significa que la temperatura o presión es estable. Un valor 0.1 denota un crecimiento modesto; un valor de –0.3 denota valores en rápido descenso. Un valor cercano a –1 o 1 sugiere un cataclismo ambiental, que dispararía una alarma. En la figura 3.4, se visualiza una tendencia creciente de la temperatura, al obtener una pendiente positiva entre las muestras del sensor valor bajo y valor alto.

Sin embargo la tendencia (figura 3.5) no es común a todos los sensores, excepto en los de temperatura y presión en donde tiene sentido. De esta manera la tendencia se puede sacar como un factor común a la clase sensor y englobarse en una superclase común cuya responsabilidad y operación será la de informar de la tendencia del sensor.

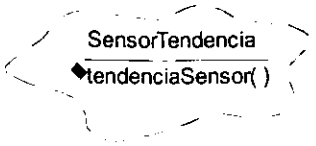


Figura 3.5 Sensor tendencia.

Para unificar las abstracciones de sensores, se plantea la clase base Sensor (figura 3.6), la cual sirve como superclase para las clases derivada SensorCalibrado y SensorTendencia.

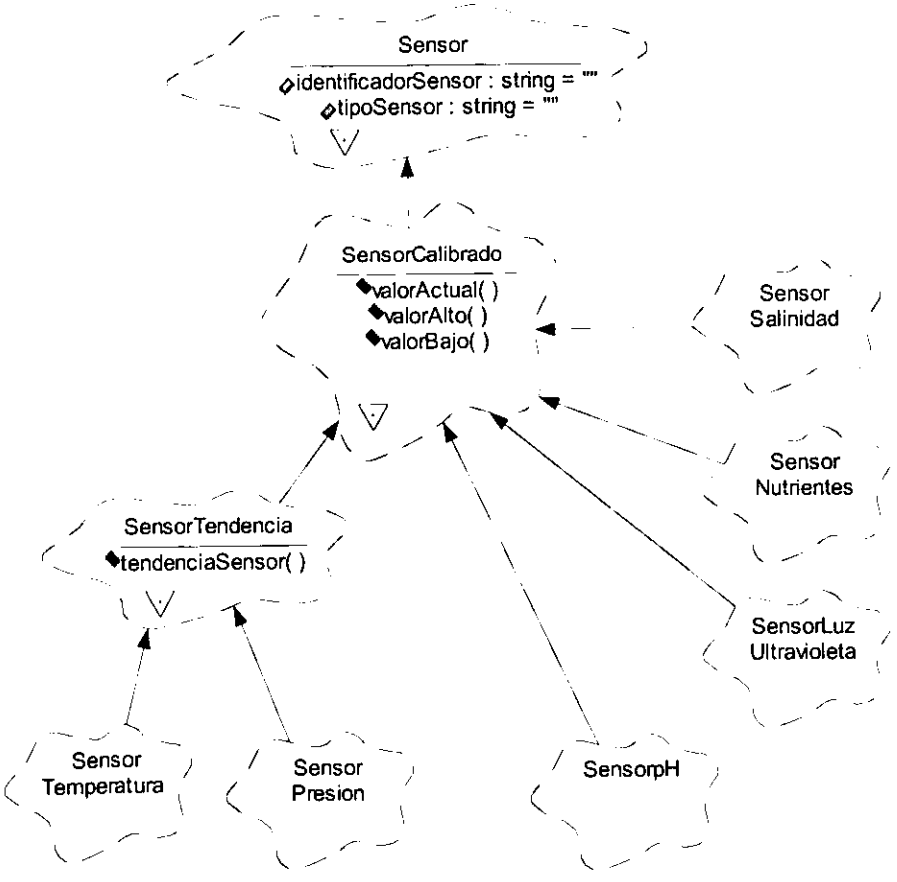


Figura 3.6 Diagrama jerárquico de la clase Sensor.

Además de los sensores, tenemos el muestreador (multiplexor), a través del cual se selecciona y enruta un canal analógico/digital para digitalizarlo y a continuación, en base a impulsos de reloj, previamente establecidos, se cambia a otro canal y repite la operación nuevamente. En una palabra, el muestreador (figura 3.7), nos permite ejecutar, el muestreo de los sensores y para ello debe contar con una función que establezca con que frecuencia se deben muestrear.

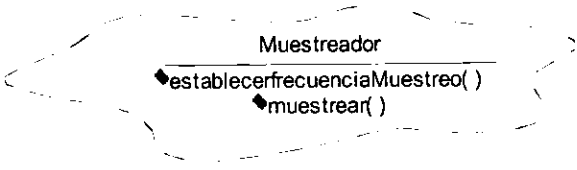


Figura 3.7 Clase Muestreador.

El temporizador (reloj), notifica periódicamente al muestreador cuando ha transcurrido un determinado intervalo de tiempo. El muestreador le envía un mensaje al temporizador "avísame cada cinco minutos". El temporizador, envía entonces mensajes de interrupción recurrentes para señalizar los intervalos. Un término habitual en programación que se utiliza en circunstancias de este tipo es la llamada Callback, en la que el cliente (Muestreador) suministra una función Callback al servidor (Temporizador) y cada cierto tiempo, el temporizador llama a esa función y lanza una interrupción, en base a la cual el Muestreador realiza una tarea determinada, como podría ser la de muestrear ciertos sensores.

La responsabilidad de temporizador (figura 3.8) será la de interceptar todos los eventos temporizados y despachar la función Callback correspondiente y la responsabilidad de la clase muestreador será la de registrar los datos de cada sensor monitorcado.

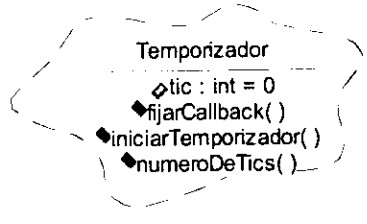


Figura 3.8 Clase Temporizador.

La etapa de análisis debe dar lugar a una declaración de lo que hace el sistema no como lo hace, sin embargo, cualquier afirmación intencionada sobre él <<como>>, es con el fin de exponer el comportamiento del sistema. También debe entender el analista que al hacerse abstracciones y discriminar cuales objetos considerar y cuales no, en cierta forma está tomando decisiones de diseño.

A continuación se tiene la clase Vista (figura 3.9), la cual nos permite gestionar y despachar la entrada del usuario, es también, la interfaz a través de la cual se ofrece la visualización de las medidas principales, como el mínimo y el máximo o la tendencia de los valores de algún sensor en particular. Esta clase en especial, por decisiones de diseño está relacionado con el ambiente de implantación, está pensada para derivarse del ambiente de programación Visual C++ (como será evidente en el capitulo de implantación).

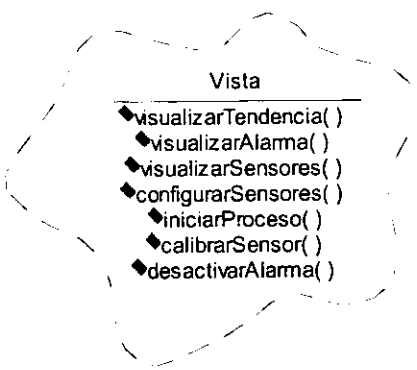


Figura 3.9 Clase Vista o ventana de interfaz de usuario.

Continuando con el análisis de identificar las clases y objetos, nos encontramos con otro elemento fundamental de nuestro sistema de adquisición de datos, a saber, el cultivo de moluscos. Aun cuando el requerimiento 1, habla específicamente, del cultivo de moluscos, vemos en general, la posibilidad de otros cultivos: peces, crustáceos, algas, etcétera. Esta situación nos conduce a pensar en objetos polimorfos, es decir que las operaciones que se puedan ejecutar en el cultivo de moluscos, también puedan ejecutarse en un cultivo de peces. En el modelo a objetos, si A se deriva de B vía herencia, entonces A es polimorfo con B.

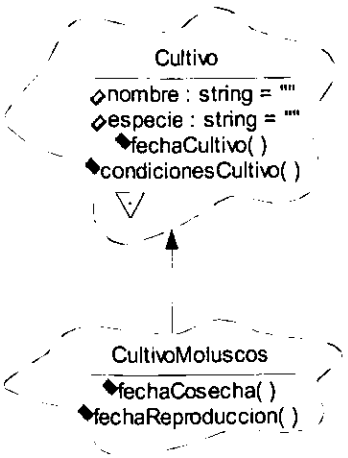


Figura 3.10. Relación de herencia de la clase cultivo.

Una forma de expresar las abstracciones y evitar redundancia, es utilizando una relación de herencia. La clase Cultivo (figura 3.10) es una clase abstracta que sirve de puente hacia sus subclases. **Cultivo** representa un grupo abstracto de todas las especies que pueden ser cultivadas.

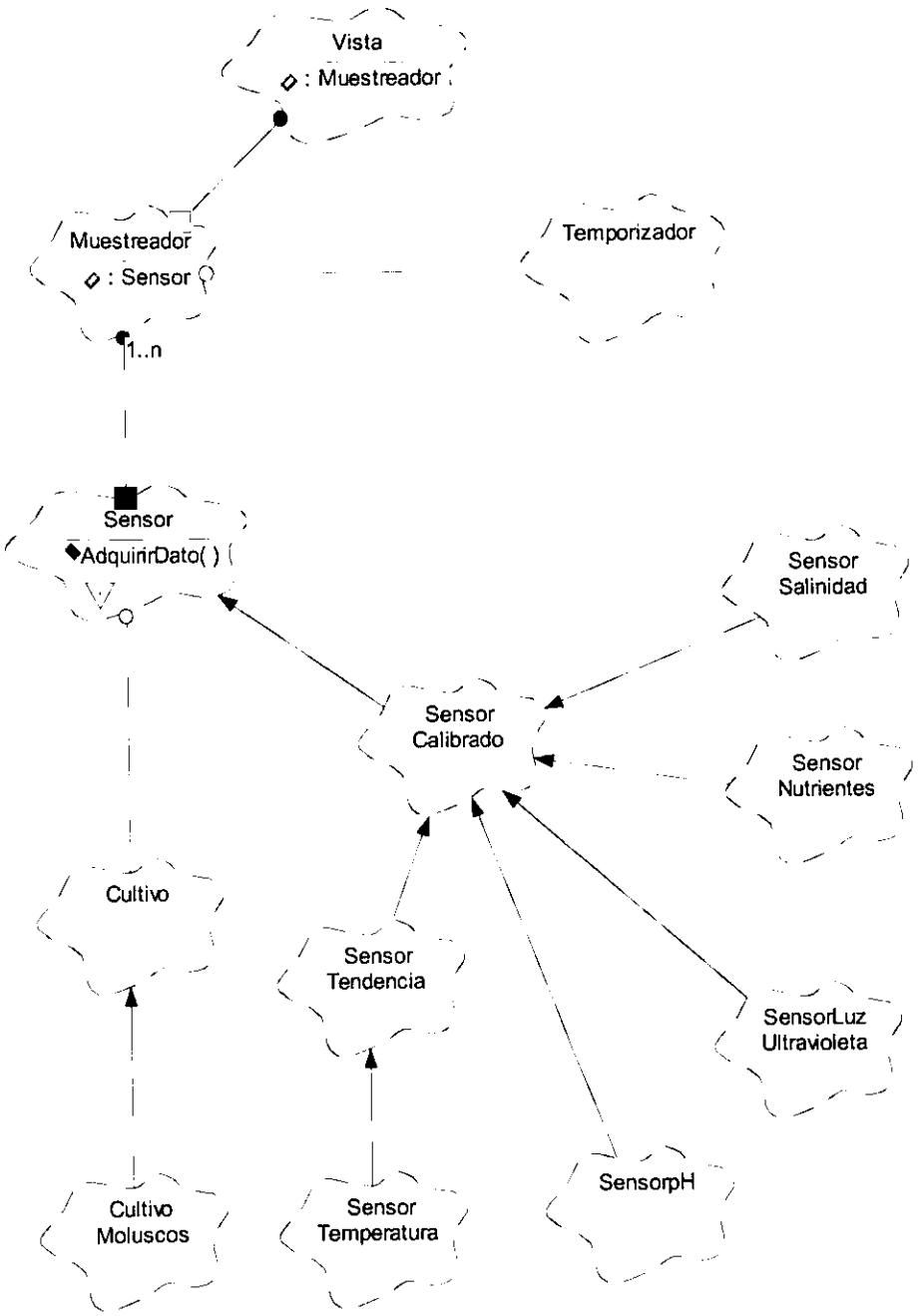
La relación de herencia pública también denominada "ES UN", además permite tratar en forma polimorfa a los objetos, es decir, el polimorfismo permite tratar a un grupo de objetos de diferentes tipos como si ellos pertenecieran a un mismo tipo.

Resumiendo, las anteriores consideraciones se pueden expresar de la siguiente manera:

- Algunos sensores son de temperatura otros de salinidad, otros de oxígeno, etcétera. Es decir cada sensor tiene asociada una semántica, dependiendo de su tipo, no obstante que todos valores muestreados son de tipo flotante.
- De las frecuencias de muestreo, emerge la clase Muestreador, a través de la cual los sensores son monitoreados cada minuto, otros cada cinco minutos, otros cada diez minutos y otros cada 24 horas. El usuario, establece estas frecuencias de muestreo de acuerdo a un plan de monitoreo previamente establecido para cada sensor.
- Los sensores adquieren un valor que corresponde a una medida monitoreada. Este valor es recibido por una instancia de la clase Muestreador.
- Los sensores están ubicados en los cultivos, a partir de estas posiciones, envían sus valores muestreados. Sin embargo, la clase Muestreador no distingue a los sensores por su posición, sino por su identificador, es decir, el identificador de sensor esta asociado a la posición que ocupa.
- Cada sensor se pueden calibrar.
- Los sensores de temperatura calculan su tendencia.

El diagrama de clases que representa estas relaciones aparece en la figura 3.11 (página siguiente).

Figura 3.11. Modelo estático. Muestreo del cultivo de moluscos.



Dos importantes conceptos se han empleado aquí. Primero, la clase Sensor es una clase base abstracta contenida en la clase Muestreador como una colección de sensores. Segunda, esta clase proporciona una interfaz, que la clase Muestreador usa para enviar sus mensajes a sus clases derivadas.

Lo anterior hace uso de un importante patrón de diseño llamado BRIDGE[1]. Un BRIDGE existe cuando una clase cliente (tal como Muestreador) contiene una clase abstracta servidor (tal como Sensor). El cliente puede invocar al servidor sin depender de las implantaciones del servidor; de esta manera, podemos usar al cliente en muchos contextos diferentes.

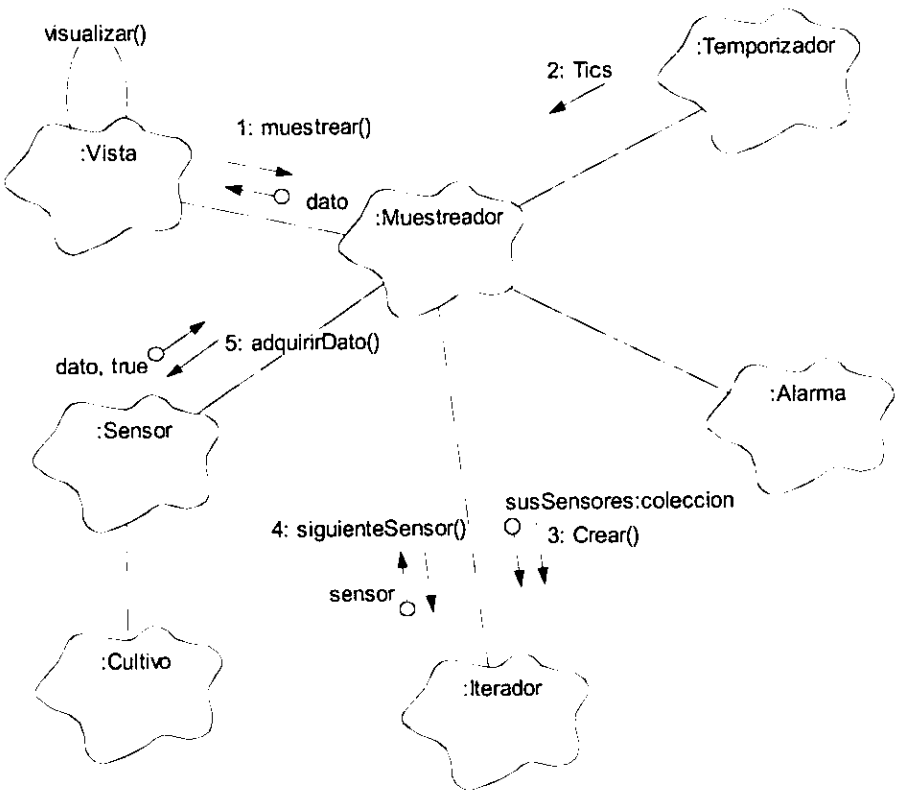


Figura 3.12 Modelo dinámico del muestreo del cultivo de moluscos.

La figura 3.12 muestra el modelo dinámico para el **Muestreo**. Note que el mensaje 1. muestrear() es producido por el objeto :Vista, no es un objeto real, éste representa todos los usuarios de la clase Muestreador. A continuación el objeto Muestreador, inicia el proceso

[1] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison Wesley, 1995).

de muestreo. Este diagrama ilustra el rol de **Sensor** e **Iterador**. Sin pretender demasiado formalismo con los nombres de los mensajes. Se ha usado el mensaje **Crear** para describir la construcción del iterador y el mensaje **siguienteSensor** para describir el acceso al objeto **Sensor** a través del iterador. Se ha implementado la presencia de un ciclo usando los números secuenciales 3, 4 y 5.

En la siguiente página se muestra mediante la figura 3.13 el diagrama del escenario que ilustra la colaboración de las abstracciones principales para el requerimiento 1. En dicha figura, el mensaje 1 inicia la colaboración requiriendo que **Muestreador** inicie el muestreo. El mensaje 2 envía recurrentemente los tic's de reloj. El mensaje 3, crea el iterador y pasa **susSensores** (la colección) a éste. El mensaje 4 inicia el ciclo; aquí se ve como el muestreador extrae un sensor desde el iterador. El sensor es entonces requerido en el mensaje 5 por su valor muestreado y se repite el ciclo, para otro sensor.

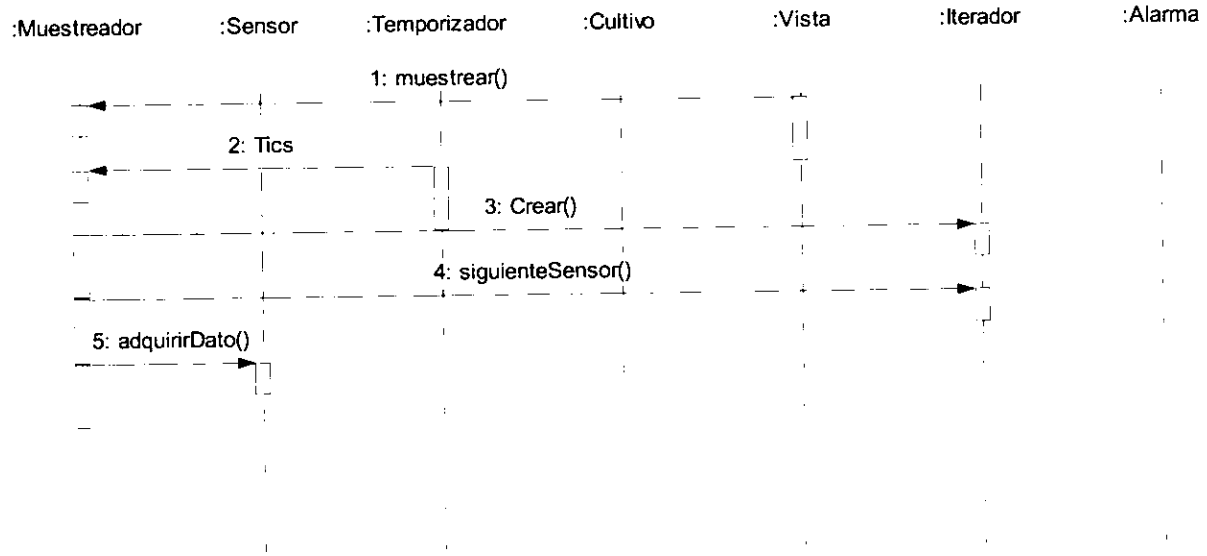


Figura 3.13. Escenario de monitoreo de las mediciones básicas del cultivo de moluscos.

Reflexión

Con esto concluye nuestro análisis preliminar del requerimiento 1. Las oscilaciones hacia atrás y hacia delante son típicas entre el modelo estático y el dinámico. Cada avance revela deficiencias en los anteriores. Cada cambio incrementa tanto la aproximación como la complejidad del modelo. El proceso termina cuando el modelo parece razonablemente consistente.

Requerimiento 2. Muestreador, Temporizador, Sensor, Cultivo.

5. Monitorear los cultivos de peces para determinar:
   - Las concentraciones de amonio disueltas en el agua.
   - Las concentraciones de nitratos disueltas en el agua.
   - La temperatura del agua.

Trasladando este requerimiento en su correspondiente caso de uso.

Caso de uso 2. Requisitos de muestreo las concentraciones de amonio, nitratos y temperatura del agua en los cultivos de peces. Cuando un usuario inicia el periodo de muestreo, el sistema monitorea los sensores recurrentemente cada cierto tiempo. Los datos adquiridos del cultivo de peces se muestran en pantalla. Si la cantidad de un sensor en particular es menor que un valor establecido, se dispara la alarma correspondiente.

El segundo requerimiento es muy similar al primero. En lugar de abordar el análisis desde cero, aprovecharemos esta oportunidad para analizar el impacto de incorporarlo, en la arquitectura obtenida para el monitoreo del cultivo de moluscos.

Una de las ventajas del uso del modelo de objetos es que promueve la reutilización no solo del software, sino de diseños enteros, conduciendo a la creación de marcos de desarrollo de aplicaciones reutilizables.

En nuestro caso, se ha obtenido un diseño para el monitoreo de los cultivos de moluscos y nos encontramos con un nuevo requerimiento de monitoreo para un cultivo de peces, tenemos ante sí, el reto de probar la estabilidad de nuestro diseño y verificar su flexibilidad ante este nuevo requerimiento y analizar su comportamiento evolutivo en el tiempo.

El segundo requerimiento dice que el sistema debe monitorear los cultivos de peces. En la figura 3.10 (página 34), se tomo una decisión de diseño de crear una clase abstracta de Cultivo, para compartir código y unificar las relaciones entre los diversos cultivos marinos. Esto es justamente un ejemplo más de cómo el diseño y el análisis se entremezclan.

Notamos que el cultivo de Peces también responde al mismo tipo común de operaciones que la clase Moluscos, estas operaciones están englobadas en una superclase llamada Cultivo (figura 3.14) unidas mediante relaciones de herencia. Es decir, los métodos implantados en las subclases de cultivo se pueden invocar desde la superclase común. Esta característica se llama polimorfismo y nos permite explotar una vez mas el patrón de diseño BRIDGE, que se muestra en el diagrama siguiente.

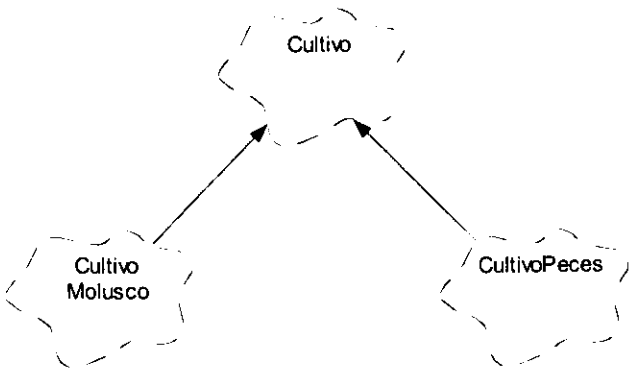


Figura 3.14. Jerarquía de clases de cultivo.

Booch considera que el polimorfismo y la abstracción son las características más potentes de los lenguajes de programación orientados a objetos y es un concepto central en el diseño orientado a objetos.

En cuanto a las abstracciones de los sensores que llevaran a cabo las mediciones para el cultivo de peces, se ve que se puede explotar nuevamente el patrón de diseño BRIDGE, para la clase Sensor. En la siguiente página se muestra el diagrama de la jerarquía de las clase Sensor, con los sensores para el cultivo de peces ya incorporados.

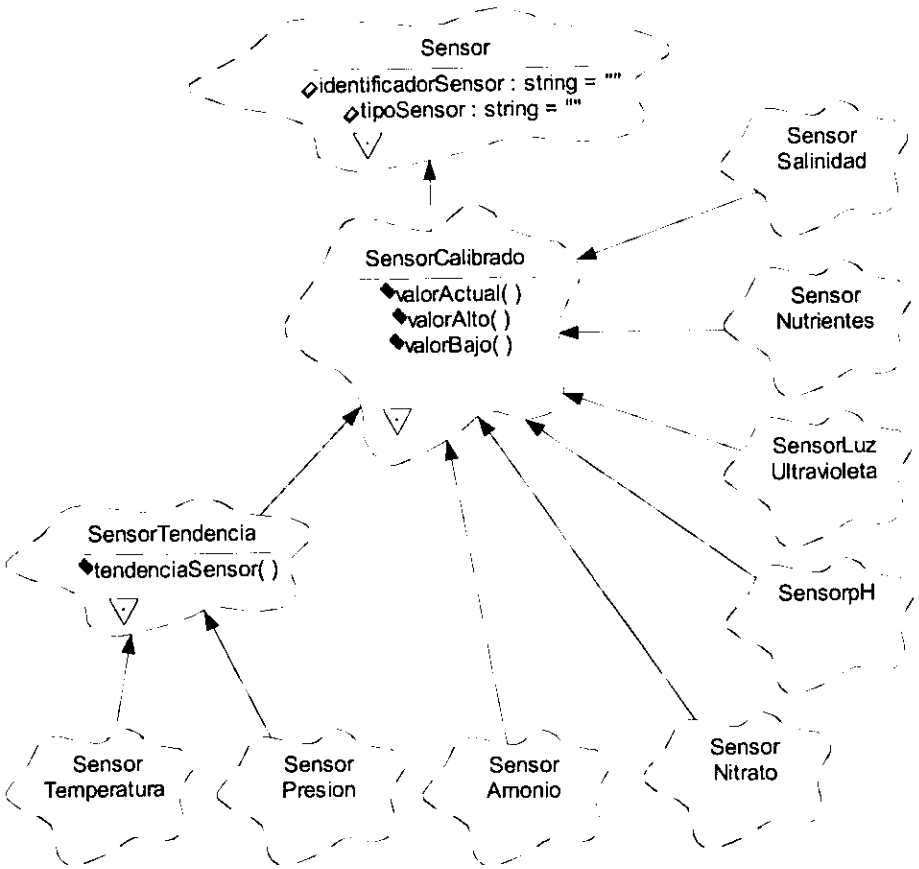


Figura 3.15. Jerarquía de la clase Sensor para el monitoreo de los cultivos de peces y moluscos.

Después de este análisis preliminar de la clase Cultivo y la clase Sensor, ¿Cuál es el impacto de incorporar en nuestro diseño inicial el monitoreo para los cultivos de peces? La figura 3.15, muestra el diseño lógico para el monitoreo de los cultivos de moluscos en el que se ha adaptado el nuevo requerimiento y se ha mantenido nuestro desarrollo sano.

Se debe monitorear otro tipo de cultivo y usar otros tipos de sensores, pero que tienen respectivamente interfaces comunes con los cultivos y sensores analizados para el requerimiento 1. Modelamos lo anterior como se muestra en la figura 3.16 (página siguiente).

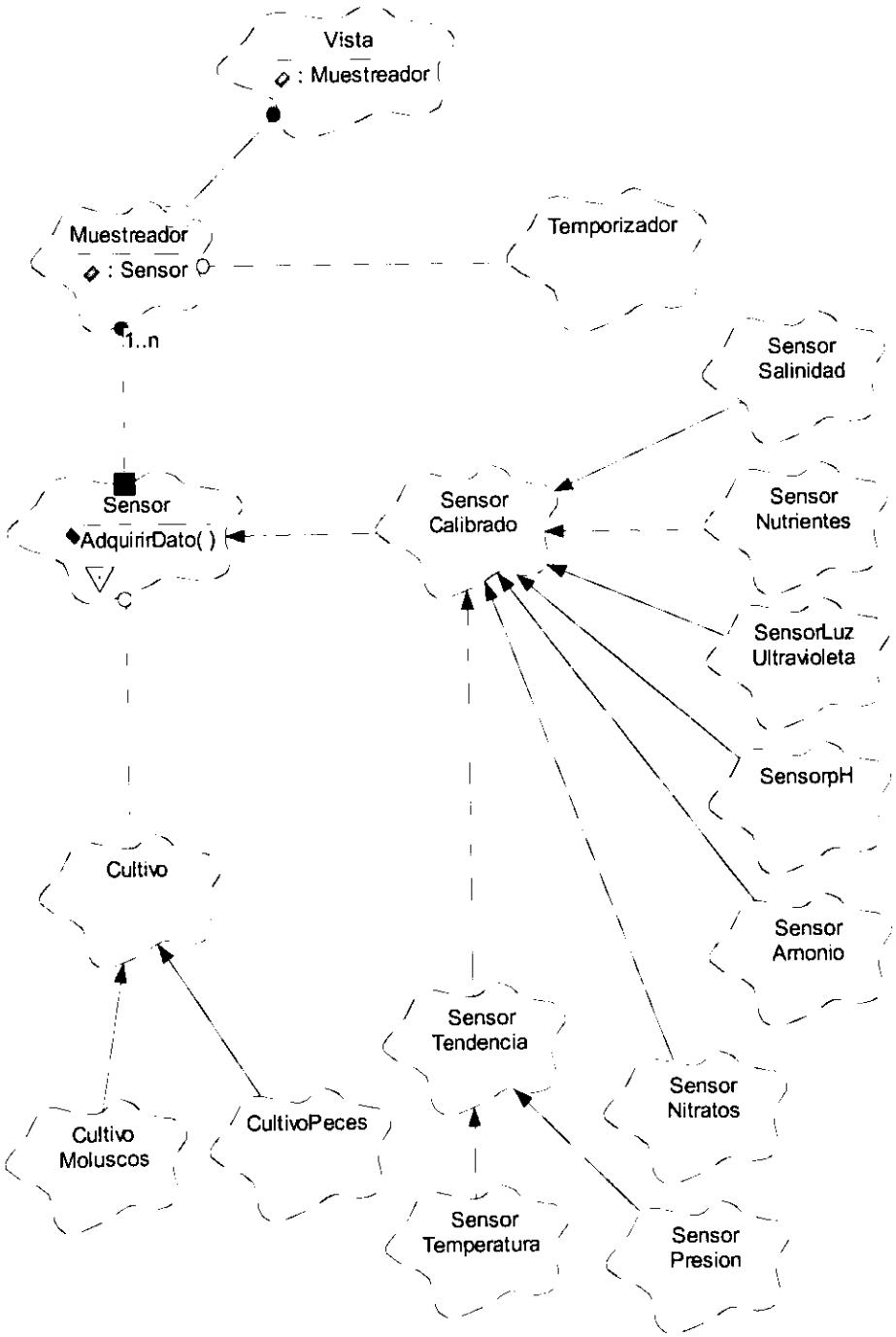


Figura 3.16 Modelo estático. Monitoreo de sensores.

Este modelo es interesante porque el usuario tiene mas flexibilidad. Ahora cuando el usuario desee monitorear otro tipo de cultivo, puede reutilizar el código y el diseño. Se omiten los correspondientes diagramas dinámicos y de escenario para el requerimiento 2, en vista de su similitud con los del requerimiento 1 (ver figuras 3.12 y 3.13).

Reflexiones.

Durante el análisis de este requerimiento, no se tuvo que especular demasiado en el modelo estático. El comportamiento del sistema es el mismo, solo creció en tamaño y la complejidad fue fácilmente manejada. También se nota que el sistema ha evolucionado influenciado fuertemente por los requerimientos.

Requerimiento 3. Muestreador, Temporizador, Sensor, Ducto.

6. Monitorear las tuberías para determinar:
   - La presión del agua de la red de tuberías que abastecen los recipientes de los cultivos. Con ello se busca tener un mejor control sobre los cartuchos de filtrado. Una disminución en la presión, puede significar la necesidad de sustituir un cartucho de filtrado
   - Monitoreo de la presión del agua de la red de tuberías que abastecen los depósitos secundarios. Con ello se busca tener un mejor control sobre el rendimiento de las bombas. Una disminución de la presión, puede significar la necesidad de sustituir una bomba.

Trasladando este requerimiento en su correspondiente caso de uso.

Caso de uso 3. Requisitos de muestreo de la presión del agua en la red de tuberías.
Cuando un usuario inicia el periodo de muestreo, el sistema monitorea los sensores recurrentemente cada cierto tiempo. Los datos adquiridos de la red de tuberías se muestran en pantalla. Si la cantidad de un sensor en particular es menor que un valor establecido, se dispara la alarma correspondiente.

Puesto que la única manera de monitorear los sensores ubicados en las tuberías es a través del objeto Muestreador. Los sensores deben aparentemente conocer en que tubería están. El modelo es mostrado en la figura 3.17 (página siguiente).

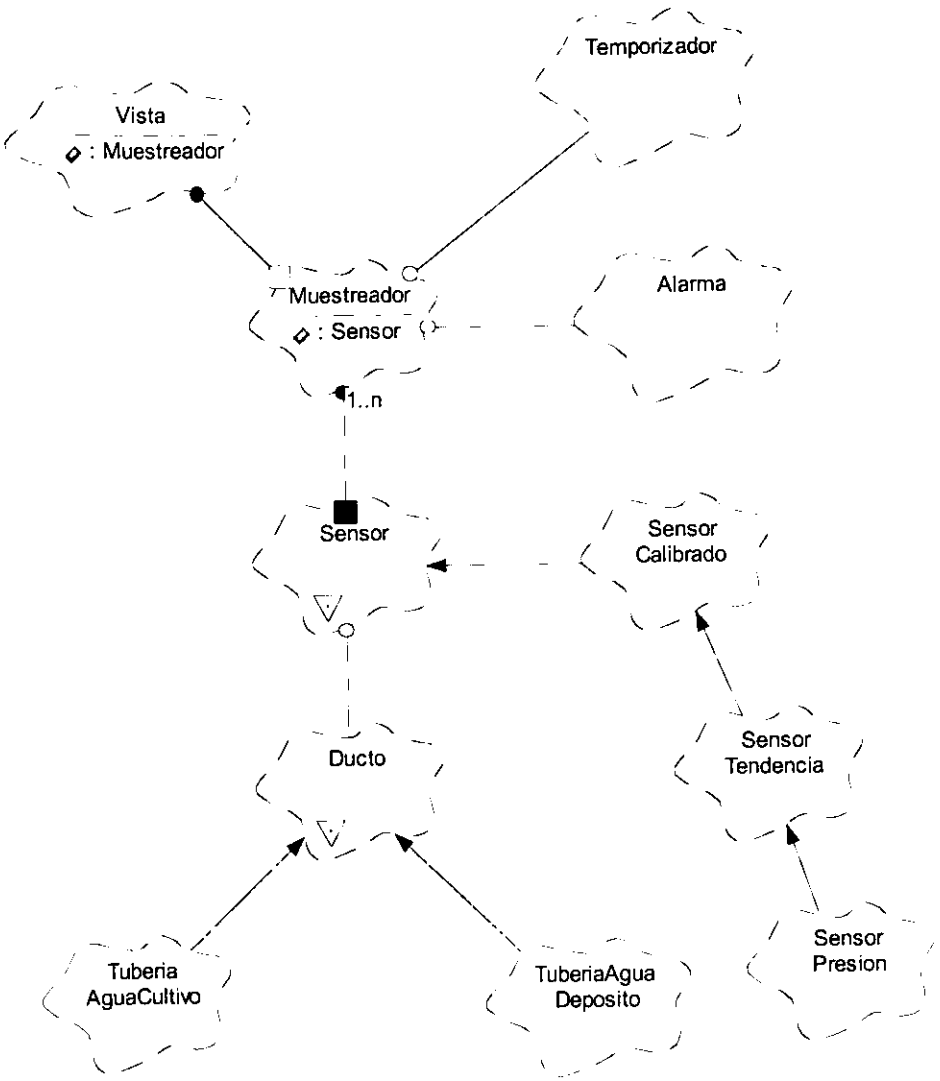


Figura 3.17. Monitoreo de tuberías.

¿Qué justifica que se haya derivado **TuberiaAguaCultivo** y **TuberiaAguaDeposito** de la clase base **Ducto**? No se ha documentado ninguna clase abstracta que pueda residir en **Ducto**. Es importante recordar el principio de diseño BRIDGE. La justificación principal para la herencia es que proporciona una interfaz abstracta para clases diferentes, de esta manera puede usarse cualquier función a través de una clase base. La figura 3.17 muestra que **Ducto** es un medio para usarse como una interfaz abstracta que puede aceptar métodos de muestreo de alguna clase de objeto sensor. No conocemos que sensor es, pero conocemos de alguna manera el mecanismo relativo al manejo del sensor para enviar un

método de muestreo a un objeto que tiene una interfaz **Sensor**. Si mas adelante se presentara un nuevo requerimiento para un ducto de aire o de cualquier otro fluido, estaría lista la interfaz.

Hay únicamente un escenario de comportamiento (figura 3.18), y éste es tan simple como los vistos para los anteriores requerimientos.

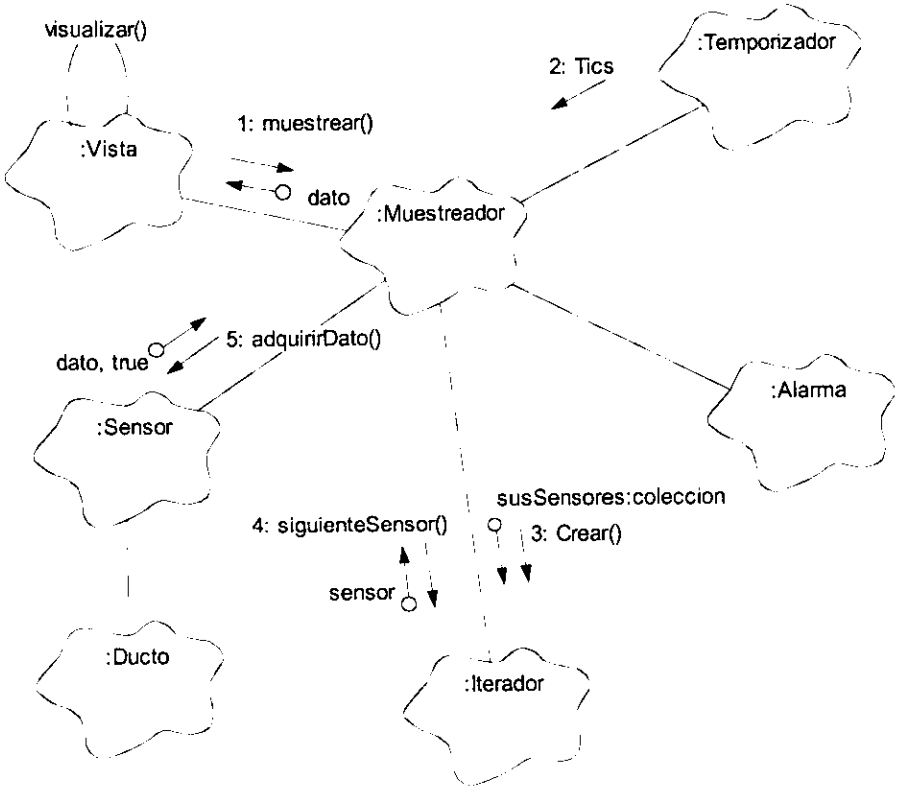


Figura 3.18. Escenario dinámico, para el muestreo de tuberias.

En este escenario algún cliente dentro de Vista envía un mensaje al Muestreador para que inicie el proceso de muestreo. A continuación el Muestreador primero crea un iterador con una colección de sensores y luego inicia un ciclo enviando un mensaje a cada sensor para que envíe los datos de las medidas de presión que se están detectando. Estas medidas son generadas por una clase derivada de Sensor. Un **SensorPresionTuberiaAguaCultivo** puede detectar la presión del agua de una tubería que abastece a los cultivos y un **SensorPresionTuberiaAguaDeposito** puede detectar la presión del agua de una tubería que abastece a los depósitos de almacenamiento.

Reflexión.

Se ha elegido describir los requerimientos mediante casos de uso, ésta pudo haber sido una mala decisión, sin embargo, se ha descubierto que los requerimientos son extremadamente similares. Lo que quizá no hubiera ocurrido de haber usado otra técnica como una lista de nombres y verbos. El hecho de que los casos de uso pudieran haber sido mejores en esta situación particular, no significa que siempre sea lo mejor.

Capítulo 4.

Implantación del sistema de adquisición de datos.

4.1 Implantación.

La implantación del sistema se ha construido con una funcionalidad mínima que monitoriza sólo un sensor, pero que incluye, la colaboración de prácticamente todas las abstracciones clave: las clases CSensor, CMuestreador, CTemporizador (timer de Windows), Vista (CView de Visual C++) y Alarma (MessageBeep de Visual C++).

Los puntos funcionales del sistema descansan en una arquitectura de marcos de tiempo. La clase CMuestreador toma el tiempo y lo divide en varios marcos de longitud fija, los cuales se dividieran posteriormente en submarcos, cada uno con algún comportamiento, de esta manera se tiene un control de los eventos con respecto al momento en que ocurren. Así, tendremos submarcos para adquirir un dato, otro para calcular la tendencia y otro para visualizar, etcétera. Un objeto de la clase CTemporizador, tiene la responsabilidad de lanzar una interrupción cada cierto tiempo, de acuerdo a la función callback proporcionada por el cliente, en este caso un objeto de la clase CMuestreador. De esta forma se instrumenta la separación de intereses entre estas clases. Los objetos de la clase CMuestreador, se liberan del conocimiento sobre como interceptar eventos temporizados y los objetos de la clase CTemporizador sobre que hacer cuando sucede un evento de este tipo.

El Temporizador (timer) de Windows es un dispositivo de entrada que notifica periódicamente a una aplicación cuándo ha transcurrido un determinado intervalo de tiempo. Por ejemplo, el objeto de CMuestreador, le indica a Windows el intervalo. Windows envía entonces al Objeto de CMuestreador mensajes WM_TIMER recurrentes para señalizar los intervalos.

El comportamiento de las clases CMuestreador y CSensor y sus clases derivadas se expresa a través de sus interfaces que a continuación se describen:

```
// Muestreador.h

#ifndef _INSIDE_VISUAL_CPP_MUESTREADOR
#define _INSIDE_VISUAL_CPP_MUESTREADOR

class CSensorTemperatura;

//---------------------------------------------------
// Nombre
// CMuestreador
//
```

```
// Descripción
// El muestreador consulta por turno a cada sensor, pero se salta
// intencionadamente algunos de ellos con el fin de muestrearlos
// a una frecuencia menor. Consultando a cada sensor, en lugar de
// permitir que cada sensor actúe como un hilo de control.
// La ejecución del sistema se hace mas predecible, porque el
// muestreador puede controlar el flujo de los eventos.
//

class CMuestreador : public CObject
{

        DECLARE_SERIAL(CMuestreador)

public:
// Atributos
        int m_nTic;
        CSensorTemperatura* m_nSensorTemperatura;
public:
// Constructores
        CMuestreador();
// Funciones miembro
//      void fijarFrecMuestreo(const CSensorTemperatura* szSensor, int szTic);
        void fijarFrecMuestreo(int szTic);

        void muestrear(int szTic);
// Destructor
        virtual ~CMuestreador();
// Persistencia
        virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
        void Dump(CDumpContext& dc) const;
#endif // _DEBUG
};

#endif // INSIDE_VISUAL_CPP_MUESTREADOR
```

Cuadro 4.1 Clase muestreador.

El código del protocolo de la clase muestreador se presenta en el cuadro 4.1. Durante el diseño se estableció que contiene la colección de sensores por valor, es decir, el tiempo de vida de la colección depende del tiempo de vida de la instancia muestreador. Note que m_nSensorTemperatura es un atributo de la clase Muestreador.

Las responsabilidades para un objeto de la clase muestreador son: establecer su configuración (fijarFrecuenciaMuestreo), iniciar el muestreo (muestrear) y serializarse (Serialize).

La interfaz de usuario para la configuración del objeto muestreador y para el muestreo de sensores se presentan en las figuras 4.1 y 4.3 respectivamente.

El proceso de guardar y restablecer objetos se llama serialización. La idea es que los objetos pueden ser persistentes y se pueden guardar en disco cuando termine un programa y restablecerse cuando inicie. En este caso se aprovecha la biblioteca de clases de Visual C++, motivo por el cual la clase Muestreador se deriva de Cobject, para que se pueda serializar directamente. La figura 4.1 muestra la opción para guardar la configuración de un objeto muestreador.

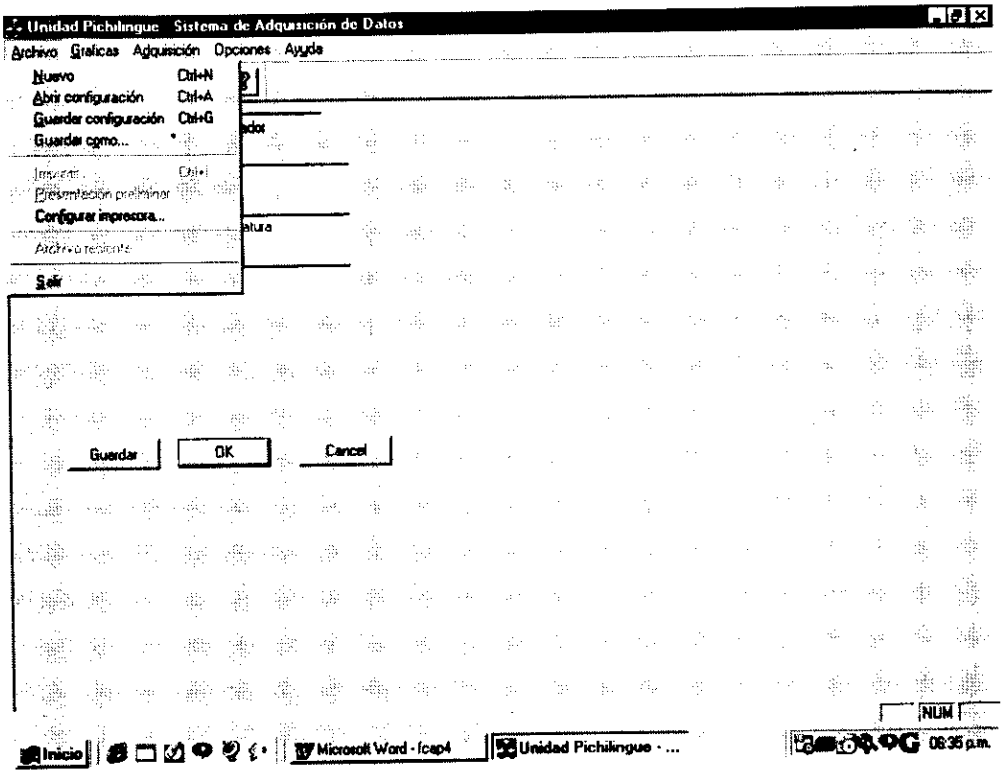


Figura 4.1 Serialización de la configuración de muestrador.

```
// Sensor.h

#ifndef _INSIDE_VISUAL_CPP_CSENSOR
#define _INSIDE_VISUAL_CPP_CSENSOR


//---------------------------------------------------
// Nombre
// CSensor
//
// Descripción
// Esta clase es una interface abstracta para los usuarios
// de la interface de sensor. Para mantener el ambiente
// adecuado en los cultivos marinos, hay que controlar diversos
// factores, como temperatura, concentraciones de nutrientes,
// etcétera. Para cada medida existe un sensor diferente.
//

class CSensor // : public CObject
{
protected:
// Atributos
        CString m_nIdSensor;
public:
// Constructores
        CSensor();
        CSensor(const CString& szIdSensor);
// Destructor
        virtual ~CSensor();
// Metodos publicos para obtener y establecer elementos de datos
        CString obtIdSensor() const;
        void estIdSensor(CString szIdSensor);
// Funciones virtuales puras, se definen en una clase derivada
        virtual float valorActual()=0;
        virtual float valorBruto()=0;
};

#endif // _INSIDE_VISUAL_CPP_SENSOR
```

Tabla 4.2 Protocolo de la clase sensor.

La tabla 4.2 presenta el código del protocolo de la clase base abstracta sensor, de la cual como se estableció en el diseño se deriva la jerarquía de clases sensores. La idea es que en esta jerarquía se ubiquen adecuadamente las operaciones polimorficas con el fin de extraer los aspectos más comunes. El constructor de esta clase proporciona a sus instancias un conocimiento de una identificación y un tipo de sensor. Es una clase abstracta porque incluye funciones miembro virtuales puras.

```
//////////////////////
// SensorCalibrado.h
//////////////////////

#ifndef _INSIDE_VISUAL_CPP_CSENSORCALIBRADO
#define _INSIDE_VISUAL_CPP_CSENSORCALIBRADO

#include "Sensor.h"


//---------------------------------------------------
// Nombre
// CSensorCalibrado
//
// Descripción
// Esta clase se construye sobre la clase base CSensor.
// El usuario calibra al sensor para ajustarlo a valores
// reales conocidos y los números intermedios proporcionados
// por el sensor se traducen al tipo de medida por
// interpolación líneal simple.
//

class CSensorCalibrado : public CSensor
{
protected:
// Miembros datos
        float m_nValorMinimoMedida;
        float m_nValorMaximoMedida;
        float m_nValorMinimoSensor;
        float m_nValorMaximoSensor;
        float m_nValorBruto;
        float m_nValorActual;
public:
// Constructores
        CSensorCalibrado();
        CSensorCalibrado(const CString& szIdSensor);
// Destructor
        virtual ~CSensorCalibrado();
// Metodos publicos para obtener y establecer elementos de datos
        float obtValorBruto() const;
        void estValorBruto(float szValorBruto);
        float obtValorActual() const;
        void estValorActual(float szValorActual);
        float obtValorMinimoSensor() const;
        void estValorMinimoSensor(float szValorMinimoSensor);
        float obtValorMinimoMedida() const;
        void estValorMinimoMedida(float szValorMinimoMedida);
        float obtValorMaximoSensor() const;
        void estValorMaximoSensor(float szValorMaximoSensor);
        float obtValorMaximoMedida() const;
        void estValorMaximoMedida(float szValorMaximoMedida);
```

```
// Funciones virtuales, se obtiene un valor del fenómeno
// físico calibrado.
// El valor bruto se refiere al valor del sensor, sin
// realizar aun la conversión al fenómeno físico. se
// espera que una clase derivada lo defina.

        virtual float valorActual();
        virtual float valorBruto() = 0;
};

#endif // _INSIDE_VISUAL_CPP_SENSORCALIBRADO
```

Tabla 4.3 Protocolo de la clase sensor calibrado.

La tabla 4.3 presenta el código de la clase sensor calibrado, que como se nota, se construye sobre la clase base sensor. En esta clase se establecen los valores mínimo y máximo permitidos para una medida física, como por ejemplo la temperatura y los valores mínimo y máximo para los valores adquiridos por el sensor. Por ejemplo el valor analógico (voltaje) convertido a su forma digital de un sensor, debe corresponder a una medida física que se esté monitoreando. La operación miembro que se introduce valorActual(), sirve para este propósito.

```
////////////////////
// Sensortendencia.h
////////////////////
#ifndef _INSIDE_VISUAL_CPP_SENSORTENDENCIA
#define _INSIDE_VISUAL_CPP_SENSORTENDENCIA

#include "SensorCalibrado.h"


//---------------------------------------------------------
// Nombre
// CSensorTendencia
//
// Descripcion
// Esta clase hereda de CSensorCalibrado y añade una nueva
// responsabilidad: calcular la tendencia de una medida.
//

class CSensorTendencia : public CSensorCalibrado
{
protected:
// Miembro dato
        float m_nTendencia;
public:
// Constructores
        CSensorTendencia();
```

```
        CSensorTendencia(const CString& szIdSensor);
// Destructor
        virtual ~CSensorTendencia();
// Métodos públicos para obtener y establecer elementos de datos
        float obtTendencia() const;
        void estTendencia(float sztendencia);
// Modificadores
        float calcularTendencia();
};

#endif // _INSIDE_VISUAL_CPP_SENSORTENDENCIA
```

Tabla 4.4 Protocolo para la clase sensor tendencia.

El código del protocolo de la clase sensor tendencia presentado en la tabla 4.4, introduce una nueva función miembro, que calcula la tendencia de las medidas físicas adquiridas, en base a un valor previamente establecido. Por ejemplo si 23° centígrados es la temperatura ideal para un cultivo, cualquier medida adquirida por encima o por debajo indicara una tendencia positiva o negativa, respectivamente.

```
////////////////////
// SensorTemperatura.h
////////////////////
#ifndef _INSIDE_VISUAL_CPP_SENSORTEMPERATURA
#define _INSIDE_VISUAL_CPP_SENSORTEMPERATURA

#include "SensorTendencia.h"

//---------------------------------------------------
// Nombre
// CSensorTemperatura
//
// Descripción
// Esta es una subclase concreta, de la cual se crearan instancias
// para los sensores de temperatura distribuidos en los
// lugares estratégicos del sistema acuícola.
//

class CSensorTemperatura: public CSensorTendencia, public CObject
{
        DECLARE_SERIAL(CSensorTemperatura)
        CString m_nTipoSensor;
        int m_nPuerto;
        bool m_nAlarma;

        float m_nValorTemperatura;
public:
// Constructores
```

```
        CSensorTemperatura();
        CSensorTemperatura(const CString& szIdSensor, const CString& szTipoSensor,
        float szValorMinimoMedida,
        float szValorMaximoMedida,
        float szValorMinimoSensor,
        float szValorMaximoSensor,
        float szTendencia);
// Constructor de copia
        CSensorTemperatura (const CSensorTemperatura& s);
// Asignacion
        const CSensorTemperatura& operator = (const CSensorTemperatura& s);
// Operadores de comparación
        BOOL operator ==(const CSensorTemperatura& s) const;
        BOOL operator !=(const CSensorTemperatura& s) const;
// Destructor
        virtual ~CSensorTemperatura();
// Métodos públicos para obtener y establecer elementos de datos
        CString obtTipoSensor() const;
        void estTipoSensor(CString szTipoSensor);
        int ObtPuerto() const;
        void EstPuerto(int szPuerto);
        bool ObtAlarma() const;
        void EstAlarma(bool szAlarma);
// Modificadores
        virtual float valorBruto();
        virtual float temperaturaActual();
        virtual bool sonarAlarma();
// Persistencia
        void Serialize(CArchive& ar);

#ifdef _DEBUG
        void Dump(CDumpContext& dc) const;
#endif // _DEBUG
};

#endif // _INSIDE_VISUAL_CPP_SENSORTEMPERATURA
```

Tabla 4.5 Protocolo de la clase sensor temperatura.

Finalmente alcanzamos la clase concreta de la cual se obtendrán instancias que lleven a cabo la adquisición de datos. En este caso se tiene un ejemplo de herencia múltiple, por un lado se hereda el comportamiento de la clase sensor tendencia y por el otro de la clase CObject de la biblioteca de Visual C++. La razón de implantar de esta manera la clase sensor temperatura obedece simplemente a aprovechar la serialización proporcionada por la clase Cobject.

Esta clase incluye las operaciones valorBruto() con la cual obtiene el valor digital de un sensor a través de un puerto de entrada (puerto paralelo) y la operación temperaturaActual() a través de la cual se convierte el valor digital obtenido por el sensor a su correspondiente medida de temperatura.

4.2 Mecanismo de Interfaz de usuario.

La implantación que completa la funcionalidad de las clases CTimer, CView, CDocument, que pertenecen a la MFC (Microsoft Fundamental Class) de Visual C++ no requieren trabajo de diseño. Combinan las decisiones que se tomaron durante el análisis se derivan las siguientes interfaces:

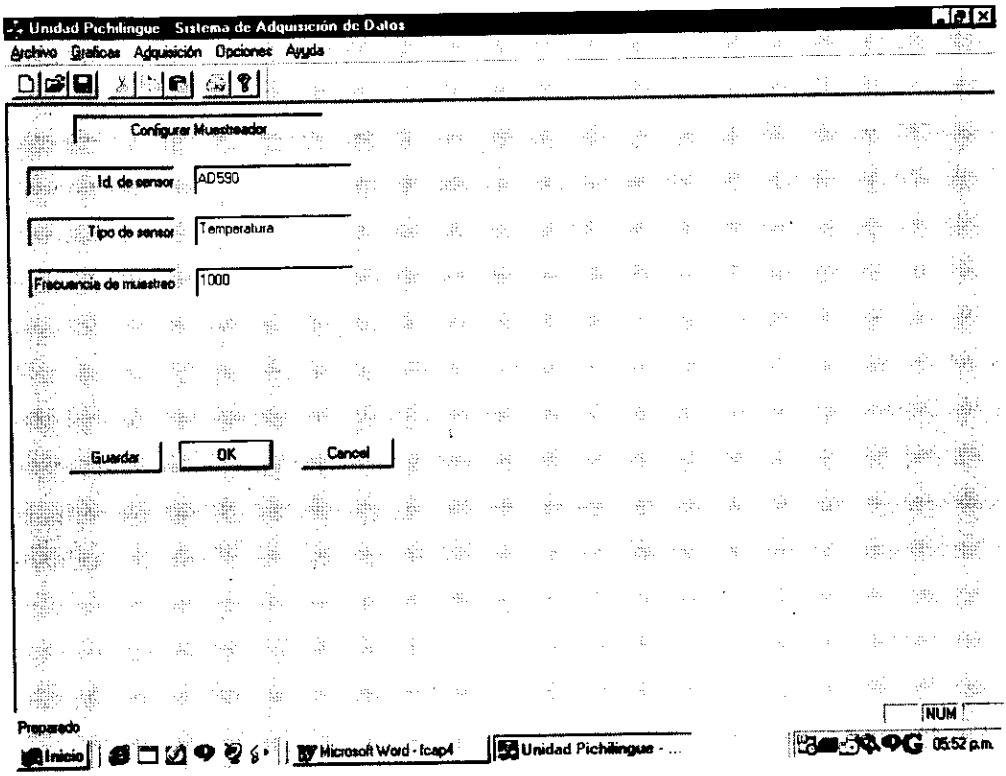


Figura 4.2 Configuración de muestreador.

El sistema inicia mostrando la interfaz de operación mediante un menú que incluye las opciones de Archivo (Nuevo, Guardar datos, Guardar estadística), Graficar (Graficar tendencia), Adquisición (Muestrear sensores, Análisis de sensores), Opciones (Configurar Muestreador y Configurar sensor) y Ayuda. La opción que se presenta en la Figura 4.2 es la correspondiente a la configuración del Muestreador. La configuración para esta corrida de prueba indica el identificador del sensor AD590, su tipo temperatura y su frecuencia de muestreo cada minuto (1000 milisegundos).

La opción configuración de sensores, para establecer el estado del sensor que iniciara el muestreo se muestra en la Figura 4.3. En esta interfaz se establecen los valores iniciales para calibrar el sensor y una medida de temperatura que sirve como punto de equilibrio para el cultivo que se esta monitoreando.

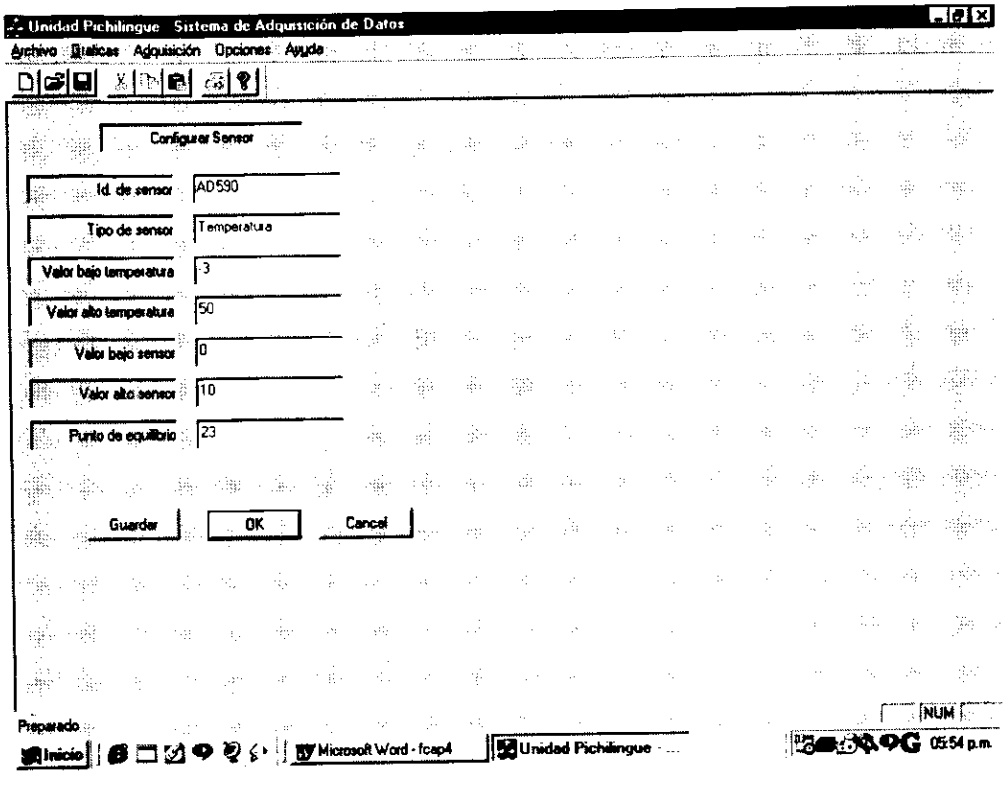


Figura 4.3 Configuración de Sensor.

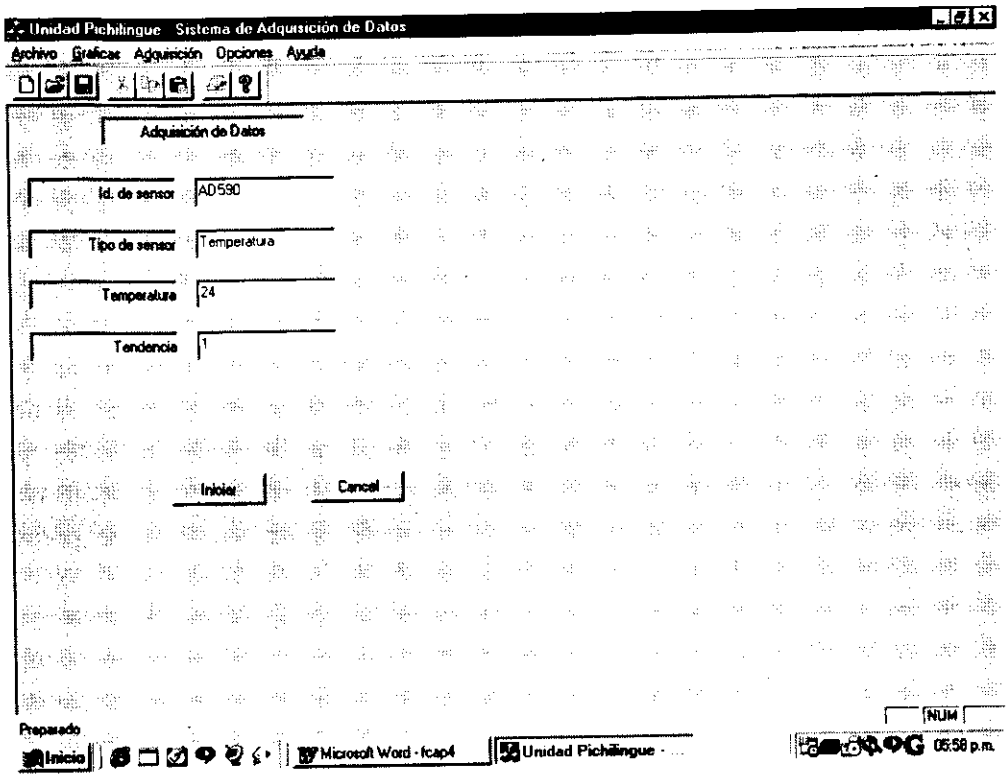


Figura 4.4 Proceso de muestreo

La figura 4.4, nos muestra el proceso de muestreo, al ejecutarse la opción Muestrear sensores del menú Adquisición. Se presenta una medida adquirida y la tendencia positiva.

4.3 Mantenimiento.

La implantación completa de este sistema de adquisición de datos solo abarca un puñado de clases, sin embargo, es una pieza de software, que aprovecha muy bien algunas de las ventajas promovidas por la Metodología Orientada a Objetos, a saber, un análisis directo, que emerge fácilmente de los requerimientos del usuario, la flexibilidad y la reutilización de las clases abstraídas del sistema.

Por ejemplo, que ocurriría, si fuera necesario sensar alguna otra medida importante del sistema, en este caso no es necesario alterar la arquitectura. Es propio de los sistemas orientados a objetos bien construidos, que al hacer un cambio como éste no se haga pedazos la arquitectura existente, sino que se reutilicen y aumente sus mecanismos existentes.

Conclusión

Durante el diseño del sistema de adquisición de datos, no hubo razón para forzar el uso del paradigma orientado a objetos en ninguno de sus aspectos y, se logro que las dependencias de código fuente fueran reducidas a las abstracciones. Por otro lado, los diagramas y los textos del diseño son mucho más breves que el de los requerimientos y describen el comportamiento del sistema con más rigor y precisión, creando un diseño reusable y mantenible.

La notación de Booch, permitió enfocar nuestro esfuerzo de solución en una vista abstracta de los requerimientos del problema, en lugar de una vista del mundo real de los detalles necesarios de su sustentación.

Al implantar el sistema se ha logrado integrar completamente el diseño de clases. Es muy importante resaltar que el tiempo empleado para el diseño del sistema (solución a los requerimientos del usuario) son mucho mayores que el tiempo empleado en la implantación. Y aun se abatirían mas los tiempos de implantación, creando controles Active X, con la biblioteca de clases de adquisición de datos.

El resultado más importante de este trabajo es que el análisis y diseño orientado a objetos ha probado ser un método idóneo en la solución problemas de adquisición de datos, sobre todo al aprovechar las principales técnicas del método de Booch.

Apéndice. Método de Booch.

El método de Booch, es una guía practica para el diseño orientado a objetos de sistemas de software. La notación de Booch, permite documentar el diseño orientado a objetos de un sistema y su instrumentación en C++.

En éste apéndice, se presentan los conceptos fundamentales de la notación de Booch.

Diagrama de clases.

Un diagrama de clases se utiliza para mostrar la existencia de clases y sus relaciones en la visión lógica de un sistema. Las clases se representan por una nube punteada en las que se incluye el nombre de la clase, sus atributos, sus métodos y posiblemente alguna marca.

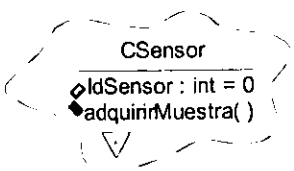


Figura A1. Una nube punteada representa una clase.

Las relaciones de contención se representan como se muestra en la figura A2.

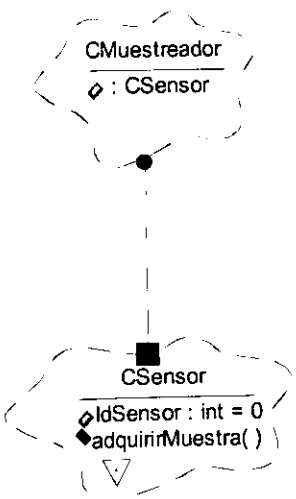


Figura A2. Relación de contención por valor.

El circulo negro indica la clase contenedora. Cada instancia de CMuestreador contiene una instancia de CSensor. La clase contenida está asignada en una variable miembro public (a

menos que se indique private). El cuadrito negro indica contención por valor, significa que el tiempo de vida del objeto contenido está controlado por el objeto contenedor. En código C++

```
class CMuestreador :
{
public:
        CSensor SensorTemperaturaCultivoMoluscos;
};
o también
class CMuestreador :
{
public:
        CSensor* SensorTemperaturaCultivoMoluscos;
};
```

Las relaciones por referencia, se representan como se muestra en la figura A3.

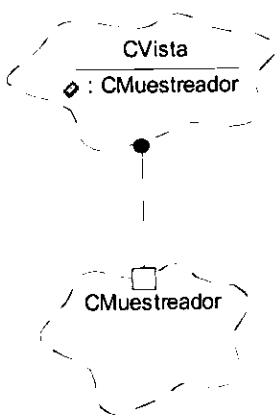


Figura A3. Relación de contención por referencia.

El cuadrito abierto, significa contención por referencia. La contención por referencia no implica propiedad. El objeto contenedor no es responsable del tiempo de vida del objeto contenido. Los tiempos de vida de los dos objetos son independientes, con la restricción, de que el objeto contenido debe durar mientras dure el objeto contenedor. En código C++

```
class CVista
{
public:
        CMuestreador(const CMuestreador& szMuestreador)
        :m_nMuestreador(szMuestreador) {}
private:
        const CMuestreador& m_nMuestreador;
};
```

La relación de herencia, se representa como en la figura A4.

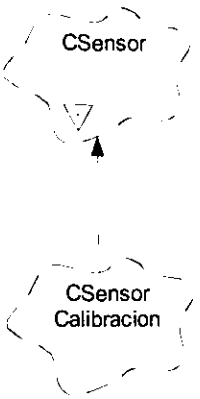


Figura A4. Relación de herencia entre clases.

La relación de herencia se representa por una flecha que apunta a la clase base. En la Figura A4, CSensorCalibracion heredan de la clase abstracta CSensor. En código C++

class CSensorCalibracion: public CSensor {...};

¿Qué significa que CSensorCalibracion hereda de CSensor? La respuesta inmediata es que el CSensorCalibración también requiere un identificador de sensor y lo puede compartir con otras subclases de CSensor. Una respuesta no tan inmediata es que se ha tomado una decisión estratégica para aislar el conocimiento de CSensorCalibración del resto del programa. De ésta manera se puede usar CSensorCalibracion en cualquier contexto en que se espera un CSensor.

La contención con cardinalidad se representa como en la figura A6.

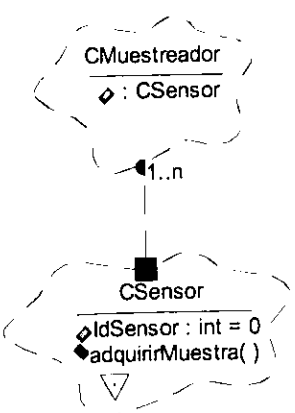


Figura A5. Relación de contención con cardinalidad.

El diagrama de la figura A5, muestra que instancias de CMuestreador contiene una o más instancias de la clase CSensor. La cardinalidad ésta representada por "1..n". En código C++

```
class CMuestreador
{
public:
        enum {numeroDeSensores=100;}
private:
        CSensor* sensores[numeroDeSensores];
};
```

Otra posibilidad de contención con cardinalidad, usando plantillas (template), ilustrada con el diagrama A6.

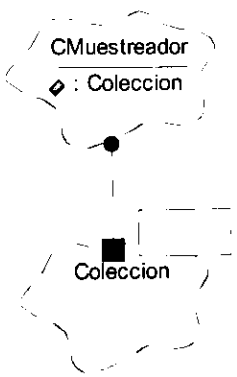


Diagrama A6. Clases instanciadas.

Esta relación es menos restrictiva que la anterior. Instancias de CMuestreador contiene una Colección de objetos CSensor. Usando una clase como CtypedPtrList de la librería de clases estándar de Visual C++, como una simple clase contenedora que puede contener un numero no especificado de elementos. En código C++

```
class CMuestreador
{
private:
        typedef CTypedPtrList<CSensor*> listaCSensor;
}
```

La relación de contención polimorfa se muestra en el diagrama A7.

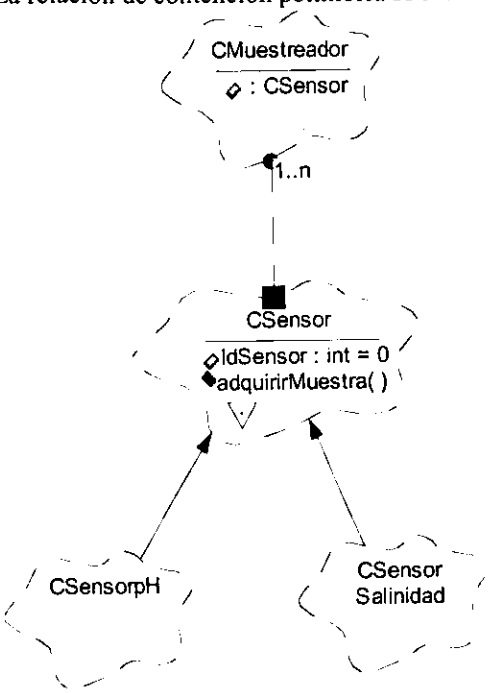


Figura A7. Contención polimorfica.

El diagrama A7, muestra que un CMuesteador puede contener dos clases de CSensor; sin embargo, CMuestreador no conoce la diferencia entre estas. Este manipula ambas clases de CSensor a través de la interfaz genérica CSensor. En código C++, el polimorfismo se maneja usando apuntadores o referencias a la clase base. Así, si queremos manipular objetos CSensorpH y CSensorSalinidad debemos hacerlo a través de la interfaz genérica de CSensor, debemos hacerlo a través de apuntadores o referencias a la clase CSensor. Esto es asi, porque listaCSensor de la clase Cmuestreador es declarada como typedef CTypedPtrList<CSensor*> listaCSensor. La colección contiene apuntadores a la clase base CSensor, por lo tanto los objetos CSensor pueden manipularse polimorficamente.

Diagrama de objetos.

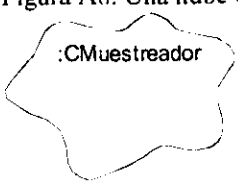
Booch, define un objeto como algo que tiene ciertas propiedades invariantes y puede manipularse, cambiando su estado. Cuando un objeto es manipulado a través de su interfaz, éste debe hacer algo útil. Este debe expresar su comportamiento. Así, un objeto es algo que tiene estado, comportamiento e interfaz.

En el diseño orientado a objetos, los objetos son manipulados enviando mensajes a estos. En C++, se envían mensajes a los objetos, llamando a sus funciones miembro. Por ejemplo, dado un objeto sensorTemperatura, podemos enviar un mensaje a éste diciéndole que adquiera una muestra.

```
class CSensor
{
        public:
        void adquirirMuestra();
        ...
};

CSensor sensorTemperatura;
SensorTemperatura.adquirirMuestra();
```

Figura A8. Una nube cerrada, representa un objeto.



El diagrama de la figura A9, muestra un ejemplo de un modelo dinámico.

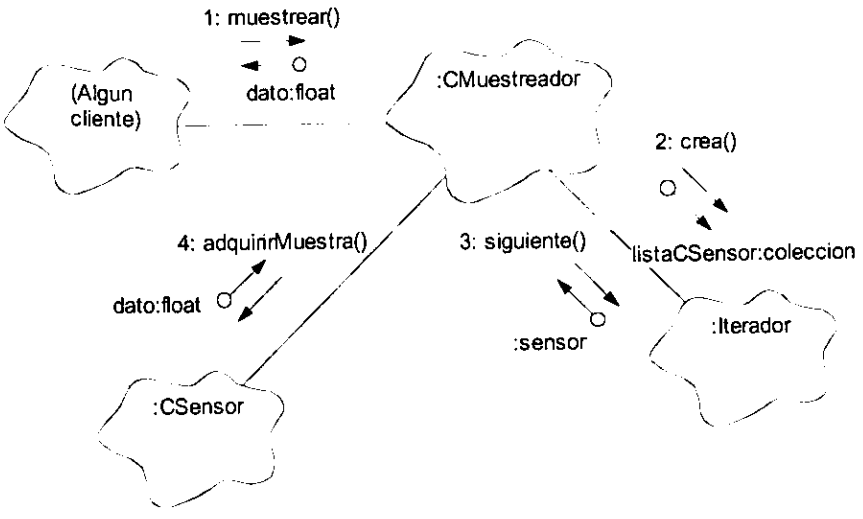


Figura A9. Iteración en un diagrama objeto.

El diagrama A8 muestra que pasa cuando el mensaje 1:muestrear() es enviado a un objeto CMuestreador por algún cliente usuario de la clase CMuestreador. CMuestreador debe iterar a través de todos los objetos CSensor contenidos y obtener sus medidas adquiridas. Los números que preceden a cada mensaje muestran el orden en que se envían los mensajes. Los números 3 y 4 corresponden a un ciclo, mientras la colección aun tenga sensores.

Bibliografía

James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen. Object-Oriented Modeling and Design. Prentice Hall, 1991

Grady Booch. Análisis y Diseño Orientado a Objetos con aplicaciones. Segunda edición. Addison Wesley/Díaz Santos, 1994.

Miguel Katrib Mora. Programación Orientada a Objetos. INFOSYS. 1994.

Robert Cecil Martín. Designing Object-Oriented C++ Applications, Using the Booch Method. Prentice Hall, 1995.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. 1995.

Howard Hutchings. Interfacing with C. BUTTERWORTH/HEINEMANN. 1995.

David J. Kruglinski. Inside Visual C++. Fourth Edition. Microsoft Press. 1997.

David McCombs. Detecting the Word, R&D Books. 1999.

Mike Bunnell and Mitch Bunnell. Real-time Data Acquisition. Dr. Dobb's Journal, june-1989.

Tom Nolan. Real-time Data Acquisition using DMA. Dr. Dobb's Journal, january-1990.

James F. Farley and Peter D. Varhol. A Visual Approach to Data Acquisition. Dr. Dobb's Journal, may-1993.

Brian Hook and Dennis Shuman. Digital i/o with the PC. Dr. Dobb's Journal, april-1994.

Guadalupe Ibarguengoitia González, Hanna Oktaba. El Rol del Analista para Desarrollos con Tecnología Orientada a Objetos. Soluciones Avanzadas, abril 1998.

National Instrument. The Measurement and Automation, Catalog 2000.

Citas.

Parnas	Cuando se afirma que un sistema se describe con una función continua, quiere decirse que no puede contener sorpresas ocultas. Pequeños cambios en las entradas siempre producirán cambios consecuentemente pequeños en la salida.
Peter	Cuanto más complejo sea el sistema, más abierto está al derrumbamiento total.
Simon	El hecho de que muchos sistemas complejos tengan una estructura jerárquica y casi descomponible es un factor importante de ayuda que nos capacita para comprender, describir e incluso "ver" estos sistemas y sus partes.
Rechtin	Todos los sistemas tienen subsistemas y todos los sistemas son parte de sistemas mayores... El valor añadido por un sistema debe proceder de las relaciones entre las partes, no de las partes por sí mismas.
Simon	Simon llama a los sistemas jerárquicos "descomponibles", porque pueden dividirse en partes identificables; los llama "casi descomponibles", porque sus partes no son completamente independientes.
Simon	Los sistemas complejos evolucionarán a partir de sistemas simples con mucha mayor rapidez si hay formas intermedias estables que si no las hay.
Gall	Se encontrará invariablemente que un sistema complejo que funciona ha evolucionado de un sistema simple que funcionaba... Un sistema complejo diseñado desde cero nunca funciona y no puede parcharse para conseguir que lo haga. Hay que volver a empezar, partiendo de un sistema simple que funcione.
Miller	El máximo número de bloques de información que un individuo puede comprender de forma simultánea es del orden de siete más o menos dos.
Simon	La velocidad de proceso es un factor limitador: La mente necesita alrededor de cinco segundos para aceptar un nuevo bloque de información .
Dijkstra	La técnica de dominar la complejidad se conoce desde tiempos remotos: "divide et impera" (divide y vencerás) .

Parnas — La descomposición inteligente ataca directamente a la complejidad inherente al software forzando una división del espacio de estados del sistema.

Stein — La programación estructurada parece derrumbarse cuando las aplicaciones superan alrededor de 100.000 líneas de código.

Shaw — (los humanos) hemos desarrollado una técnica excepcionalmente potente para enfrentarnos a la complejidad. Realizamos abstracciones. Incapaces de dominar en su totalidad un objeto complejo, decidimos ignorar sus detalles no esenciales, tratando en su lugar con el modelo generalizado e idealizado del objeto.

Petroski — La concepción de un diseño para una nueva estructura puede involucrar un salto de la imaginación y una síntesis de experiencia y conocimiento tan grandes como el que requiere cualquier artista para plasmar su obra en una tela o en un papel. Y una vez que el ingeniero ha articulado ese diseño como artista, debe analizarlo como científico aplicando el método científico con tanto rigor como cualquier científico lo haría.

Mostow — El propósito del diseño es construir un sistema que:

- Satisface determinada (quizás informal) especificación funcional.
- Se ajusta a las limitaciones impuestas por el medio de destino.
- Respeta requisitos implícitos sobre rendimiento y utilización de recursos.
- Satisface criterios de diseño implícitos o explícitos sobre la forma de artefacto.
- Satisface restricciones sobre el propio proceso de diseño, tales como su longitud o coste, o las herramientas disponibles para realizar el diseño.

Stroustrup — El propósito del diseño es crear una estructura interna (a veces llamada también arquitectura) clara y relativamente simple... Un diseño es el producto final del proceso de diseño.

Rentsch — Mi impresión es que la programación orientada a objetos va ha ser en los ochenta lo que fue la programación estructurada en los sesenta. Todo el mundo va a estar a favor de ella. Todos los fabricantes van a promocionar sus productos afirmando que la soportan. Todos los administradores hablarán bien de ella. Todos los programadores la practicarán (de forma diferente). Y nadie va a saber exactamente qué es.

Stefik — Define los objetos como entidades que combinan las propiedades de los procedimientos y los datos en el sentido de que realizan computaciones y conservan el estado local.

Stroustrup — Si el término "orientado a objetos", significa algo, debe significar un lenguaje que tiene mecanismos que soportan bien el estilo de programación orientada a objetos... un lenguaje soporta bien un estilo de programación si proporciona capacidades que hacen conveniente utilizar tal estilo. Un lenguaje no soporta una técnica si exige un esfuerzo o habilidad excepcionales escribir tales

programas; en ese caso, el lenguaje se limita a permitir a los programadores el uso de esas técnicas.

Cardelli — Cierto lenguaje es orientado a objetos si y sólo si satisface los siguientes requisitos:

- Soporta objetos que son abstracciones de datos con un interfaz de operaciones con nombre y estado local oculto
- Los objetos tienen un tipo asociado [clase]
- Los tipos [clase] pueden heredar atributos de los supertipos [superclases].

Jenkins — La mayoría de los programadores trabajan en un lenguaje y utilizan sólo un estilo de programación. Programan bajo un paradigma apoyado por el lenguaje que usan. Frecuentemente, no se les han mostrado vías alternativas para pensar sobre un problema, y por tanto tienen dificultades para apreciar las ventajas de elegir un estilo más apropiado para el problema que tienen entre manos.

Bobrow — Un estilo de programación es una forma de organizar programas sobre las bases de algún modelo conceptual de programación y un lenguaje apropiado para que resulten claros los programas escritos en ese estilo.

Dahl — La abstracción surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias.

Shaw — Una abstracción es como una descripción simplificada o especificación de un sistema que enfatiza algunos de los detalles o propiedades del mismo mientras suprime otros. Una buena abstracción es aquella que enfatiza detalles significativos al lector o usuario y suprime detalles que son, al menos por el momento, irrelevantes o causa de distracción.

Berzins — Un concepto merece el calificativo de abstracción sólo si se puede describir, comprender y analizar independientemente del mecanismo que vaya a utilizarse eventualmente para realizarlo.

Meyer — Se puede caracterizar el comportamiento de un objeto considerando los servicios que presta a otros objetos, así como las operaciones que puede realizar sobre otros objetos. Este punto de vista obliga a concentrarse en la visión exterior del objeto y, nos lleva al "el modelo contractual".

Ingalls — Ninguna parte de un sistema complejo debe depender de los detalles internos de otras partes.

Liskov — Para que la abstracción funcione, la implementación debe ser encapsulada.

Stroustrup — La ocultación es para prevenir accidentes, no para prevenir el fraude.

Myers — El acto de fragmentar un programa en componentes individuales puede reducir

su complejidad en algún grado... Aunque la fragmentación de programas es útil por ésta razón, una justificación más poderosa para ésta fragmentación es que crea una serie de fronteras bien definidas y documentadas dentro del programa. Estas fronteras o interfaces, tienen un incalculable valor para a la comprensión del programa.

Liskov — La modularización consiste en dividir un programa en módulos que puedan compilarse separadamente, pero que tienen conexiones con otros módulos.

Parnas — Las conexiones entre módulos son las suposiciones que cada módulo hace acerca de todos los demás.

Zelkowitz — Puesto que no puede conocerse la solución cuando comienza la etapa de diseño, la descomposición en módulos más pequeños puede resultar bastante difícil. Para aplicaciones más antiguas (como la escritura de compiladores), éste proceso puede llegar a estandarizarse, pero para otras nuevas (como sistemas de defensa o control de naves espaciales) puede ser bastante complicado.

Britton and Parnas — El objetivo de fondo de la descomposición en módulos es la reducción del coste del software al permitir que los módulos se diseñen y revisen independientemente... La estructura de cada módulo debería ser lo bastante simple coma para ser comprendida en su totalidad; debería ser posible cambiar la implantación de los módulos sin saber nada de la implantación de los demás módulos y sin afectar el comportamiento de éstos y, la facilidad de realizar un cambio en el diseño debería guardar una relación razonable con la probabilidad de que ese cambio fuese necesario.

Parnas, Clements and Weiss — Los detalles de un sistema, que probablemente cambien de forma independiente, deberían ser secretos en módulos separados; las únicas suposiciones que deberían darse entre módulos son aquellas cuyo cambio se considera improbable. Toda estructura de datos es privada a algún módulo; a ella pueden acceder directamente uno a más programas del módulo. Cualquier otro programa que requiera información almacenada en los datos de un módulo debe obtenerla llamando a programas de éste.

Meyer — La forma más apropiada de definir sistemas de software prácticos es decir que ofrecen un cierto número de servicios. Normalmente la definición de estos sistemas como funciones simples es posible, pero eso produce respuestas bastante más artificiales... Los sistemas reales no tienen una parte superior.

Cox — Sin herencia, cada clase sería una unidad independiente, desarrollada partiendo de cero. Las distintas clases no guardarían relación entre sí, puesto que el desarrollador de cada clase proporcionaría métodos según le viniese en gana. Toda consistencia entre clases es el resultado de una disciplina por parte de los programadores. La herencia posibilita la definición de nuevo software de la misma forma en que se presenta un concepto a un recién llegado, comparándolo con algo que ya le resulte familiar.

Danforth and Tomlinson	La abstracción de datos intenta proporcionar una barrera opaca tras de la cual se ocultan los métodos y el estado; la herencia requiere abrir ésta interfaz en cierto grado y puede permitir el acceso a los métodos y al estado sin abstracción.
Liskov	La subclase podría acceder a una variable de instancia de su superclase, o referenciar directamente a superclases de su superclase.
Zilles	Un tipo es una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades.
Tesler	Existen varios beneficios importantes que se derivan del uso de lenguajes con tipos estrictos: • Sin la comprobación de tipos, un programa puede "estallar" de forma misteriosa en ejecución en la mayoría de los lenguajes. • En la mayoría de los sistemas, el ciclo editar-compilar-depurar es tan tedioso que la detección temprana de errores es indispensable. • La declaración de tipos ayuda a documentar los programas. • La mayoría de los compiladores pueden generar un código más eficiente si se han declarado los tipos.
Lim and Johnson	El diseño de características para la concurrencia en lenguajes de POO no es muy diferente de hacerlo en otros tipos de lenguajes –la concurrencia es ortogonal a la POO en los niveles más bajos de abstracción. Se trate de POO o no, continúan existiendo todos los problemas tradicionales en programación concurrente.
Lim and Johnson	A los niveles más altos de abstracción, la POO puede aliviar el problema de la concurrencia para la mayoría de los programadores mediante la ocultación de la misma dentro de abstracciones reutilizables.
Strostrup	No siempre está claro cómo aprovechar mejor un lenguaje como C++. Se han logrado mejoras significativas de forma consistente en la productividad y la calidad del código utilizado C++ como un "C mejorado" con una pizca de abstracción de datos añadida donde era claramente útil. Sin embargo, se han obtenido mejoras distintas y apreciablemente mayores aprovechando las jerarquías de clase en el proceso de diseño. Esto se llama muchas veces diseño orientado a objetos, y aquí es donde se han encontrado los mayores beneficios del uso de C++.
Robson	Muchas personas que no tienen ni idea de cómo funciona un computador encuentran bastante natural la idea de los sistemas orientados a objetos.
Smith and Tockey	Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema.
Adams	Un papel es una máscara que se pone un objeto.

Wirfs-Brock — Las responsabilidades de un objeto de forma que incluyen dos elementos clave: el conocimiento que un elemento mantiene y las acciones que puede llevar a cabo. Las responsabilidades están encaminadas a transmitir un sentido del propósito de un objeto y de su lugar en el sistema. Las responsabilidades de un objeto son todos los servicios que proporciona para todos los contratos que soporta.

Ingalls — En lugar de un procesador triturador de bits que golpea y saquea estructuras de datos, tenemos un universo de objetos bien educados que cortésmente solicitan a los demás que lleven a cabo sus diversos deseos.

Meyer and Snyder — La programación es en gran medida un asunto de "contratos": las diversas funciones de un problema mayor se descomponen en problemas más pequeños mediante subcontratas a diferentes elementos del diseño.

Cardelli and Wegner — Los lenguajes convencionales con tipos, como Pascal, se basan en la idea de que las funciones y los procedimientos, y, por tanto, los operandos, tiene un único tipo. Tales lenguajes se dicen que son monomórficos, en el sentido de que todo valor y variable puede interpretarse que tiene un tipo y sólo uno. Los lenguajes de programación monomórficos pueden contrastarse con los lenguajes polimórficos en los que algunos valores y variables pueden tener más de un tipo.

Deutsch — El polimorfismo es más útil cuando existen muchas clases con los mismos protocolos.

Deutsch — El polimorfismo no se necesita alrededor del 85% del tiempo, así que el paso de mensajes puede reducirse a menudo a simples llamadas a procedimientos.

Snyder — Se puede ver la herencia como una decisión privada del diseñador para "reusar", código porque es útil hacerlo; debería ser posible cambiar con facilidad tal decisión. Alternativamente, se puede ver la herencia como la realización de una declaración pública de que los objetos de la clase hija obedecen la semántica de la clase padre, de modo que la clase hija simplemente especializa o refina la clase padre.

Vlissedes and Linton — Fuerza frecuentemente al programador a derivar de una sola de entre dos clases igualmente atractivas. Esto limita la aplicabilidad de las clases predefinidas, haciendo muchas veces necesario el duplicar código. Por ejemplo, no existe forma de derivar un gráfico que es a la vez un círculo y una imagen; hay que derivar de uno o del otro y reimplantar la funcionalidad de la clase que se excluyó.

Meyer — Uno de los problemas delicados planteados por la presencia de herencia múltiples es lo que sucede cuando una clase es un antecesor de otra por más de una vía. Si se permite herencia múltiple en un lenguaje, antes o después alguien escribirá una clase D con dos padres B y C, cada uno de los cuales tiene como padre a una clase A –o alguna otra situación en la que D herede dos (o más

veces) de A. Esta situación se llama herencia repetida y debe tratarse de forma correcta.

Hendler — Una clase aditiva es sintácticamente idéntica a una clase normal, pero su intención es distinta. El propósito de tal clase es únicamente... [añadir] funciones a otras [clases] .

Meyer — La herencia es un mecanismo más potente que la genericidad y que gran parte de los beneficios de la genericidad puede conseguirse mediante la herencia, pero no al revés.

Stroustrup — La parametrización de tipos permitirá parametrizar las funciones aritméticas respecto al tipo numérico básico, de forma que los programadores puedan (por fin) obtener un modo uniforme de tratar con enteros, números en punto flotante de precisión simple, de doble precisión, etcétera.

Ingalls — Un sistema debería construirse con un conjunto mínimo de partes inmutables; estas partes deberían ser tan generales como fuese posible; y todas las partes del sistema deberían conservarse en un marco de referencia uniforme.

Stevens, Myers and Constantine — La medida de la fuerza de la asociación establecida por una conexión entre un módulo y otro. El acoplamiento fuerte complica un sistema porque los módulos son más difíciles de comprender, cambiar o corregir por sí mismos si están muy interrelacionados con otros módulos. La complejidad puede reducirse diseñando sistemas con los acoplamientos más débiles posibles entre los módulos.

Meyer — Un buen diseñador sabe cómo encontrar el equilibrio apropiado entre subcontratar demasiado, lo que produce fragmentación, o demasiado poco, lo que produce módulos de tamaño inmanejable.

Sakkinen — Los métodos de una clase no deberían depender de ninguna manera de la estructura de ninguna clase, salvo de la estructura inmediata (de nivel superior) de su propia clase. Además cada método debería enviar mensajes sólo a objetos pertenecientes a un conjunto muy limitado de clases.

Meyer — La herencia es apropiada si toda instancia de B puede verse también como una instancia de A. La relación de cliente es apropiada cuando toda instancia de B simplemente posee uno a más atributos de A.

Wirth — La elección de la representación es frecuentemente algo bastante difícil, y no está determinado de manera unívoca por las posibilidades disponibles. Debe tomarse siempre a la luz de las operaciones que van a realizarse sobre los datos.

Shaw — El desarrollo de abstracciones individuales sigue frecuentemente un patrón común. Primero, los problemas se resuelven 'ad hoc'. A medida que se acumula la experiencia, se va viendo que algunas soluciones funcionan mejor que otras, y se transfiere informalmente una especie de folklore de persona a persona. Eventualmente, las soluciones útiles se comprenden de forma más sistemática, y

se codifican y analizan. Esto permite el desarrollo de modelos que admiten una implantación automática y de teorías que permiten generalizar la solución. Esto a su vez da lugar a un nivel de práctica más sofisticado y nos permite atacar problemas más difíciles, a los que con frecuencia se brinda un enfoque 'ad hoc', comenzando el ciclo de nuevo.

Wirfs-Brock — El conocimiento que un objeto tiene y las acciones que un objeto puede realizar. Las responsabilidades están encaminadas a comunicar una expresión del propósito de un objeto y su lugar en el sistema. Las responsabilidades de un objeto son todos los servicios que suministra para todos los contratos que soporta.

Arango — Se define el análisis de dominio como un intento de identificar los objetos, operaciones y relaciones que los expertos del dominio consideran importantes acerca del mismo.

Jacobson — Un caso de uso se define como una forma o patrón o ejemplo concreto de utilización, un escenario que comienza con algún usuario del sistema que inicia alguna transacción a secuencia de eventos interrelacionados.

Stroustrup — Frecuentemente esto quiere decir que el programador debe centrarse en las preguntas: ¿cómo se crean los objetos de ésta clase? ¿Pueden los objetos de ésta clase copiarse y/o destruirse? ¿Qué operaciones pueden hacerse en esos objetos? Si no hay buenas respuestas a tales preguntas, el concepto probablemente no estaba 'limpio' desde el principio, y podría ser buena idea pensar un poquito más sobre el problema y la solución propuesta en lugar de empezar inmediatamente a "codificar alrededor" de los problemas.

Halbert and O'Brien — Uno no siempre diseña tipos en una jerarquía de tipos comenzando por un supertipo y creando a continuación los subtipos. Frecuentemente, uno crea varios tipos aparentemente dispares, se da cuenta de que están relacionados, y entonces factoriza sus características comunes en uno o más supertipos... Normalmente se necesitan varias pasadas arriba y abajo para producir un diseño del programa correcto y completo.

Stroustrup — Las reorganizaciones más habituales en una jerarquía de clases son la factorización de las partes comunes de dos clases en una nueva clase, y la división de una clase en otras dos nuevas.

Whitehead — Al aliviar el cerebro de todo el trabajo innecesario, una buena notación lo libera para concentrarse en problemas más avanzados.

Defence Science Board — El desarrollo de software es y será siempre una tecnología que exige trabajo intensivo [...]. Aunque nuestras máquinas pueden hacer el trabajo sucio y ayudarnos a tener bajo control nuestros edificios, el desarrollo de un concepto es la quintaesencia de la actividad humana [...]. La parte del desarrollo de software

que no va a desaparecer es la creación de estructuras conceptuales; la parte que puede desaparecer es la labor de expresarlas.

Kleyn and Gingrich — Uno debe comprender tanto la estructura como la función de objetos involucrados. Uno debe comprender la estructura taxonómica de las clases, los mecanismos de herencia utilizados, los comportamientos individuales de los objetos y el comportamiento dinámico del sistema en su conjunto. El problema es un tanto análogo al de ver un evento deportivo tal como un partido de tenis o fútbol. Se requiere muchos ángulos de cámara diferentes para proporcionar una comprensión de lo que está sucediendo. Cada cámara revela aspectos particulares de la acción que no podrían ser transmitidos por una sola cámara.

Brocks — La integridad conceptual es la consideración más importante en el diseño de un sistema.

Mellors, Hecht, Tryon and Hywari — El propósito del análisis es proporcionar una descripción de un problema. La descripción debe ser completa, consistente, legible y revisable por las diversas partes interesadas, [y] verificable frente a la realidad.