Universidad de Valladolid

E. T. S. DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

Grado en Ingeniería Informática

Diagnosis de fallos de un sistema de climatización

Alumno: Alberto Hernández Cerezo

Tutores: José Belarmino Pulido Junquera
Carlos J. Alonso González

Agradecimientos

Como siempre, mi más sincero agradecimiento a Belarmino y mis padres, por su implicación a lo largo del desarrollo del trabajo, sin la cual el llegar a este punto habría sido imposible.

Agradecimientos también a los miembros del GSI por su colaboración y apoyo con el desarrollo del proyecto, y a mi familia y amigos por haber estado conmigo en todo momento, especialmente Álvaro y María, por estar dándome siempre sus más sinceros ánimos; y Ángel y Raquel, cuya compañía en el desarrollo de este trabajo ha hecho de esta carga algo mucho más fácil de llevar.

Por último dedicar este agradecimiento también a Orlando, antiguo tutor mío sin el cual no hubiese llegado a donde estoy a día de hoy.

A todos vosotros muchas gracias por estar ahí.

Abstract

El marco teórico de este trabajo es el Diagnóstico Basado en Consistencia (DBC), que es la aproximación del campo de la Inteligencia Artificial al Diagnóstico basado en Modelos.

Mediante las técnicas comprendidas por dicha aproximación es posible efectuar la detección, aislamiento e identificación en sistemas reales. Sin embargo existen problemas derivados de la presencia de incertidumbre en los sistemas.

Las RBD permiten trabajar con incertidumbre en sistemas. Su principal inconveniente es el alto coste computacional asociado a su proceso de inferencia. Bajo esta premisa, resulta oportuno trabajar con RBD lo más pequeñas posibles. En este trabajo para estimar el comportamiento de un modelo podremos utilizar la simulación o bien los filtros de partículas, que permiten aproximar la inferencia realizada por una RBD.

Para estudio del diagnóstico de sistemas recurrimos a la técnica de DBC empleando Posibles Conflictos (PCs). Los PCs son una técnica de compilación que permiten aplicar, bajo ciertas suposiciones, la DBC en sistemas dinámicos en línea. Cada PC se concibe como conjuntos de ecuaciones minimales sobredeterminados con redundancia analítica y asignación causal. Mediante los PCs es posible descomponer el sistema en subsistemas con Redes Bayesianas Dinámicas (RBD) asociadas, de menor complejidad que una RBD del sistema completo, que pueden ser simuladas independientemente.

En esta memoria se describe el trabajo realizado para el estudio de la técnica de DBC empleando PCs y su aplicación práctica a un caso de estudio concreto asociado a un sistema de climatización AHU-9, proporcionado por la Universidad College Cork, de Irlanda.

El objetivo del trabajo consiste en implementar la técnica de los PCs sobre este sistema dentro del marco de la Competición internacional de diagnóstico, DXC.

En el trabajo se han analizado los modelos existentes del AHU-9 en Modelica, se han simplificado y se ha generado una versión equivalente en Matlab. Además, se ha analizado el modelo para extraer los PCs y estos se han implementado como nuevos modelos en Matlab. Finalmente, se ha desarrollado una herramienta de diagnosis emulando la DXC donde se ha probado la corrección de los PCs en distintos escenarios: sin fallos y con distintos tipos de fallos, tanto en válvulas como sensores, obteniendo resultados satisfactorios.

Abstract

The theoretical context of this work is the Consistency-Based Diagnosis (CBD), which is the approximation of the Artificial Intelligence's field to the Model-Based Diagnosis (MBD).

By using the techniques developed by this approximation it is possible to achieve the detection, isolation and identification of fails in real systems. However, there are problems caused by of the presence of the uncertainty in systems.

DBNs allow us to work with uncertainty in systems. Their main disadvantage is the high computational cost associated to its inference process. Because of that, it is necessary to work with the smallest DBN we can use. In this work, to estimate our model's behavior, it is possible to use simulation or particle filters, which can approximate the inference made by a DBN.

For system diagnosis studies we use the CBD technique using PCs. PCs are a compilation technique that allows the application of CBD in online dynamic systems, under certain suppositions. Every PC is conceived like overdeterminated minimal equations sets with analytical redundancy and causal assignment. Thanks to the PCs we are able to factorize a system in subsystems with Dynamic Bayesian Networks (DBN) associated, less complex than a complete system's RBD, that can be simulated independently.

In this memory it is described the job developed for the CBD technique using PCs studio and its practical use in a concrete case of study associated to a climate control system AHU-9, provided by the College Cork University, of Ireland.

The main goal of this work consists on implementing the PCs technique over this system, inside the Diagnostic International Competition's context, DXC.

In this work, Modelica AHU-9 existing models have been analyzed, simplified and reimplemented in an equivalent Matlab version. In addition to that, this model has been studied to extract every PC contained in it, that have been implemented like new Matlab models. Finally, a new diagnosis tool has been developed. This software emulates a DXC contest simulation, which evaluates the correction of the PC's behavior in different scenarios: with no fails and with different kinds of fails (in valves and sensors), obtaining successful scores.

Índice de Contenidos:

LISTA DE FIGURAS **11**

INTRODUCCIÓN **15**

1.1. CONTEXTO 17
1.2. OBJETIVOS 19
1.2. ORGANIZACIÓN DE LA MEMORIA 19
1.3. AGRADECIMIENTOS 21

ESTADO DEL ARTE **23**

2.1. INTRODUCCIÓN 25
2.2. DIAGNOSIS Y FALLOS 25
2.2.1. ¿Qué es un fallo? *25*
2.2.2. Tipos de Fallos *26*
2.4. DIAGNOSIS BASADOS EN MODELOS 29
2.4.1. Aproximación "Fault Detection and Isolation" *30*
2.4.2. Aproximación DX *31*
2.4.3. Diagnosis Basada en Consistencia (DBC) *33*
2.5. CONCLUSIONES 35

DIAGNÓSTICO BASADO EN CONSISTENCIA UTILIZANDO POSIBLES CONFLICTOS **37**

3.1. INTRODUCCIÓN 39
3.2. CASO DE ESTUDIO 39
3.2.1. Sistema de Tres Tanques *39*
3.3. POSIBLES CONFLICTOS 40
3.3.1. Definición de Posibles Conflictos *40*
3.3.2. Generación de Posibles Conflictos *41*
3.4. REDES BAYESIANAS DINÁMICAS 45
3.4.1. Definición de Redes Bayesianas Dinámicas *45*
3.5. FILTRO DE PARTÍCULAS 47
3.5.1. ¿Qué es un Filtro de Partículas? *47*
3.5.2. Introducción al ámbito teórico del Filtro de Partículas *47*
3.5.3. Filtro de Partículas y Algoritmo de Importancia de Muestreo Secuencial *48*
3.5.4. Aplicación de un Filtro de Partículas Genérico *50*
3.6. DIAGNOSIS MEDIANTE REDES BAYESIANAS DINÁMICAS 53
3.7. CONCLUSIONES 54

MODELO DE CASO DE ESTUDIO **55**

4.1. INTRODUCCIÓN 57
4.2. SISTEMA REAL AHU-9: 57
4.2.1. Introducción *57*

4.2.2. Descripción del sistema AHU-9 ..... *57*
4.3. MODELICA: 59
4.3.1. ¿Qué es Modelica? *59*
4.3.2. Modelado de sistemas en Modelica *59*
4.3.3. Modelado de Sistemas AHU-9 ..... *61*
4.4. MODELO SOFTWARE DEL SISTEMA AHU-9: 62
4.4.1. Introducción *62*
4.4.2. Descripción del modelo de sistema AHU-9 ..... *62*
4.4.3. Componentes del modelo de sistema AHU-9 ..... *76*
4.5. MODELO SOFTWARE DEL SISTEMA AHU-9 SIMPLIFICADO 78
4.5.1. Introducción *78*
4.5.2. Descripción del modelo de sistema AHU-9 Simplificado *79*
4.5.3. Componentes del modelo de sistema AHU-9 Simplificado *91*
4.5.4. Analisis y Estudio de las "Health Variables" *93*
4.6. CONCLUSIONES 97

CASO DE ESTUDIO 101

5.1. INTRODUCCIÓN 103
5.2. PRESENTACIÓN DEL SISTEMA 103
5.2.1. Introducción *103*
5.2.2. Reescritura del sistema AHU-9 a una notación funcional *104*
5.3. ECUACIONES DEL SISTEMA COMPLETO Y DE LOS PCS 112
5.4. PROCESO DE DIAGNÓSTICO DEL SISTEMA AHU-9: 114
5.4.1. Detección de fallos *114*
5.4.2. Escenarios de fallo *115*
5.5. ANÁLISIS DE RESULTADOS Y CONCLUSIONES 135

CONCLUSIONES 139

6.1. INTRODUCCIÓN 141
6.2. CONCLUSIONES 141
6.3. LÍNEAS DE TRABAJO FUTURAS 143

REFERENCIAS 145

ANEXOS 149

8.1. INTRODUCCIÓN 151
8.2. DESCRIPCIÓN DEL SOFTWARE DE DIAGNÓSTICO AUTOMÁTICO 151
8.2.1. Introducción *151*
8.2.2. Listado de Ficheros *151*
8.3. INSTRUCCIONES DE USO DEL SOFTWARE DE DIAGNÓSTICO AUTOMÁTICO 153
8.4. CÓDIGO FUENTE DEL MODELO DE SISTEMA AHU-9: 156
8.4.1. Introducción *156*
8.4.2. Código Fuente *156*
8.5. DOCUMENTACIÓN SOFTWARE DEL MÓDULO FILTRO DE PARTÍCULAS 161
8.5.1. Introducción *161*
8.5.2. Documento Software *162*

8.6. DATOS DE SIMULACIÓN DEL SISTEMA AHU-9 174

Lista de Figuras:

FIGURA 1.1 ESQUEMA SIMBÓLICO DEL RAZONAMIENTO BASADO EN MODELOS 17

FIGURA 2.1 ESQUEMA TEÓRICO DEL DIAGNÓSTICO BASADO EN MODELOS 29

FIGURA 2.2 CLASIFICACIÓN DE LAS TÉCNICAS DE DIAGNOSIS DE LA COMUNIDAD DX 32

FIGURA 3.1 SISTEMA DE TRES TANQUES CONECTADOS 39

FIGURA 3.2 EJEMPLO DE HIPERGRAFO PARA EL MODELADO DE UN SISTEMA DE TRES TANQUES 41

FIGURA 3.3 POSIBLE CONFLICTO 1 (PC1) DEL SISTEMA DE TRES TANQUES 43

FIGURA 3.4 POSIBLE CONFLICTO 2 (PC2) DEL SISTEMA DE TRES TANQUES 44

FIGURA 3.5 POSIBLE CONFLICTO 3 (PC3) DEL SISTEMA DE TRES TANQUES 44

FIGURA 3.6 EJEMPLO DE RED BAYESIANA DINÁMICA PARA EL SISTEMA DE TRES TANQUES (RBD) 46

FIGURA 4.1 SISTEMA REAL AHU-9 57

FIGURA 4.2 MODELO DE SISTEMA AHU-9 MODELICA 63

FIGURA 4.3 MIXING BOX (COMPARTIMENTO DE MEZCLA) 64

FIGURA 4.4 OUTDOORSOURCE 65

FIGURA 4.5 RETURNFAN 65

FIGURA 4.6 RETURNSOURCE 66

FIGURA 4.7 COOLINGCOIL 67

FIGURA 4.8 HYSTERESIS 67

FIGURA 4.9 CC_WPUMP 68

FIGURA 4.10 CC_WSOURCE 68

FIGURA 4.11 HEATINGCOIL 69

FIGURA 4.12 HYSTERESIS 70

FIGURA 4.13 HC_WPUMP 70

FIGURA 4.14 HC_WSOURCE 71

FIGURA 4.15 HUMIDIFIER 72

FIGURA 4.16 HYSTERESIS 73

FIGURA 4.17 H_STEAMPUMP..... 73
FIGURA 4.18 H_WSOURCE 74
FIGURA 4.19 SUPPLYFAN..... 74
FIGURA 4.20 SUPPLYSINK..... 74
FIGURA 4.21 MODELO DE SISTEMA AHU-9 SIMPLIFICADO 79
FIGURA 5.1 MODELO DE SIMULACIÓN DEL SISTEMA AHU-9..... 106
FIGURA 5.2 RBD ASOCIADA AL PC1 112
FIGURA 5.3 RBD ASOCIADA AL PC2 113
FIGURA 5.4 RBD ASOCIADA AL PC3 113
FIGURA 5.5 ESCENARIO NOMINAL (PREHEATING SENSOR) 116
FIGURA 5.6 ESCENARIO NOMINAL (COOLING SENSOR)..... 117
FIGURA 5.7 ESCENARIO NOMINAL (REHEATING SENSOR) 117
FIGURA 5.8 ESCENARIO DE FALLO CC VALVE 5% (PREHEATING SENSOR – PC1) 118
FIGURA 5.9 ESCENARIO DE FALLO CC VALVE 5% (COOLING SENSOR – PC2)..... 118
FIGURA 5.10 ESCENARIO DE FALLO CC VALVE 5% (REHEATING SENSOR – PC3) 119
FIGURA 5.11 ESCENARIO DE FALLO CC VALVE 20% (PREHEATING SENSOR – PC1) 119
FIGURA 5.12 ESCENARIO DE FALLO CC VALVE 20% (COOLING SENSOR – PC2)..... 119
FIGURA 5.13 ESCENARIO DE FALLO CC VALVE 20% (REHEATING SENSOR – PC3) 120
FIGURA 5.14 ESCENARIO DE FALLO CC VALVE 50% (PREHEATING SENSOR – PC1) 120
FIGURA 5.15 ESCENARIO DE FALLO CC VALVE 50% (COOLING SENSOR – PC2)..... 120
FIGURA 5.16 ESCENARIO DE FALLO CC VALVE 50% (REHEATING SENSOR – PC3) 121
FIGURA 5.17 ESCENARIO DE FALLO PC SENSOR 5% (PREHEATING SENSOR – PC1)..... 123
FIGURA 5.18 ESCENARIO DE FALLO PC SENSOR 5% (COOLING SENSOR – PC2) 123
FIGURA 5.19 ESCENARIO DE FALLO PC SENSOR 5% (REHEATING SENSOR – PC3)..... 123
FIGURA 5.20 ESCENARIO DE FALLO PC SENSOR 20% (PREHEATING SENSOR – PC1)..... 124
FIGURA 5.21 ESCENARIO DE FALLO PC SENSOR 20% (COOLING SENSOR – PC2)..... 124
FIGURA 5.22 ESCENARIO DE FALLO PC SENSOR 20% (REHEATING SENSOR – PC3)..... 124
FIGURA 5.23 ESCENARIO DE FALLO PC SENSOR 50% (PREHEATING SENSOR – PC1)..... 125
FIGURA 5.24 ESCENARIO DE FALLO PC SENSOR 50% (COOLING SENSOR – PC2)..... 125
FIGURA 5.25 ESCENARIO DE FALLO PC SENSOR 50% (REHEATING SENSOR – PC3)..... 125
FIGURA 5.26 ESCENARIO DE FALLO PC UA 5% (PREHEATING SENSOR – PC1)..... 128

FIGURA 5.27 ESCENARIO DE FALLO PC UA 5% (COOLING SENSOR – PC2)..... 128
FIGURA 5.28 ESCENARIO DE FALLO PC UA 5% (REHEATING SENSOR – PC3) 129
FIGURA 5.29 ESCENARIO DE FALLO PC UA 20% (PREHEATING SENSOR – PC1) 129
FIGURA 5.30 ESCENARIO DE FALLO PC UA 20% (COOLING SENSOR – PC2)..... 129
FIGURA 5.31 ESCENARIO DE FALLO PC UA 20% (REHEATING SENSOR – PC3) 130
FIGURA 5.32 ESCENARIO DE FALLO PC UA 50% (PREHEATING SENSOR – PC1) 130
FIGURA 5.33 ESCENARIO DE FALLO PC UA 50% (COOLING SENSOR – PC2)..... 130
FIGURA 5.34 ESCENARIO DE FALLO PC UA 50% (REHEATING SENSOR – PC3) 131
FIGURA 5.35 ESCENARIO DE FALLO PC MB 5% (PREHEATING SENSOR – PC1)..... 132
FIGURA 5.36 ESCENARIO DE FALLO PC MB 5% (COOLING SENSOR – PC2) 132
FIGURA 5.37 ESCENARIO DE FALLO PC MB 5% (REHEATING SENSOR – PC3)..... 132
FIGURA 5.38 ESCENARIO DE FALLO PC MB 20% (PREHEATING SENSOR – PC1)..... 133
FIGURA 5.39 ESCENARIO DE FALLO PC MB 20% (COOLING SENSOR – PC2) 133
FIGURA 5.40 ESCENARIO DE FALLO PC MB 20% (REHEATING SENSOR – PC3)..... 133
FIGURA 5.41 ESCENARIO DE FALLO PC MB 50% (PREHEATING SENSOR – PC1)..... 134
FIGURA 5.42 ESCENARIO DE FALLO PC MB 50% (COOLING SENSOR – PC2) 134
FIGURA 5.43 ESCENARIO DE FALLO PC MB 50% (REHEATING SENSOR – PC3)..... 134
FIGURA 8.1 EJEMPLO DE FICHERO DE LISTA DE CONFIGURACIONES 154
FIGURA 8.2 EJEMPLO DE ESCENARIO DE FALLO 155
FIGURA 8.3 DIAGRAMA DE PAQUETES..... 166
FIGURA 8.4 DIAGRAMA DE CLASES DE DISEÑO..... 167
FIGURA 8.5 INTERFAZ DE USUARIO..... 169

Capítulo 1:

Introducción

1.1. Contexto

El presente documento recoge toda la labor documental asociada al diagnóstico del sistema "Air Handling Unit - Nine" AHU - 9.

Vivimos en un mundo fuertemente industrializado. Actualmente todo entorno laboral cuenta con dispositivos electrónicos y maquinaria de trabajo automatizada. El empleo de estas herramientas implica la realización de tareas profesionales de forma efectiva y rentable. Esto supone un aumento en la calidad de productos y seguridad de los operarios. Como claros ejemplos tenemos las cadenas de montaje industrial, empleadas para el ensamblaje en masa de maquinaria de todo tipo: coches, dispositivos electrónicos, embalajes, etc.

Dentro de estos entornos, garantizar que los sistemas operan de forma adecuada es de vital importancia. La presencia de fallos en sistemas puede suponer pérdidas de producción y costes; y en ocasiones incluso poner en riesgo la seguridad del trabajador.

Ante la posibilidad que puedan tener lugar dichas situaciones, surge la necesidad de establecer una supervisión rigurosa del correcto funcionamiento de los sistemas. Esta tarea define lo que entendemos a día de hoy como "diagnóstico de fallos". Según L.Console, se puede definir el diagnóstico como:

"La tarea que, dado un sistema y un conjunto de observaciones de un funcionamiento con fallo, determina qué está mal en el sistema con el objetivo de recuperar su modo de funcionamiento correcto."

Dicha labor no es nada trivial, y requiere un estudio en profundidad de la misma. Actualmente existen diversas ramas de investigación orientadas al estudio y desarrollo de "Técnicas de Diagnosis", donde se persigue la detección, aislamiento e identificación de fallos en sistemas complejos. Dentro de estas vías de investigación, cabe destacar un tipo de diagnosis particular, en el cual se centra nuestra investigación: la "Diagnosis Basada en Modelos", o DBM.

La Diagnosis Basada en Modelos parte del concepto de modelo, abstracción de un sistema real representado de forma simbólica (habitualmente mediante un sistema de ecuaciones), el cual permite obtener una estimación del comportamiento correcto del mismo. La técnica de DBM deriva del paradigma principal basado en conocimiento conocido como "razonamiento basado en modelos".

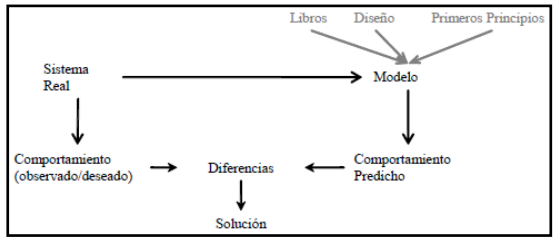


Figura 1.1 Esquema simbólico del Razonamiento Basado en Modelos

Mediante la DBM podemos efectuar una estimación del estado del sistema real en base a las discrepancias de comportamiento detectables entre un cierto sistema real y su modelo. Aplicando al modelo las entradas que afectan a su sistema real correspondiente, es posible obtener unos resultados hipotéticamente análogos a los que se deberían obtener a partir de las observaciones del sistema real. Hay que tener en cuenta, en referencia a lo expuesto, que a la hora de trabajar con sistemas complejos debemos hacer frente al problema de incertidumbre, generalmente motivado por los sensores de los sistemas (componentes que nos proporcionan información acerca de su estado). Dicha incertidumbre puede motivar la presencia de discrepancias, no significativas para el diagnóstico de sistemas, entre datos del sistema real y su modelo. Otro factor de incertidumbre adicional es la posible discrepancia en los parámetros del modelo frente a lo que serían los parámetros físicos del sistema.

Ignorando el factor de incertidumbre en sensores, se puede formular que de la discrepancia entre valores medidos y simulados (de sistema y modelo, respectivamente), se determina que el sistema real no está respondiendo de forma adecuada a lo que se espera de él; es decir, el sistema sufre un fallo. Esta es la filosofía básica de la que parte la Diagnosis Basada en Consistencia, DBC; una de las diversas técnicas de diagnosis presente en el campo de la DBM, y más concretamente la aproximación desarrollada en el Grupo de Sistemas Inteligentes del Depto. de Informática, conocida como Técnica de los Posibles Conflictos (PCs) [Pulido&Alonso2004].

Sin entrar en detalles del desarrollo de dicha técnica, podemos resumir la labor de diagnosis en el estudio completo del sistema a diagnosticar, la generación de un modelo asociado al mismo, y la aplicación de la técnica de diagnosis basada en PCs para la detección, aislamiento e identificación de fallos. En el presente proyecto se plantea la idea de abordar el diagnóstico del sistema "Air Handling Unit - Nine" (AHU - 9).

Se trata de un modelo de sistema de aire acondicionado comercial instalado en una escuela de música en Cork, Irlanda. Dicho modelo es uno de los posibles sistemas propuestos por la organización DXC-13' para su diagnóstico en el concurso de diagnosis celebrado en el año 2013. La "Competición Internacional de Diagnosis", DXC, consiste en una actividad competitiva comprendida dentro de la agenda de actividades del "Taller Internacional de Principios de Diagnosis" (International Workshop on Principles of Diagnosis - DX). Dicho taller se concibe con un evento internacional que se viene organizando desde 1989, y va dirigido a la comunidad de profesionales que realizan investigación en el diagnóstico basado en modelos dentro del campo de la IA. En el 2013 la organización celebrará la 24º edición de dicho taller. En él se celebrará por cuarto año consecutivo la "Competición de Diagnosis Internacional". Esta actividad se diseñó con el objetivo de acelerar la investigación en las diferentes materias que competen al taller; motivando el desarrollo de plataformas software que proporcionen un medio rápido, accesible y eficiente a las tecnologías de diagnóstico. El aspecto práctico del concurso se resume en la implementación de los algoritmos y técnicas de diagnóstico de los investigadores, ofreciéndoles un entorno de trabajo que les permita probar y validar sus técnicas frente a sistemas reales, así como comparar su rendimiento y eficacia.

1.2. Objetivos

Recopilando todo lo expuesto podemos resumir el objetivo del proyecto en una línea:

Efectuar el diagnóstico del sistema AHU - 9 aplicando la técnica de diagnosis basada en Posibles Conflictos; ajustándonos a las restricciones propias del marco del concurso DXC -13'.

Esta es la meta final que se persigue. Sin embargo, para alcanzar dicho objetivo se requiere completar una serie de puntos clave entendidos a modo de sub-tareas, que a continuación enumeraremos:

- *Estudio de la técnica de diagnosis basada en Posibles Conflictos.*
- *Estudio de las herramientas teóricas asociadas a la técnica de diagnosis basada en Posibles Conflictos.*
- *Estudio del sistema AHU - 9, tanto su implementación real como su modelo, ya diseñado por la organización del concurso.*
- *Estudio de las herramientas técnicas asociadas al modelado del sistema AHU - 9.*
- *Implementación de los modelos de simulación del AHU-9 para los PCs encontrados en el modelo.*
- *Realización de la labor de diagnosis del sistema AHU -9 según las directrices del concurso, aplicando la técnica de los Posibles Conflictos.*
- *Documentación de la labor formativa y de diagnosis para la generación de un informe memoria susceptible de ser validado y aprobado como Trabajo de Fin de Grado (TFG).*

Dado que al encuadrarse el proyecto dentro de un concurso cuya organización es ajena a la labor del participante, en los objetivos descartamos toda consideración referente a resultados de la competición. Nuestra labor se centra en el estudio de las técnicas de diagnóstico y la aplicación del mismo a un caso de estudio práctico y accesible al público.

1.3. Organización de la Memoria

A continuación se presenta una lista de los diferentes apartados en los que se organiza nuestra memoria. Cada entrada de la lista presenta un apartado concreto. Todos ellos han sido ordenados por orden de disposición la documentación del proyecto:

1. Introducción: descripción del contexto en el que se encuadra el proyecto y definición del mismo.
   1.1. Contexto.
   1.2. Objetivos.
   1.3. Estructura del Documento.
2. Estado del Arte: contexto teórico en el que se encuadra el proyecto. Descripción de marco teórico de las técnicas de diagnosis.
   2.1. Conceptos Fundamentales.
   2.2. Sistemas Basados en Modelos.
   2.3. Diagnosis Basada en Modelos.
3. Diagnóstico Basado en Consistencia utilizando Posibles Conflictos.
   3.1. Introducción.
   3.2. Redes Bayesianas Dinámicas (DBNs).
   3.3. Posibles Conflictos.
   3.4. Filtro de Partículas.
4. Modelo del sistema AHU - 9.
   4.1. Introducción.
   4.2. Sistema Real AHU-9.
   4.3. ¿Qué es Modelica?
   4.4. Modelo de Sistema AHU-9 Completo.
   4.5. Modelo de Sistema AHU-9 Simplificado.
5. Caso de Estudio: documentación detallada de la labor de diagnóstico del sistema AHU - 9 empleando la técnica de los PCs.
   5.1. Presentación del Sistema.
   5.2. Posibles Conflictos.
   5.3. DBNs de Sistema Completo y PCs.
   5.4. Proceso de Diagnóstico del Sistema AHU-9:
      5.4.1. Detección de Fallos.
      5.4.2. Aislamiento de Fallos.
      5.4.3. Escenarios de Fallo Detectados.
   5.5. Comparación comportamiento DBN completa y DBN de PCs.
6. Conclusiones: explicación e interpretación de los resultados obtenidos al final del proyecto.
   6.1. Líneas de Trabajo Futuras.
7. Apéndice.

1.4. Agradecimientos

Para el desarrollo del presente proyecto se requiere de la disposición de los modelos de sistema del AHU-9. Antes de empezar con los contenidos teóricos de la memoria consideramos oportuno hacer una mención de los doctores Provan y Feldman, de la Universidad College Cork, de Irlanda.

Agradecemos su disposición de cara a proporcionarnos los modelos del sistema AHU-9 diseñados para la competición de diagnosis, su descripción y un conjunto de datos de entrada para la realización de pruebas; además de su labor de apoyo a la hora de responder a nuestras cuestiones acerca de los mismos.

Capítulo 2:

Estado del Arte

2.1. Introducción

En el siguiente capítulo de la memoria se procede a realizar una breve descripción del contexto teórico en el que se encuadra nuestro proyecto. A lo largo de las siguientes líneas se expondrá un resumen detallado donde se explicará el concepto de diagnosis y fallo y las técnicas de diagnosis que nos atañen.

Mediante esta información se busca facilitar al usuario unos conocimientos elementales para la comprensión de contenidos del presente documento, y un punto de partida para introducir nuestro proyecto de investigación.

2.2. Diagnosis y Fallos

2.2.1. ¿Qué es un fallo?

Como venimos recogiendo desde la introducción del proyecto, se entiende la labor de diagnóstico como la "identificación de la naturaleza y causa de algo". Se trata de una definición ambigua e inexacta en comparación a la realizada por L. Console [Console2000] que mencionábamos anteriormente, pero sencilla y fácil de entender.

La tarea del diagnóstico de sistemas viene motivada por el concepto de fallo. Efectuamos la diagnosis con el fin de detectar, aislar e identificar fallos. Esta forma particular de diagnosis se la conoce como "diagnóstico de fallos".

Antes de abordar el estudio de las técnicas de diagnóstico actuales resulta oportuno desarrollar el concepto de fallo y las posibles clasificaciones que se pueden hacer de los mismos.

Una definición general de fallo es aquella en la que se refiere al mismo como una anomalía en la configuración de un cierto sistema que genera un comportamiento del mismo distinto al adecuado. Si queremos recurrir a una definición más formal, resulta oportuno mencionar la definición de fallo formulada por Isermann y Ballé [Isermann&Ballé1997]:

> *"Desviación no permitida en al menos una propiedad o parámetro del sistema que lo aparta de su situación aceptable, habitual o estándar"*

Derivado del concepto de fallo, surge la necesidad común de conseguir que nuestros sistemas sean tolerantes a fallos; esto es, que sean capaces de reducir las consecuencias de un fallo en la medida de sus posibilidades.

Dicho objetivo se alcanza a través de la labor de diagnóstico de fallos. La diagnosis de fallos se concibe como un proceso iterativo dividido en tres etapas básicas:

1. **Detección de Fallos:** determinar si hay presencia de fallo o no en el sistema actual. Esta etapa contempla la determinación del instante en el que ocurrió el fallo.

2. **Aislamiento o Localización de Fallos:** determinar los componentes donde se localiza el fallo. Consiste en la localización del fallo.

3. **Identificación de Fallos:** caracterizar el fallo por tipo y severidad; estimando instante en que apareció y magnitud.

2.2.2. Tipos de Fallos

A nivel de estudio de diagnosis, se considera la clasificación de fallos en base a diversos criterios. Esto se debe a que no es posible establecer una única clasificación de fallos. A continuación enumeraremos cada uno de ellos:

- **Considerando el "modelo de proceso":**
  - **Fallo Aditivo:** entradas desconocidas que actúan en la planta. Presentan un valor nulo por defecto, cuya alteración provoca cambios en la salida del sistema, independientemente del valor de las entradas conocidas (Ejemplo: entrada inesperada de un caudal de agua fría).
  - **Fallo Multiplicativo:** cambios en algunos parámetros del sistema. Su aparición genera una alteración que provoca cambios en la salida del sistema, los cuales dependen de las entradas al mismo (Ejemplo: fallo en bomba de agua de circuito de refrigeración).
- **Considerando la dependencia con el tiempo:**
  - **Fallo Abrupto:** aparece de forma repentina en un cierto instante de tiempo, su magnitud permanece constante.
  - **Fallo Incipiente:** aparece de forma gradual, su magnitud varía con el tiempo.
- **Considerando la permanencia en el tiempo:**
  - **Fallo Permanente:** aparece de manera constante a lo largo del tiempo.
  - **Fallo Intermitente:** aparece y desaparece continuamente.
- **Considerando los componentes afectados por el fallo:**
  - **Fallo en la Planta (o Fallo en el Proceso):** cambia las propiedades dinámicas de entrada/salida del sistema.

- **Fallo en Sensor:** obtención de medidas erróneas en base a errores en las lecturas del sensor. Las propiedades de la planta no se ven afectadas.
- **Fallo en Actuador:** interrupción o modificación de la acción del controlador sobre la planta. Las propiedades de la planta no se ven afectadas.

2.3. Razonamiento Basados en Modelos

Dentro de nuestro estudio se ha considerado, del conjunto global de técnicas de diagnosis disponibles a día de hoy, la técnica de diagnosis basada en PCs, derivada de la DBC. Esta técnica se engloba dentro de un conjunto de técnicas de diagnosis de mayor rango conocido como Diagnosis Basada en Modelos (DBM).

Actualmente no existe una taxonomía común para el ámbito científico con la que clasificar las diferentes técnicas de diagnóstico. De entre las disponibles, y en referencia a la DBM, cabe destacar la realizada por Getler [Getler1998], que distingue entre dos tipos diferentes de técnicas de diagnóstico:

- **Técnicas no Basadas en Modelos:** técnicas de diagnóstico que no se asientan en el uso de modelos abstractos del sistema para su diagnóstico (Ejemplos: redundancia física, uso de sensores especiales, razonamiento lógico).

- **Técnicas Basadas en Modelos:** técnicas de diagnosis que recurren al uso de modelos abstractos del sistema (matemáticos) para su diagnóstico. Se distingue dentro de este grupo una clasificación entre los de estimación analítica y redundancia analítica. Estos últimos contemplan cierta relación con otros conceptos teóricos como el Filtro de Kalman (FK) o los observadores de estado, los cuales atañen a la técnica de PCs y desarrollaremos en detalle más adelante.

Entendido el ámbito en el que encuadramos nuestra técnica de diagnosis, resulta apropiado entrar a detallar el concepto de DBM. Antes de ello es necesario describir el concepto de Sistemas Basados en Modelos (SBM), que es un caso particular de Sistema Inteligente (SI). Se define un Sistema Inteligente como un Agente Racional. Entendemos el agente como una entidad capaz de percibir su entorno a través de sensores, y modificar el mismo por medio de actuadores. Se dice que el agente es racional cuando la entidad presenta un comportamiento tal que es capaz de realizar acciones de las que se espera que maximicen su medida de rendimiento en base a una secuencia de percepciones adquiridas, y el conocimiento que el agente tenga almacenado.

Para entender esto, supongamos que estamos trabajando con un sistema de aire de acondicionado (similar al AHU - 9); podemos considerar éste como un sistema inteligente cuyas percepciones se basan en mediciones de la temperatura ambiente de la sala que aclimata, los actuadores los módulos de enfriamiento y calentamiento del aire, y sus acciones las de calentamiento o enfriamiento del aire entrante. Hasta este punto hemos definido todos los aspectos relativos a un sistema inteligente: entorno, sensores, actuadores, y la función agente, que en este caso es ajustar la temperatura de la sala a la deseada por los usuarios.

Considerando como medida de rendimiento el margen de tiempo empleado para el ajuste de la temperatura de la sala a la deseada; nuestro sistema de acondicionamiento será inteligente si maximiza dicha medida de rendimiento, detectando lo antes posible el estado de la sala y actuando conforme a dicha percepción de manera que adapte la temperatura a los valores deseados.

En nuestro caso de estudio particular, el comportamiento del sistema va enfocado a la detección de fallos, no a como éste es capaz de responder a las percepciones de temperatura para actuar con mayor eficacia.

Volviendo a la línea argumental, existen clasificaciones de sistemas inteligentes en base a diversas características [Balakrishnan98]:

- Sistemas Basados en Conocimiento o Sistemas Expertos.
- Razonamiento Basado en Casos.
- Métodos de Aprendizaje y Minería de Datos.
- Razonamiento Basado en Modelos.

Es este último paradigma, el Razonamiento Basado en Modelos (RBM) el que da nombre a los SBM. El RBM basa la resolución de problemas en la inferencia de resultados a partir de las estimaciones de un sistema a partir de su comportamiento.

En el RBM, el diagnóstico se entiende como un proceso de razonamiento derivable a partir del modelo. En él se procede a detección de diferencias producidas entre el comportamiento del sistema real y el modelo del mismo, el cual se supone que es una representación del comportamiento correcto del sistema real. La discrepancia entre comportamientos indica la presencia de fallo (recordando lo dicho anteriormente sobre los fallos, este razonamiento es exactamente la propia definición del fallo).

Se pueden destacar una serie de puntos a favor en el uso del paradigma de RBM frente a otros tipos de paradigmas y SI:

1. **Independientes de experiencia previa:** para su razonamiento no se requiere de una secuencia de percepciones previas.
2. **Independientes del dispositivo.**
3. **Capacidad para tratar con situaciones complejas (en nuestro caso, fallos múltiples).**
4. **Sólido y completo:** siempre respecto de los modelos.
5. **Capacidad de mantenimiento y reutilización de conocimiento gracias a la creación de bibliotecas de modelos (disponibles desde el diseño).**

Como es lógico, existen también inconvenientes en el uso de dicho paradigma; como son la dificultad de obtención de modelos correctos y el requerimiento de una mayor carga computacional en el proceso de razonamiento. Este último punto puede resultar crítico para nuestros intereses, ya que supone de una limitación de tiempo de ejecución en aplicaciones de tiempo real, lo cual resulta relevante cuando se habla de diagnosis de sistemas en línea.

Entendido la base de razonamiento en la que se asientan el RBM, y que trabajamos con los mismos para efectuar la DBM, estamos en condiciones de desarrollar los aspectos teóricos que atañen a este conjunto de técnicas de diagnosis.

2.4. Diagnosis Basados en Modelos

La Diagnosis Basada en Modelos (DBM) se define como una técnica de diagnóstico de sistemas que sigue los principios del RBM. Entendemos el modelo como la abstracción conceptual de un sistema real, finito y con capacidad de respuesta frente a estímulos equivalentes a los procesados por el sistema real que lo define [Console2000].

En esta técnica, los resultados de la diagnosis se obtienen de la comparación del comportamiento real observado de nuestro sistema con el modelo de funcionamiento del sistema, empleado con el fin de obtener una previsión de la conducta del sistema modelado [Pulido&Alonso2004].

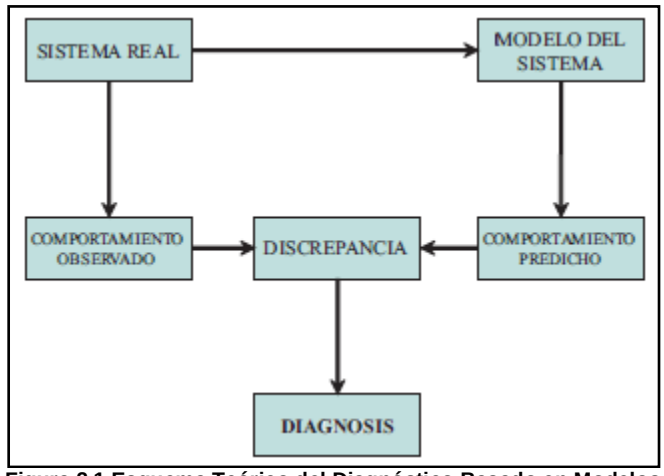


Figura 2.1 Esquema Teórico del Diagnóstico Basado en Modelos

La DBM presenta, frente al resto de técnicas de diagnosis, una serie de ventajas derivadas de su relación con el RBM: capacidad de efectuar un diagnóstico con independencia de la experiencia y los dispositivos empleados, capacidad para tratar con situaciones complejas, como casos de fallo múltiple; y ofrece la posibilidad de crear bibliotecas de modelos para su mantenimiento y reutilización.

Asimismo, la DBM también carga con los inconvenientes propios del RBM, como son la dificultad a la hora de disponer de un modelo preciso del sistema, y los problemas computacionales derivados de la necesidad de calcular todas las posibles diagnosis para un

fallo observado. Este último punto no solo hace referencia al hecho comprobable del elevado coste computacional que requiere simular el modelo de sistema nominal (sin fallo), sino también de la simulación de dicho modelo ajustado a cada uno de los posibles fallos que se pueden considerar dentro del sistema.

El auge de las técnicas de DBM viene influenciado por los avances técnicos aparecidos a lo largo de las décadas de los años 70 y 80 del siglo XX. Se considera la década de los años 70' punto de partida de los sistemas expertos de primera generación, principalmente en el área de la medicina. Estos sistemas, también conocidos como “basados en conocimiento”, se corresponden a uno de los cuatro paradigmas principales para la resolución de problemas.

Los sistemas expertos realizan una identificación y explotación del conocimiento humano para la resolución de problemas. Para ello se parte de las suposiciones de que el conocimiento proviene de la experiencia, que el conocimiento está especificado mediante heurística (reglas de producción del tipo "SI antecedente ENTONCES consecuente"); y que éste puede ser codificado mediante un Lenguaje de Representación de Conocimiento.

Retomando el hilo de la historia de la DBM, la aparición de los primeros sistemas expertos motivaron un estudio y desarrollo de los mismos, que reveló las limitaciones a las que estaban sometidos. Por enumerar algunas de ellos, cabe mencionar la limitación en la forma de expresar el conocimiento mediante asociaciones heurísticas del tipo síntoma-fallo y la imposibilidad de tratar con anomalías no detectadas anteriormente. Esto restringía su aplicación en entornos industriales, al no ser capaces de cubrir la localización de nuevos fallos causados por síntomas no considerados anteriormente.

Sería más tarde, en la década de los años 80, cuando la aparición de nuevas técnicas de diagnosis basada en modelos, cuando se conseguirá dar respuesta a estas limitaciones.

Introducido el concepto de DBM, procedemos a la explicación de las aproximaciones existentes a dicha técnica.

2.4.1. Aproximación “Fault Detection and Isolation”

La aproximación Fault Detection and Isolation (FDI) está comprendida dentro del ámbito de estudio de la Ingeniería de Control Moderna, y su origen se remonta a los años 70.

Esta aproximación emplea modelos numéricos analíticos, expresados de diferentes formas: modelo de espacio de estados, modelo de entrada-salida o modelos de caja negra obtenidos a partir de identificación.

FDI basa su mecanismo de detección de fallos en la generación de residuos [Gertler98]. Se define el residuo como la desviación entre medidas obtenidas del sistema real y estimaciones del modelo de sistema. El residuo es un indicador de fallo en sistema. Bajo la suposición de que los modelos fueran perfectos, los sistemas ideales, y que los modelos representasen el comportamiento correcto esperado del componente, los valores de medidas y estimación deberían ser idénticos, determinando un valor de residuo nulo en todo instante de tiempo.

Los fallos generan anomalías en el sistema, lo que provoca desviaciones en sus salidas. Dichas desviaciones marcan discrepancias en el valor del residuo, que deja de ser nulo, lo que determina la presencia de fallo.

Sin embargo, en el mundo real hacemos frente a sistemas no ideales, y a modelos que no ofrecen una representación exacta del sistema asociado. Estas situaciones pueden motivar la aparición de valores residuales no nulos y no significativos. Ante semejante problema se requiere del desarrollo de técnicas que determinen el valor significativo de un residuo como indicador de fallo.

Existen tres técnicas fundamentales empleadas en la aproximación FDI:

- **Estimación de Parámetros:** identificación inicial de parámetros del modelo de referencia del sistema en ausencia del fallo. A partir de este paso se determinan los valores en línea de dichos parámetros. Empleando las desviaciones de parámetros iniciales y estimados del sistema se efectúa la detección y aislamiento de los mismos.

- **Observadores de Estados:** se definen las variables de salida y control del sistema para la estimación de las variables de estado del mismo. Mediante estas variables se persigue la estimación de las variables de salida del sistema; cuyos valores se emplean para la obtención de residuos, que permitan la detección y aislamiento de fallos.

- **Ecuaciones de Paridad:** empleo del modelo entrada-salida del sistema adaptado para que solo dependa de variables conocidas. La detección y aislamiento se logra por medio de los residuos transformados.

2.4.2. Aproximación DX

La aproximación DX basa su funcionamiento en las tareas de localización e identificación de fallos. No existe una clasificación universalmente aceptada del conjunto de técnicas de diagnóstico dentro del marco DX, sin embargo podemos recurrir a aproximaciones bastante acertadas como la de L.Console para hacernos una idea de la jerarquía de técnicas disponibles en este campo de estudio [Console2000]:

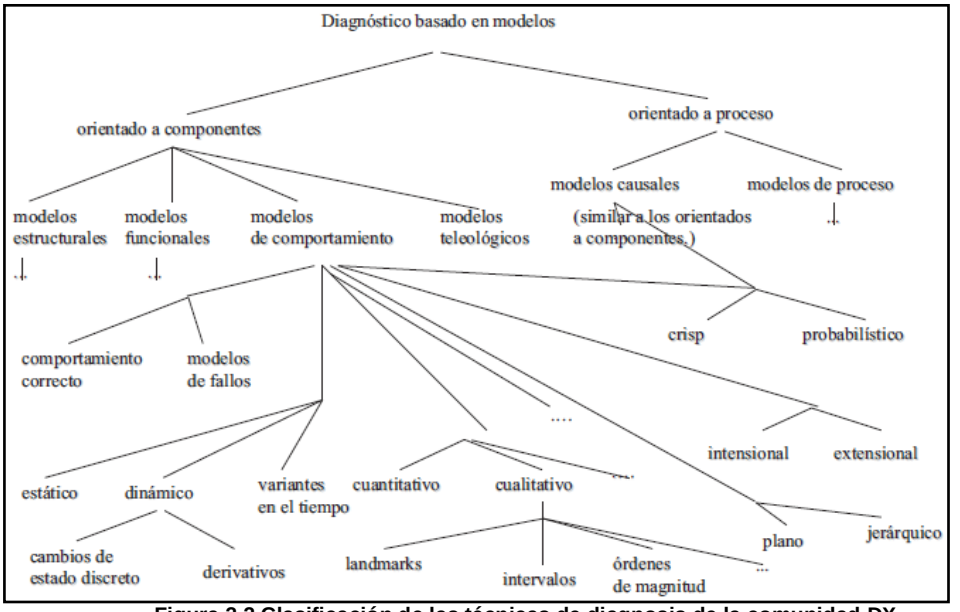


Figura 2.2 Clasificación de las técnicas de diagnosis de la comunidad DX

Dentro de esta clasificación, la Diagnosis Basada en Consistencia (DBC) es la aproximación más común [Console&Kleer92]. Dicha técnica se basa en que, dada la descripción de un sistema y las observaciones disponibles, la detección de inconsistencias entre observaciones y comportamiento esperado permite determinar los componentes del sistema que funcionan de manera indebida; así la corrección de los mismos devuelve el sistema a un hipotético estado de funcionamiento correcto, ajeno a este tipo de inconsistencias.

Esta aproximación tiene sus orígenes en el año 1987, cuando Reiter formalizó teóricamente una aproximación lógica a la diagnosis conocida como "diagnosis a partir de primeros principios". En dicha aproximación se plantea un problema que tiene como punto de partida la descripción de un sistema, junto con una serie de observaciones que entran en conflicto con el comportamiento esperado del sistema. El problema a resolver es la detección de los componentes defectuosos del sistema, cuya restauración devuelva al sistema a un estado consistente; con su comportamiento esperado. De aquí deriva el etiquetado de "consistencia", con el que se alude a esta técnica de diagnosis.

Las ventajas que proporciona esta técnica de diagnosis frente a otras posibles alternativas son las siguientes:

- Solo requiere conocimiento de la estructura y comportamiento del sistema.
- Es capaz de tratar con un rango de problemas cada vez más amplio.
- Es capaz de hacer frente a escenarios de fallo no contemplados anteriormente.

2.4.3. Diagnosis Basada en Consistencia (DBC)

La DBC basa su funcionamiento en el uso de modelos. Esta técnica plantea una definición propia de *sistema*, junto con otros conceptos como los de *síntoma* y *conflicto*, necesarios para entender la misma.

Para la DBC un sistema se define formalmente como una tupla (SD, COMPS, OBS), donde SD hace referencia a la descripción del sistema, compuesta por un conjunto de sentencias de primer orden; COMPS representa el conjunto de componentes del sistema, conjunto finito de constantes; y OBS define el conjunto de observaciones, formado por sentencias de primer orden.

Esta es una notación estándar para referirse a los diferentes elementos de la DBC, más adelante se irán enumerando otras convenciones de notación básica para entender las características de la DBC.

La detección del fallo en la DBC se obtiene a través de inconsistencias. Dentro de la notación técnica, estas inconsistencias son entendidas como *síntomas*. Un *síntoma* es cualquier diferencia entre observaciones del comportamiento del sistema y predicciones obtenidas mediante inferencia. La presencia de síntomas en nuestro sistema supone la existencia de fallo en uno o más de sus componentes, y marca el paso de detección de fallo en el desarrollo de la labor de diagnosis. Una vez se detecta el fallo hay que localizar el componente en el que se encuentra.

Aquí es donde entra en consideración el *conjunto conflicto*. El *conjunto conflicto* se define como un grupo de componentes para los cuales existen indicios de que al menos uno de ellos no funciona correctamente.

Este formalismo puede ser definido de manera formal como la inconsistencia del conjunto de componentes con funcionamiento correcto, en unión a la descripción del sistema y sus observaciones. Esta definición viene recogida en [MOYA2010]:

> *"Un conflicto para (SD, COMPS,OBS) es un conjunto* $\{c_1, ..., c_k\} \subseteq$ *COMPS tal que SD* $\cup$ *obs* $\cup \{\neg ab(c_1), ..., \neg ab(c_k)\}$ *es inconsistente."*

La notación "*ab(c)*" hace referencia al predicado "*ab*", aplicado sobre el componente "*c*", que hace referencia al comportamiento incorrecto del componente "*c*". Siguiendo las recomendaciones expuestas en [BREGÓN2010], prescindiremos de esta notación estándar, pero en desuso, por la del predicado "*ok(c)*", con el que se indica que el componente "*c*" funciona correctamente.

Para la DBC el problema de diagnosis se define como un problema de "asignación de modos". La asignación de modos consiste en la asignación de predicados *ok(x)* y ¬*ok(x)* a cada uno de los elementos *x* del conjunto COMPS, definiéndose así los subconjuntos de COMPS, disjuntos, FAULTY y OK; tales que FAULTY $\cup$ OK = COMPS. La asignación de modos completa se representa según la notación:

$$D(FAULTY,\cdot) \equiv D(FAULTY,COMPS\setminus FAULTY)$$

La definición de **diagnosis** es la de una asignación de modos completa de un determinado sistema (SD,COMPS,OBS) tal que el conjunto:

$$SD \cup OBS \cup D(FAULTY\setminus COMPS)$$

es consistente.

A su vez, del concepto de diagnosis, entendida a través de conjuntos, es posible inferir el concepto de "**diagnosis minimal**". La diagnosis minimal es una diagnosis $D(FAULTY\setminus COMPS)$ tal que no existe ningún subconjunto $FAULTY' \subset FAULTY$ el cual haga que $D(FAULTY'\setminus COMPS)$ sea una diagnosis.

De esta definición de diagnosis minimal es posible extraerse un colorario que determina que para toda diagnosis $D(FAULTY\setminus COMPS)$ existe una diagnosis minimal $D(FAULTY_{min}\setminus COMPS)$, donde $FAULTY_{min} \subseteq FAULTY$.

Consideramos el concepto de diagnosis minimal para hacer frente a los problemas de complejidad que supone el diagnóstico de sistemas. Cuando trabajamos con un sistema, considerando que cada elemento del conjunto COMPS puede tener un predicado *ok(x)* o *¬ok(x)*, existen un total de $2^{|COMPS|}$ posibles diagnosis asociadas a un sistema.

Cada definición de diagnosis lleva asociada una probabilidad estadística de la ocurrencia del mismo. Lógicamente, a mayor cardinalidad del conjunto FAULTY asumimos que un mayor número de componentes está fallando, lo que se traduce a un diagnóstico de fallo menos probable. El hecho de que existan fallos con mayor probabilidad de suceder que otros es lo que motiva a considerar el concepto de diagnosis minimal.

La diagnosis minimal se efectúa recurriendo a los conflictos minimales. El conflicto minimal es un conflicto $\{c_1, ..., c_k\}$ sin ninguna subcláusula propia que pueda ser etiquetada de conflicto $\{c_1, ..., c_{k'}\}$.

Dicho esto, la obtención del diagnóstico minimal se obtiene del teorema que afirma que *entendiendo Π como el conjunto de conflictos minimales asociados a un sistema (DS,COMPS,OBS) y FAULTY ⊆ COMPS, entonces D(FAULTY\COMPS) es una diagnosis si y solo si Π U D(FAULTY\COMPS) es consistente.*

La DBC se lleva a cabo por medio de un proceso iterativo de cuatro etapas clave:

- Predecir el comportamiento del sistema.
- Detectar conflictos en el sistema.
- Generar y organizar los candidatos a fallo.
- Refinar sucesivamente el diagnóstico inicial mediante medidas adicionales.

2.5. Conclusiones

Se define un fallo como la desviación del comportamiento real de un sistema con su comportamiento esperado. La diagnosis plantea soluciones al problema de detección de fallos en sistemas, para su posterior corrección.

Se trata de un problema de alta complejidad, dada la variedad de fallos que se pueden presentar en un sistema, y las posibles vías de las que se dispone para detectar los mismos. Esto ha motivado que a lo largo de las últimas décadas hayan proliferado diversas vertientes en el ámbito de la investigación destinadas al estudio y solución de la diagnosis de sistemas.

La Diagnosis Basada en Modelos es una de estas vertientes. Trabaja con modelos abstractos de sistemas para poder emular el comportamiento real esperado de un cierto sistema, y detectar anomalías del sistema real que representa.

Nuestra técnica de diagnóstico basada en PCs se encuadra dentro de esta línea de estudio, pero más particularmente dentro de las técnicas de Diagnosis Basadas en Consistencia, una particularización de la DBM. En la DBC los conjuntos conflicto, o conjuntos de componentes sospechosos asociados a un fallo, juegan un rol fundamental.

Mediante estos conjuntos la DBC busca definir una diagnosis minimal, entendida como un conjunto minimal de componentes del sistema susceptibles de sufrir un fallo. Esto se consigue mediante la detección de “síntomas” de fallo entre comportamiento esperado de componentes y observaciones; y la posterior generación de los conjuntos minimales.

Todo ello asienta las bases de la diagnosis basada en PCs. En el siguiente capítulo veremos en qué consiste esta técnica, como calculamos los conjuntos conflicto minimales y como se obtiene a partir de los mismos una diagnosis correcta del sistema.

Capítulo 3:

Diagnóstico Basado en Consistencia utilizando Posibles Conflictos

3.1. Introducción

En el capítulo anterior se introdujo el concepto de DBM y se describió todo el contexto teórico que rodea a la DBC empleando Conflictos. Con esta información el lector dispone de los conocimientos necesarios para comprender la técnica de diagnosis basada en PCs.

En el siguiente capítulo revisamos a fondo los fundamentos teóricos de dicha técnica y describimos las diferentes herramientas y conceptos teóricos requeridos por el investigador para ser capaz de tratar con esta serie de diagnósticos.

3.2. Caso de Estudio

3.2.1. Sistema de Tres Tanques

Para facilitar la comprensión de los conceptos teóricos introducidos en este capítulo se va a presentar un sistema de tres tanques como ejemplo de caso de estudio sobre el que aplicar las técnicas de diagnóstico descritas a continuación.

El sistema de tres tanques se define como una interconexión de tanques de masa entre los cuales circula un flujo de masa entrante. Este sistema ha sido objeto de estudio en [Moya2010] y [Hernández2012].

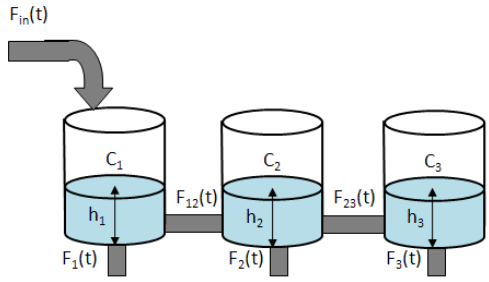


Figura 3.1 Sistema de tres tanques conectados

Describiendo brevemente el sistema, podemos ver como éste se compone básicamente de tres tanques de masa interconectados. Un flujo de masa entrante, F_{in} , introduce el líquido en el sistema, que comienza a circular por el primer tanque.

El flujo entrante en cada tanque se reparte en dos flujos: uno de salida hacia el exterior, y otro que alimenta al siguiente tanque del sistema. El sistema tiene tres observaciones: el flujo de salida de la parte inferior del primer tanque (F_1), el flujo entre los tanques primero y segundo (F_{12}) y el flujo de salida del tercer tanque (F_3).

El funcionamiento del sistema se rige por el siguiente conjunto de leyes físicas:

$$
\begin{aligned}
&1) && \dot{h}_1(t) = F_{in}(t) - F_{12}(t) - F_1(t)\\
&2) && \dot{h}_2(t) = F_{12}(t) - F_{23}(t) - F_2(t)\\
&3) && \dot{h}_3(t) = F_{23}(t) - F_3(t)\\
&4) && F_{in}(t) = input\ flow\\
&5) && F_1(t) = R_1 \cdot \sqrt{h_1(t)}\\
&6) && F_{12}(t) = R_{12} \cdot sign(h_1(t) - h_2(t)) \cdot \sqrt{|h_1(t) - h_2(t)|}\\
&7) && F_2(t) = R_2 \cdot \sqrt{h_2(t)}\\
&8) && F_{23}(t) = R_{23} \cdot sign(h_2(t) - h_3(t)) \cdot \sqrt{|h_2(t) - h_3(t)|}\\
&9) && F_3(t) = R_3 \cdot \sqrt{h_3(t)}\\
&10) && \dot{h}_1(t) = dh_1(t)/dt\\
&11) && \dot{h}_2(t) = dh_2(t)/dt\\
&12) && \dot{h}_3(t) = dh_3(t)/dt
\end{aligned}
$$

Las ecuaciones 1, 2 y 3 modelan el balance de masa de cada tanque. La ecuación 4 representa el flujo de entrada como variable de entrada del sistema (conocida). Las ecuaciones 5, 6, 7, 8 y 9 permiten la estimación de los flujos de salida de cada tanque, y los de intercambio de masa entre los mismos. Las ecuaciones 10, 11 y 12 son derivadas de las variables de altura de masa en tanque $h_1(t)$, $h_2(t)$ y $h_3(t)$.

Las ecuaciones 5, 6 y 9 hacen referencia a la estimación de variables observables disponibles en el modelo observacional del sistema.

Existen dos tipos de ecuaciones: ecuaciones algebraicas y ecuaciones diferenciales. En nuestro modelo las ecuaciones 4, 5 ,6 ,7 ,8 y 9 son algebraicas (instantáneas). Este tipo de ecuaciones permiten estimar la variable solución a partir de valores de variables de sistema correspondientes a un mismo instante de tiempo.

Las ecuaciones 10, 11 y 12 son ecuaciones diferenciales, y requieren, para la estimación de su variable solución, la aplicación de un paso de integración. Mediante dicho paso obtenemos la estimación de la variable solución en un tiempo *t+1* a partir de valores correspondientes al instante *t*. Existen diversos métodos de paso de integración, ya sea el método de Euler o Runge-Kutta entre otros.

3.3. Posibles Conflictos

3.3.1. Definición de Posibles Conflictos

A nivel teórico, un PC se define como un conjunto de ecuaciones con redundancia analítica mínima. Se define la DBC usando PCs como una técnica de compilación estructural que facilita la DBC en sistemas dinámicos de manera online [Pulido&Alonso2004].

3.3.2. Generación de Posibles Conflictos

Una de las ventajas de los PCs es que pueden ser calculados fuera de línea. Esto es, que previo inicio del proceso de diagnosis, podemos disponer de los PCs asociados a todos los modos de fallo.

La generación de PC sigue un proceso esquemático dividido en tres pasos:

- Generación de una representación abstracta del sistema como hipergrafo.
- Búsqueda de subsistemas de relaciones sobredeterminados (búsqueda de Cadenas Evaluables Minimales, CEMs).
- Obtención de todas las vías posibles de evaluación de los subsistemas (obtención de los Modelos Evaluables Minimales, MEMs).

Estructuralmente los hipergrafos se definen por nodos y arcos. Siguiendo la notación, los nodos representan variables de sistema y los arcos asociaciones entre variables (ecuaciones). Los arcos inter-slice se emplean para la representación de dependencias entre variables asociadas a un mismo instante de tiempo. Éstos se representan como "líneas punteadas", sin ecuación asociada, que marcan el paso de integración en la estimación de una variable de estado.

Así mismo, para la representación visual del hipergrafo se emplea notación propia de los grafos Y-O para indicar el conjunto de variables dependientes para la estimación de una cierta variable solución en una ecuación.

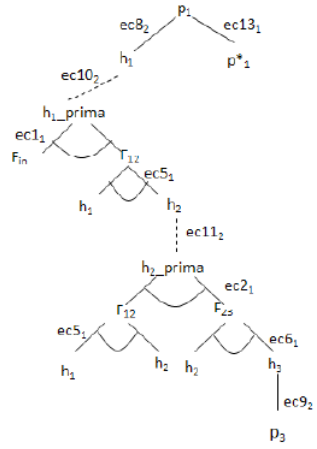


Figura 3.2 Ejemplo de hipergrafo para el modelado de un sistema de tres tanques

En los siguientes apartados procederemos a detallar punto a punto cada una de estas etapas.

- *Generación de una representación abstracta del sistema:*

  Consiste en realizar una abstracción del modelo de sistema a diagnosticar de modo que representemos el mismo como un hipergrafo $H_{SD} = \{V,R\}$:

  - $V = \{v_1,v_2, \ldots,v_n\}$ son las variables del sistema, observadas y no observadas. Se puede así definir también el conjunto V como $V = OBS \cup NOBS$. OBS y NOBS son conjuntos disjuntos.

  - $R = \{r_1,r_2, \ldots,r_n\}$ representa las restricciones entre variables de sistema (las relaciones, entendidas a modo de ecuaciones, que definen el modelo de sistema). Los r_i se entienden como subconjuntos de V*.

- *Búsqueda de Cadenas Evaluables Minimales:*

  Búsqueda del conjunto de subsistemas sobredeterminados asociados a la representación abstracta del sistema. Consiste en la búsqueda de todos los subhipergrafos parciales del sistema que sean capaces de realizar una doble estimación de una variable no observada o una estimación de una variable observada.

  Según [Bregón2010], un Cadena Evaluable H_{ce} $C= H_{SD}$ con $H_{ce} = \{V_{ce},R_{ce}\}$, $V_{ce}C=V$, $R_{ce}C=R$, $X_{ce} = V_{ce} \cap NOBS$ es el conjunto de variables no medidas y verificando:

  1. H_{ce} es conexo,
  2. $V_{ce} \cap OBS \neq 0$
  3. $\forall v_{no} \in X_{ce} \rightarrow d_{Hec}(v_{no}) \geq 2$
  4. Si G(Hce) es un grafo bipartito cuyos nodos corresponden por un lado a $x \in X_{ce}$ y por otro lado a $r_{ice} \in R_{ce}$, donde los nodos están unidos por un arco sii $x \in r_{ice}$,entonces $G(H_{ce})$ tiene un *matching* de máxima cardinalidad $m'=|X_{ce}|$ y $|R_{ce}| \geq m' + 1$.

  El matching de un grafo $G = [V,E]$ es un subconjunto de arcos de E donde cada vértice es adyacente, a lo sumo, a un arco.

  Una Cadena Evaluable es un subsistema conexo, con observaciones, donde puede realizarse propagación local y que define un conjunto de restricciones sobredeterminado.

  Se dice que una Cadena Evaluable es minimal si no existe un $H_{ce}'/ H_{ce}'C H_{ce}$ y H_{ce}' es cadena evaluable.

- *Obtención de los Modelos Evaluables Minimales:*

  Consiste en añadir información causal a las CEM. Esta modificación permite garantizar que una CEM puede resolverse por medio de criterios de propagación local.

Se define un MEM como un hipergrafo $H_{mem} = \{V_{mem}, R_{mem}\}$, con V_{mem},= V_{ce} y R_{mem}={ $r1_{k1}$,$r2_{k2}$, . . .,rm_{km}}; donde cada restricción ri_{ki} representa una causalidad concreta asiganada a $r_i \in R_{ce}$. Al conjunto de relaciones de un MEM se le denomina Posible Conflicto.

Para la obtención de PCs a través de CEM se dispone de algoritmos que se encargan de inferir los MEMs asociados a cada CEM. Dado un PC, un MEM representa un modelo computacional implementable. En el pasado se ha utilizado simulación [PULIDO 2001] u observadores [BREGON 2010], o la combinación de ambos. En este caso, nos interesa además de la simulación la generalización de filtros u observadores de estado que nos proporcionan las Redes Bayesianas Dinámicas.

En nuestro caso de estudio existen un total de tres MEMs, asociados a cada uno de los tres PCs del sistema:

PC	Ecuaciones	Variables de Entrada	Variable Estimada
PC1	1,5,10	F_{in}	F_1
PC2	2,6,11	F_{in}	F_{12}
PC3	3,9,12		F_3

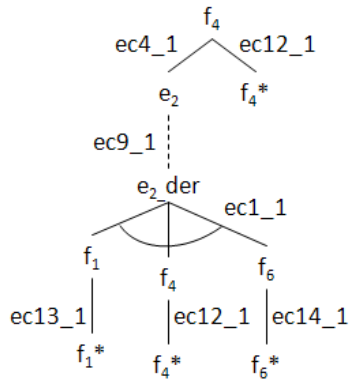


Figura 3.3 Posible Conflicto 1 (PC1) del Sistema de Tres Tanques

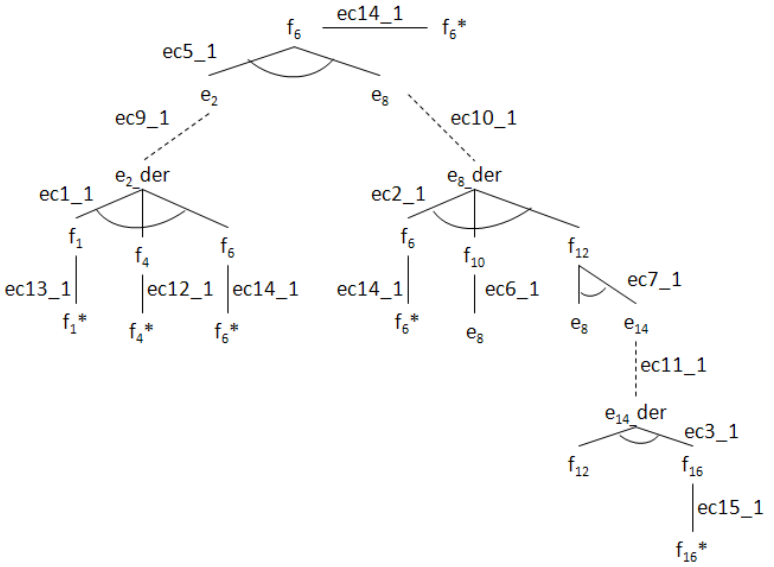


Figura 3.4 Posible Conflicto 2 (PC2) del Sistema de Tres Tanques

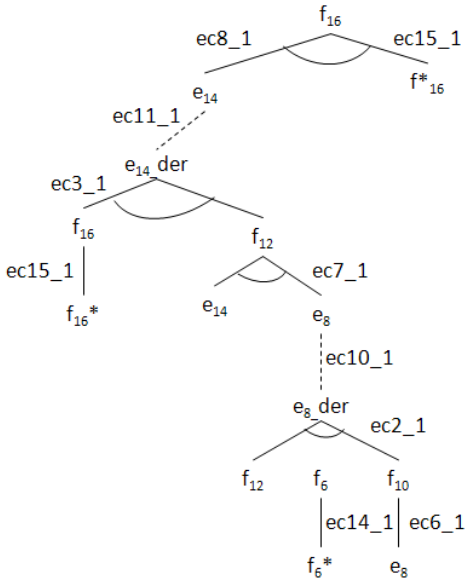


Figura 3.5 Posible Conflicto 3 (PC3) del Sistema de Tres Tanques

3.4. Redes Bayesianas Dinámicas

3.4.1. Definición de Redes Bayesianas Dinámicas

La Red Bayesiana Dinámica (RBD) es una herramienta teórica empleada para la representación temporal de sistemas dinámicos. Su estructura deriva directamente del concepto de Red Bayesiana (RB) como elemento teórico empleado para la representación de modelos. Frente a la RB convencional, la RBD añade la posibilidad de modelado temporal de sistemas al contemplar un diseño teórico de los mismos dividido en dos instantes de tiempo.

Se define una Red Bayesiana Dinámica (RBD) como una red bayesiana con dos instantes de tiempo consecutivos conectados [KollerLerner2001].

El modelado de sistemas mediante RBD se hace atendiendo a la suposición de que la estructura del sistema es invariante en el tiempo y que se trata con un proceso de Markov de primer orden. Este proceso se define como un principio que se cumple cuando la probabilidad condicional de variables en un sistema depende únicamente de valores de variables asociados a un único instante de tiempo anterior.

En nuestro caso, este principio supone que todas las variables son condicionalmente independientes del resto, conocidos sus padres (asociados al instante de tiempo anterior).

A nivel de representación gráfica, una RBD es una red de nodos y arcos. Los nodos representan el conjunto de variables asociadas al sistema, en un instante de tiempo concreto. Existen diferentes tipos de variables, mediante las cuales modelamos el sistema completo: $D = (X, Z, U, Y)$.

- **X:** variables de estado. Conjunto de variables que definen el estado de un sistema.
- **Z:** no observables. Variables internas del sistema, que modelan parte del comportamiento del mismo. Se caracterizan por no ser accesibles al usuario, esto es, se desconoce de su valor en todo momento.
- **U:** entradas. Conjunto de variables que modelamos como entradas al sistema a diagnosticar, y que son conocidas por el usuario.
- **Y:** variables observables. Se caracterizan por ser accesibles al usuario, esto es, se conoce su valor en todo momento. Son vitales para el diagnóstico, pues gracias a ellas detectamos desviaciones en el modelo, que permiten detectar y localizar fallos.

El componente estructural restante de la RBD son los arcos. Empleamos los arcos para unir nodos. Mediante estos elementos se modelan las dependencias entre elementos del sistema y la transición entre instantes de tiempo en la descripción de la red.

Los arcos que emplean las RBD son dirigidos, y presentan dos posibles configuraciones: arcos inter-slice y arcos intra-slice. Los arcos inter-slice se emplean para la representación de dependencias entre variables asociadas a un mismo instante de tiempo. Los arcos intra-slice modelan relaciones entre variables relativas a instantes de tiempo distintos y consecutivos.

El uso de uno u otro tipo de arco en base a la asociación a realizar es siempre igual:

- Arcos inter-slice: empleados para la estimación de variables observables, no observables y derivadas de variables estado.

- Arcos intra-slice: asocian variables de estado con sus derivadas. Esta asociación se traduce a nivel analítico como una integración, lo que determina la propiedad particular de estos arcos para marcar el tránsito entre instantes de tiempo consecutivos.

A partir de la construcción de modelo de sistema por medio de una RBD, es posible efectuar la simulación del mismo. Para ello se requiere una cierta información de los valores de parámetros del sistema en los instantes de tiempo inicial. Esta restricción está motivada por las dependencias temporales entre instantes consecutivos que contempla la red.

A nivel visual, la RBD se corresponde con la definición de grafo dirigido con una distribución de probabilidad condiciona en cada nodo (acorde al Teorema de Bayés).

Empleamos las RBD para la inferencia probabilística del estado de un sistema en base al conocimiento de sus entradas y medidas (variables observables en el modelo). Sin embargo esta herramienta presenta la limitación de no poder realizar una inferencia exacta de la misma (no es computacionalmente factible). Ante semejante limitación se recurre a una técnica de inferencia aproximada conocida como Filtrado de Partículas. Este método se basa en el principio de efectuar una simulación del sistema siguiendo el método de Monte Carlo para efectuar una inferencia aproximada.

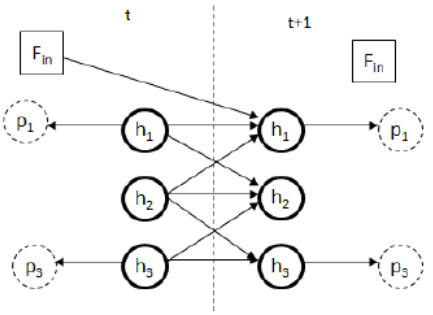


Figura 3.6 Ejemplo de Red Bayesiana Dinámica para el Sistema de Tres Tanques (RBD)

3.5. Filtro de Partículas

3.5.1. ¿Qué es un Filtro de Partículas?

El Filtro de Partículas (FP) es un método de Monte Carlo secuencial basado en representaciones de densidades de probabilidad como "partículas", aplicadas a cualquier espacio de estados de un cierto modelo, y con las que se generalizan los métodos tradicionales del Filtro de Kalman. Las partículas, o muestras, son posibles estados del sistema, representadas como puntos en el espacio de estados del mismo.

Empleamos el FP para obtener una estimación aproximada del estado del sistema por medio de un conjunto de partículas o muestras ponderadas con un peso, los cuales son conjuntos de valores asociados a cada nodo de la red, determinados en función de los nodos observables.

3.5.2. Introducción al ámbito teórico del Filtro de Partículas

En los problemas de estimación del estado del sistema la solución reside en construir la función de densidad de probabilidad (fdp) de estados en base a la información de los modelos disponibles; la cual permite una estimación del estado del sistema, y una medición de la precisión en la estimación.

Para estos casos, la solución óptima consiste en emplear filtros recursivos. Los filtros recursivos actúan en dos fases: predicción y actualización. La predicción emplea el modelo de sistema para predecir la siguiente fdp de estado a partir de las medidas. En la actualización modificamos la fdp, sujeta generalmente a ruido desconocido, por medio del teorema de Bayés, para contemplar la información basada en nuevas mediciones del siguiente instante de tiempo. Esto permite que el filtro procese información de forma secuencial a modo de ventana temporal.

Este problema de estimación se define como "problema de seguimiento". Desde una perspectiva Bayesiana, consiste en calcular recursivamente el grado de confianza de la estimación del estado "*x*" en el instante *k*, tomando diferentes valores de observaciones "*z*", dados hasta el instante *k*. Así primero construimos la fdp con medidas del instante *k-1*; y luego actualizamos la misma con las medidas *z* del instante *k*, en base al teorema de Bayés. La relación de recurrencia entre estados define la solución bayesiana óptima al seguimiento.

Existen algoritmos óptimos para la resolución del problema de seguimiento. Sin embargo, éstos presentan ciertas restricciones que impiden su uso en ciertas situaciones en las que el problema puede tener lugar. A continuación procedemos a describir los dos más conocidos [Arulampalam2002]:

- *Filtro de Kalman:*

  Asume que la fdp en todo instante es gaussiana, por lo que es parametrizable a través de la media y la covarianza. Se definen las funciones de estimación de

estados y medidas del sistema (los modelos) como matrices que definen funciones lineares.

El filtro calcula una serie de covarianzas relativas a los vectores del ruido del proceso y las medidas para la estimación de las fdps. Su solución es óptima.

- *Métodos basados en Redes de Estados:*

Sus suposiciones contemplan un espacio de estados discreto y con un número finito de estados. Para cada periodo de tiempo se dispone de una serie de estados discretos "x^i". Cada uno de ellos tiene asignado un peso. La definición de fdp posterior se formula como:

$$p(x_k|z_{1:k-1}) = \sum_{i=1}^{N_s} w_{k|k-1}^i \delta(x_k - x_k^i)$$

$$p(x_k|z_{1:k}) = \sum_{i=1}^{N_s} w_{k|k}^i \delta(x_k - x_k^i)$$

En la fase de actualización se realiza una estimación de pesos actualizados en base a las nuevas medidas.

Sin embargo es común que en las situaciones que se le presenten al investigador, las suposiciones planteadas por los algoritmos óptimos mencionados no se sostengan, y se debe recurrir a aproximaciones de estos algoritmos.

Existen diversas aproximaciones, como el "Filtro de Kalman Extendido", que permite tratar con funciones de actualización y muestreo no lineales por medio de una linearización de las mismas en base al primer término de la expresión de Taylor; o los "métodos basados en Redes de Estados Aproximados", que aborda sistemas con un espacio de estados continuo, el cual discretiza dividiéndolo en una serie de muestras, cuyos "centros" son empleados para la estimación del estado del sistema (lo cual exige de considerar un número de muestras considerable para dar con una buena aproximación, lo que implica un serio incremento del coste computacional del algoritmo).

Dentro de estas aproximaciones se encuadra el concepto de Filtro de Partículas.

3.5.3. Filtro de Partículas y Algoritmo de Importancia de Muestreo Secuencial

Existen diferentes algoritmos de Filtro de Partículas, sin embargo, el que nos atañe es el "Algoritmo de Importancia de Muestreo Secuencial" (algoritmo SIS).

El algoritmo SIS es una implementación de un FP, y por lo tanto se define como un método de Monte Carlo. El algoritmo busca representar la fdp por medio de muestras aleatorias con pesos asociados, con los que efectuar las estimaciones de estado. Un mayor número de muestras supone la aproximación del filtro a la

estimación bayesiana óptima. La estimación de la fdp actual se hace por medio de la ecuación:

$$p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i)$$

La elección de pesos para cada muestra se hace según el principio de "importance sampling", de ahí su nombre.

El algoritmo SIS se resume en una serie de pasos, donde para cada muestra se obtiene su actualización al instante actual por la función de "densidad de importancia"; y se le asigna un nuevo peso acorde a la expresión del algoritmo.

Su uso presenta el problema de degeneración, el cual supone que, después de una serie de iteraciones del algoritmo, todas las partículas menos una acaban teniendo un peso despreciable. Determinado que la variación de la importancia de pesos solo puede crecer en el tiempo, la degeneración es un fenómeno inevitable. La degeneración conlleva que el algoritmo arrastra en cada instante de tiempo muestras que no son útiles para la estimación de la fdp posterior. Esto supone un coste computacional elevado invertido en la actualización de partículas no significativas, una tarea desechable.

El grado de degeneración es estimable, determinado en número de partículas disponibles; donde tamaños pequeños indican una degeneración severa. Existen técnicas para paliar este problema:

- ✓ *Buena elección de la densidad de importancia:* selección de una función de asignación de pesos que minimice la variabilidad de pesos entre muestras en un instante de tiempo, para que la medida de degeneración sea máxima.

- ✓ *Remuestreo:* consiste en la generación de nuevas muestras cuando detectamos signos de degeneración claros. Eliminamos las partículas con pesos no significativos y nos concentramos en las de pesos mayores. Esto supone generar un nuevo conjunto de muestras y reemplazarlos. Este método introduce nuevos problemas como el de limitar la paralelización del proceso del FP debido a que se requiere de la combinación de todas las partículas para su uso.

  Además, las partículas de pesos altos son seleccionadas varias veces, lo que supone una pérdida de diversidad de partículas (empobrecimiento de partículas), de manera que todas tienden a colapsarse en un único punto del espacio de estados a las pocas iteraciones.

- ✓ *Técnicas para eludir el uso de una función de densidad de importancia subóptima:* para afrontar casos en que una función de densidad apropiada no está disponible; métodos para fomentar que las partículas estén en el lugar adecuado, de manera que se las puedan asignar pesos acordes a la distribución y remuestreo.

Frente a otras aproximaciones de algoritmos óptimos para dar una solución al "problema de seguimiento", el algoritmo SIS presenta una serie de ventajas particulares que motivan su uso frente a otros algoritmos.

La formulación del filtro no presenta restricciones que le impidan ser aplicado a ningún modelo. Su uso ofrece una aproximación global y exacta a la solución conforme el número de muestras se aproxima al infinito.

Sin embargo, el algoritmo también tiene sus contrapartidas. Su uso supone un esfuerzo computacional elevado en contraste a otras alternativas como el FK. Además presenta problemas de precisión cuando trabaja con modelos libres de ruido, especialmente en modelos dinámicos. A nivel de implementación, también presenta ciertas dificultades a la hora de depurar así como implementar de forma eficiente y con una función de densidad apropiada.

3.5.4. Aplicación de un Filtro de Partículas Genérico

A continuación vamos a describir el algoritmo SIS paso a paso y dar una interpretación del mismo. En este apartado exponemos el algoritmo genérico que implementa un filtro SIS aplicando el método de remuestreo.

La idea es ofrecer en una notación comprensible para el lector una explicación práctica del algoritmo, que ponga de manifiesto la aplicación de las ideas expuestas en la definición del algoritmo SIS y, a su vez, explique el porqué de su forma de ser.

El FP SIS es un algoritmo empleado para la propagación y actualización de un conjunto de partículas entre dos instantes de tiempo, que llevan asociadas un peso. El algoritmo actualiza las partículas entre dos instantes de tiempo, modificando sus pesos. De esta menara se representa la fdp del nuevo instante de tiempo.

El algoritmo se divide en dos fases:

- *Predicción:* cada muestra se pasa por el modelo de sistema para obtener nuevas partículas asociadas al nuevo instante de tiempo.
- *Actualización:* tiene lugar cuando se recibe información de las observaciones del sistema en el instante actual.

Algoritmo 1: Filtro de Partículas SIS

Descripción: invocación del algoritmo SIS genérico. Explica la aplicación del algoritmo SIS para la transición entre dos instantes de tiempo. Partimos del conjunto de Ns muestras de las variables estado de nuestro sistema con sus pesos asociados, $\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_s}$; y las observaciones del sistema en el instante actual, z_k.

Se obtiene un nuevo conjunto de muestras actualizadas al instante de peso actual, con una nueva asignación de pesos.

Pseudocódigo:

$\left[\{\mathbf{x_k^i}, \mathbf{w_k^i}\}_{\mathbf{i=1}}^{\mathbf{N_s}}\right] = \mathbf{FP_{ISIS}}[\{\mathbf{x_{k-1}^i}, \mathbf{w_{k-1}^i}\}_{\mathbf{i=1}}^{\mathbf{N_s}}, \mathbf{z_k}]$

1. FOR $i = 1{:}\,N_s$
   - Generar muestra $x_k^i \sim q(x_{k-1}^i | x_{k-1}^i, z_k)$
   - Asignar a cada muestra su peso, w_k^i
2. END FOR
3. Calcular el peso total: $t = \text{SUM}[\{w_k^i\}_{i=1}^{N_s}]$
4. FOR $i = 1{:}\,N_s$
   - Normalizar peso de cada muestra: $w_k^i = w_k^i \cdot t^{-1}$
5. END FOR
6. Calcular $\widehat{N_{eff}}$
7. IF $\widehat{N_{eff}} < N_T$
   - Efectuar remuestreo: $\left[\{x_k^i, w_k^i\}_{i=1}^{N_s}\right] = \text{RESAMPLE}\left[\{x_k^i, w_k^i\}_{i=1}^{N_s}\right]$
8. END IF

Por medio del conjunto de partículas actualizadas es posible obtener una estimación de la fdp según la expresión:

$$p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i)$$

La asignación de pesos se hace en función del principio de "Importancia de Muestreo". Se define la asignación de pesos a muestras como el cociente de dos funciones. Por un lado tenemos la función π(x), aproximación de p(x). Consideramos que p(x) es una fdp de la cual es difícil extraer muestras, pero para la cual π(x) puede ser evaluada. Por otro lado tenemos q(x), función que permite la generación de muestras x_k . La expresión de la asignación de pesos a muestras resulta ser:

$$w^i \propto \frac{\pi(x^i)}{q(x^i)}$$

Por lo general, la función q(x) viene a ser la función de "densidad de importancia" de transición de instantes, la cual se sustituye en la expresión de los pesos para obtener la propagación de los mismos entre instantes de tiempo:

$$q(x_k|x_{k-1}^i, x_k) = p(x_k|x_{k-1}^i)\, w_k^i \propto w_{k-1}^i p(z_k|x_k^i)$$

Después de la etapa de actualización, en cada iteración del algoritmo, es común efectuar un paso de "remuestreo". El remuestreo es un método que permite reducir el "efecto de degeneración" del filtro. Mediante la aplicación del mismo eliminamos las muestras de poco peso.

El remuestreo basa su funcionamiento en realizar un muestreo con reemplazo sobre las partículas actuales considerando los pesos; que acaban por actualizarse asignando el mismo a cada una de las nuevas partículas. Así se logra tener una

muestra cuyas partículas tienen mismo peso, pero que conservan la información de la muestra anterior.

Es posible efectuar el remuestreo en cada iteración del algoritmo, sin embargo esto puede ser innecesario debido a que puede haber situaciones en las que no resulte adecuado efectuar este paso.

Para evitar estos casos se define el concepto de "tasa de efectividad" de las partículas:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2}$$

Empleamos la tasa de efectividad para medir cuando la degeneración del algoritmo es considerable. Estableciendo un valor de degeneración umbral N_T se comprueba si

la tasa de efectividad está por debajo del umbral. En dicho caso se efectúa el paso de remuestreo. Esto viene reflejado en el paso 7. del algoritmo de FP ISIS.

Algoritmo 2: Algoritmo de Remuestreo

Descripción: explica los pasos a seguir para efectuar el remuestreo de las partículas del sistema.

Pseudocódigo:

$$\left[\{\mathbf{x_k^i}, \mathbf{w_k^i}, \mathbf{i^j}\}_{\mathbf{i=1}}^{\mathbf{N_s}}\right] = \mathbf{RESAMPLE}[\{\mathbf{x_{k-1}^i}, \mathbf{w_{k-1}^i}\}_{\mathbf{i=1}}^{\mathbf{N_s}}, \mathbf{z_k}]$$

1. Inicializar el CFD: $C_1 = 0$
2. FOR i $= 2$: N_s
   - Construir el CDF: $c_i = c_{i-1} + \mathrm{w_k^i}$
3. END FOR
4. Inicializar índice al inicio del CDF: i $= 1$
5. Dibujar el punto de inicio siguiendo una distribución normal: $u_1 = U[0, N_s^{-1}]$
6. FOR i $= 1$: N_s
   - Nos desplazamos a lo largo del CDF: $u_j = u_1 + N_s^{-1}(j-1)$
   - WHILE $u_j > c_i$
     - $i = i + 1$
   - END WHILE
   - Asignar muestra: $\mathrm{x_k^{j*}} = \mathrm{x_k^i}$
   - Asignar peso: $\mathrm{w_k^j} = N_s^{-1}$
   - Asignar padre: $i^j = i$
7. END FOR

3.6. Diagnosis mediante Redes Bayesianas Dinámicas

En la técnica de diagnosis basada en PCs empleamos la RBD para la estimación del estado del sistema a analizar y la detección de fallos.

Recordando lo anteriormente expuesto en torno a la filosofía DX, el uso de las RBD en base a los principios de esta rama de técnicas de diagnóstico comprendería el uso de una serie de RBD: una red nominal que modelaría el comportamiento correcto del sistema; y una serie de redes de fallos, que supondrían emular el sistema con una determinada hipótesis de fallo.

Con estos elementos el proceso de detección de fallo se efectuaría por medio de la simulación de la red nominal y el contraste de los resultados de la estimación con los datos observables del modelo. De la obtención de valores se efectuaría un cálculo de residuos indicadores de la presencia de fallo en el sistema.

Para la localización del fallo, una vez detectado, empleamos los modelos de fallo del sistema. En este caso lo que se busca es que las divergencias entre estimaciones y observación sean nulas, determinando el tipo de fallo modelado que se ajusta al comportamiento actual del sistema. Se trata de un procedimiento de descarte de modos de simulación de fallo.

Este método de empleo de RBD proporciona ciertas ventajas al investigador. Entre ellas cabe destacar su capacidad para contemplar un cierto grado de incertidumbre en los resultados de diagnosis.

Por otro lado, su empleo supone un coste computacional elevado proporcional a la complejidad del modelo de sistema. Dicho coste viene motivado en parte por el hecho de que el modelado de los posibles modos de fallo de un sistema conlleva la simulación del sistema completo por cada fallo contemplado.

Esto supone un problema crítico, dado que la diagnosis de un sistema real requiere una respuesta en tiempo real frente a los resultados observados en cada instante, y el aumento de la carga computacional de simulación de modelos puede dar lugar a retardos en el procesamiento que impedirían abordar una solución factible del problema.

Ante semejante situación resulta necesario buscar una solución que rebaje la carga computacional de simulación. Para dar respuesta a este problema se recurre a la técnica de los Posibles Conflictos.

El empleo de PCs permite rebajar la carga computacional de simulación de modelos, haciendo factible la diagnosis de sistemas en base al uso de modelos basados en RBD. Esto se debe a que, como describiremos en detalle a continuación, los PCs se obtienen de la factorización del sistema, simplificando las RBD que modelan fallos.

3.7. Conclusiones

La técnica de diagnosis basada en Posibles Conflictos trabaja con el concepto teórico de Posible Conflicto para la diagnosis de sistemas dinámicos. Un PC se define como un conjunto de ecuaciones con redundancia analítica mínima.

La obtención de PCs es un proceso *offline* que requiere de una serie de pasos clave: generación de una representación abstracta del sistema, búsqueda de las Cadenas Evaluables Minimales y obtención de los Modelos Evaluables Minimales. Los PCs modelan subsistemas derivados del modelo de sistema completo, estimando el valor de parámetros del sistema susceptibles de fallo.

La RBD se define como la unión de dos Redes Bayesianas que modelan instantes de tiempo diferentes y consecutivos. Se trata de un grafo donde los nodos representan las variables del sistema y las asociaciones las restricciones del mismo.

Una vez se dispone de las RBD asociadas a los PCs estamos en condiciones de simular el modelo; para ello se recurre al Filtro de Partículas, una herramienta que permite la estimación del estado interno del sistema contemplando el factor de ruido en los sensores.

La simulación del sistema genera valores observables del mismo estimados por el modelo, en lo que se correspondería con el comportamiento correcto del sistema.

Llegados a este punto, todos los aspectos teóricos a exponer ya han sido descritos. En el siguiente capítulo pasamos a profundizar en el estudio de la estructura interna del sistema AHU-9, su modelado y la generación de los PCs asociados al mismo.

Una vez superado este capítulo se considerará que el lector dispone de todos los conocimientos teóricos necesarios para la comprensión de nuestra labor, y se procederá a documentar el desarrollo de la labor de diagnosis sobre el sistema AHU-9.

Capítulo 4:

Modelo de Caso de Estudio

4.1. Introducción

En el siguiente capítulo procedemos a realizar un estudio completo del sistema Air-Handling-Unit Nine (AHU-9). El estudio abarca todos los aspectos asociados al sistema, lo que contempla tanto el estudio de las especificaciones del sistema real, como las especificaciones ofrecidas por la organización DXC de los diferentes modelos software del sistema.

Adicionalmente explicaremos aspectos técnicos corregidos y derivados del análisis de los modelos software puestos a disposición de la organización y revisados; y también se revisarán los modelos simplificados del sistema diseñados y propuestos por nosotros.

4.2. Sistema Real AHU-9

4.2.1. Introducción

En el siguiente bloque se hace una revisión rápida de los componentes que definen el sistema AHU-9. Describiremos su funcionalidad brevemente y asociaremos las descripciones a las figuras del modelo gráfico. Esto facilitará su comprensión a la hora de abordar el sistema a un nivel de detalle más complejo.

4.2.2. Descripción del sistema AHU-9

El sistema AHU-9 representa el sistema de un controlador de aire instalado en la Escuela de Música de Cork. Su funcionamiento se resume en la manipulación del aire externo y del interior de las habitaciones del complejo para aclimatar la temperatura de las mismas.

El modelo de Sistema en cuestión es el que se representa en la siguiente figura:

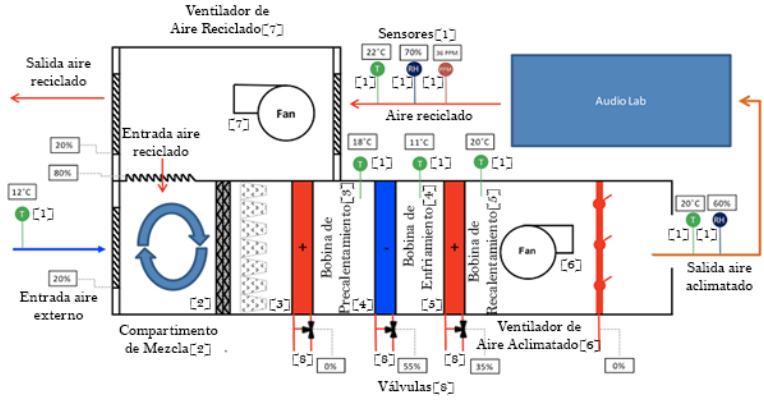


Figura 4.1 Sistema Real AHU-9

El sistema funciona a modo de bucle continuo. El aire externo entra, recorriendo todo el circuito que se describe, y sale al exterior (o bien se recicla dentro del propio bucle de aclimatamiento). El comportamiento del sistema puede ser descrito de modo secuencial:

- *Un ventilador de aire introduce el aire del exterior. Se configuran unos sensores para la medición de temperatura, humedad y nivel de CO2.*

- *Una cámara "mezcladora de aire" controla la proporción de aire interno y externo que se introduce en el sistema.*

- *Para alcanzar los niveles de temperatura y humedad adecuados el aire se hace pasar por una cámara de tres componentes: un precalentador de aire, un refrigerador y un recalentador.*

- *Una vez el aire atraviesa el recorrido, un extractor conduce el aire al interior de las salas.*

El proceso descrito se implementa a través de diversos componentes. Resulta oportuno mencionarlos uno a uno, describiendo brevemente su funcionalidad básica:

- ✓ Sensores [1]: se definen tres sensores para la temperatura, humedad y CO_2.

- ✓ Compartimento de mezcla [2]: cámara que mezcla el aire del sistema con el del exterior. El sistema está gobernado por amortiguadores que controlan la proporción entre aire del interior y el exterior. Se supone que una mezcla perfecta es aquella que logra las condiciones de CO_2, temperatura y humedad deseadas.

- ✓ Bobina de Precalentamiento [3]: primer paso del tratamiento de aire externo. Destinado a evitar la congelación del aire extremadamente frío, procedente del exterior, en el interior de la cámara de mezcla.

- ✓ Bobina de Enfriamiento [4]: permite enfriar el aire a la temperatura adecuada. Además, implementa el proceso de humidificación. Mediante él se reduce el nivel de humedad relativa del aire, por condensación del vapor de agua sobre-enfriado.

- ✓ Bobina de Recalentamiento [5]: una vez se enfría el aire, este bucle de recalentamiento se instala para reajustar la temperatura del aire al nivel deseado.
- ✓ Válvula hidráulica [8]: el sistema presenta tres válvulas hidráulicas instaladas en cada uno de los bucles del sistema. Controlan el flujo de agua que circula por cada cámara, el cual se emplea como refrigerante.

- ✓ Bombas hidráulicas [-]: tres bombas que ponen en circulación el líquido refrigerante en cada cámara del sistema (asociadas a cada uno de los bucles mencionados).

- ✓ Calentador de agua [-]: el líquido calefactor, agua caliente, viene proporcionado por una central calorífica que funciona con gas.

- ✓ Refrigerador [-]: proporciona el líquido refrigerante al sistema.
- ✓ Ventiladores [6,7]: como se mencionó anteriormente se dispone de dos ventiladores. El ventilador de entrada que introduce aire del exterior al AHU, y el ventilador de salida, que insufla el aire a temperatura deseada dentro de la sala.

Con todo ello puede verse que el sistema no tiene un funcionamiento extremadamente complejo con una descripción de alto nivel.

En el siguiente bloque vamos a revisar los modelos de componentes, uno a uno, junto con sus ecuaciones. La idea es extraer una interpretación del sistema lo más próxima posible a nuestros intereses; esto es, efectuar un diagnóstico del sistema AHU-9 empleando la técnica de Diagnosis Basada en Posibles Conflictos.

4.3. Modelica:

4.3.1. ¿Qué es Modelica?

Modelica es un lenguaje de programación orientado a objetos y declarativo; diseñado específicamente para el modelado de sistemas complejos.

Modelica difiere del resto de lenguajes de programación en la característica particular de que su lenguaje puede ser etiquetado de "modelado" más que de programación en sí.

Las clases Modelica no son compiladas en el sentido tradicional de la programación orientada a objetos, sino que su "compilación" provoca que las clases se traduzcan a objetos que se ejecutan mediante un motor de simulación.

Las clases se componen de atributos, algoritmos y ecuaciones. Las ecuaciones no describen una asignación de valores a variables, sino igualdades. En términos de Modelica, lo que vendría a decir esto es que no existe una causalidad predefinida para las ecuaciones.

Los algoritmos permiten predefinir acciones de inicialización de variables de entrada y salida previas a la ejecución del motor de simulación.

El motor de simulación trabaja con las ecuaciones de manera simbólica para determinar el orden de ejecución de las mismas, reconociendo los componentes de la ecuación que son entradas y salidas.

4.3.2. Modelado de sistemas en Modelica

Las clases Modelica difieren a nivel interno de la estructura de clase habitual, descrita por atributos y funciones. Las clases Modelica se componen de atributos, pero sustituyen las funciones por el conjunto de ecuaciones que define el comportamiento del componente que modelan.

Dentro de la estructura de las clases, también se permite definir una "representación" de las mismas. Modelica trabaja con un lenguaje propio de representación de componentes. Podemos adjuntar a las clases una determinada descripción visual, de manera que cuando un editor Modelica compatible con la funcionalidad que se describe trabaja con clases gráficamente definidas, el sistema es capaz de ofrecer una representación 2D del componente con el que se está tratando.

Como veremos en nuestro caso, el modelo complejo del sistema AHU-9 viene programado con una representación gráfica del mismo que facilita su comprensión en cierta medida.

El acceso a las ecuaciones es algo que solo puede realizar el motor de simulación, ya que cada clase del sistema, una vez instanciada, se comporta a modo de "caja negra"; la cual solo se comunica al exterior por variables de entrada y salida.

Mientras que en la programación tradicional las clases se comunican unas con otras a partir de la invocación a funciones, donde se pasan parámetros y se reciben resultados de ejecución, Modelica trabaja con variables de entrada y salida.

Cuando una clase se instancia, el motor de simulación efectúa una estimación de los valores de salida de cada función en base a los valores de entrada, de manera continua. Si una clase quiere comunicar información a otra (enviar o recibir), debe invocar el valor de los atributos de la misma. Esto se hace según la notación tradicional "*instancia.atributo*" que frecuentemente se emplea en todo lenguaje de programación orientado a objetos.

En resumen, para modelar un sistema en Modelica es necesario definir tantas clases como componentes definen el sistema con el que se trabaja. Cada componente debe contemplar una serie de variables de entrada y salida tales que permitan al mismo simular su comportamiento real.

El modelado del comportamiento se hace en base a las ecuaciones que definen al componente. Hablamos de ecuaciones como igualdades, sin una asignación de causalidad definida; y que son gestionadas por el motor de simulación exclusivamente.

Los componentes de un sistema deben interconectarse para dar lugar a la simulación del sistema completo. Esto se hace por medio de la invocación de los atributos de entrada y salida definidos en cada clase.

A nivel de código, un sistema Modelica se compone de una clase "main" y una biblioteca de clases, comúnmente entendida como "biblioteca de componentes".

Las bibliotecas de componentes no son más que una colección de clases que modelan cada uno de los componentes que definen el sistema. La clase "main" viene a ser una clase particular, que difiere del resto al no modelar ninguno tipo de componente.

La funcionalidad de la clase main es la de iniciar la construcción el modelo final del sistema e iniciar su simulación. Para ello, esta clase se implementa con un código que se resume en la instanciación de cada uno de los componentes del sistema y el establecimiento de las conexiones entre sí a través de sus atributos de entrada y salida.

Si se desea extraer resultados de simulación del modelo de sistema, la parte de algoritmos de esta clase se configura con los ficheros de ruta que se desean generar una vez se va a efectuar la simulación.

Cuando se dispone del modelo de sistema completo listo, la herramienta editora de ficheros ".mo" procede a su compilación inicial, y luego a su posterior simulación, previa especificación de los instantes de tiempo a emplear para la simulación y otros valores de interés, como los parámetros iniciales del experimento o el tipo de paso de integración.

4.3.3. Modelado de Sistemas AHU-9

Tanto el modelo de sistema AHU-9 como su versión simplificada han sido programados en Modelica. En los siguientes apartados del capítulo se procederá a una descripción detallada de los mismos.

Sin embargo, es necesario explicar en cierta medida como han sido programados los mismos para facilitar la comprensión del análisis. Aunque no se va a detallar ningún aspecto del código fuente del modelo, sí que hay que hacer referencia a ciertos aspectos funcionales de Modelica en los que el software se apoya para generar una representación del sistema AHU-9:

- *Bibliotecas:* el modelo de sistema complejo trabaja con una biblioteca de clases denominada "HVAC_IRUSE"; definiendo en un fichero aparte, "AHU_09_full.mo" la clase "main" que instancia el modelo completo del sistema y efectúa su simulación.

  El modelo simple del sistema trabaja con un único fichero que contempla la definición de todos los componentes del sistema, y la clase "main".

- *Puertos:* las conexiones entre componentes se realizan a través de atributos. Sin embargo, en sistemas complejos, el hecho de tener que conectar atributos de un componente con los de otro puede suponer una tarea tediosa si el componente dispone de un número elevado de atributos.

  Una solución a este problema es la de definir conectores o puertos. Los puertos se entienden como clases intermedias que modelan conexiones. No tienen una funcionalidad específica, aunque algunos se emplean para modelar conductos de ventilación o de flujos de masa, y pueden contemplar un cierto comportamiento interno para modelar dichos elementos.

Mediante los conectores, el programador ahorra trabajo al definir en los mismos las variables de entrada y salida que definen a un componente. Así se conectan las clases a través de atributos conectores declarados en las mismas, de manera que no hay que no es necesario declarar una conexión por cada atributo.

En nuestro caso particular, los conectares modelan válvulas de aire y masa. Hablamos de puertos neumáticos e hidráulicos, respectivamente. Los primeros se identifican con el prefijo "P", mientras que los restantes se marcan con la etiqueta "H".

En el siguiente apartado se detalla la estructura de cada una de estas conexiones y las diferentes instancias empleadas de los mismos para conectar cada componente.

4.4. Modelo Software del Sistema AHU-9

4.4.1. Introducción

Desglose del modelo de sistema AHU-9 en conexiones y componentes e e interpretación de su representación virtual en Modelica.

Se hará un análisis exhaustivo del modelo "AHU_09_full.mo", describiendo cada uno de los componentes instanciados en el mismo y las interconexiones existentes.

El estudio del fichero busca familiarizar al lector con la notación de los componentes, reflejar las similitudes y disparidades del modelo con la descripción de alto nivel y dar una noción básica para el posterior tratamiento de los componentes, bloque a bloque.

4.4.2. Descripción del modelo de sistema AHU-9

Disponemos de una descripción de alto nivel del modelo del sistema a estudiar y también de un modelo del mismo en Modelica. En las siguientes líneas haremos una breve interpretación del sistema en su representación Modelica, enumerando sus similitudes y diferencias respecto al modelo real.

En primer lugar se adjunta una representación del diagrama gráfico generador por el fichero ".mo" del sistema AHU-9 (AHU_09_full.mo). Para su correcta visualización.

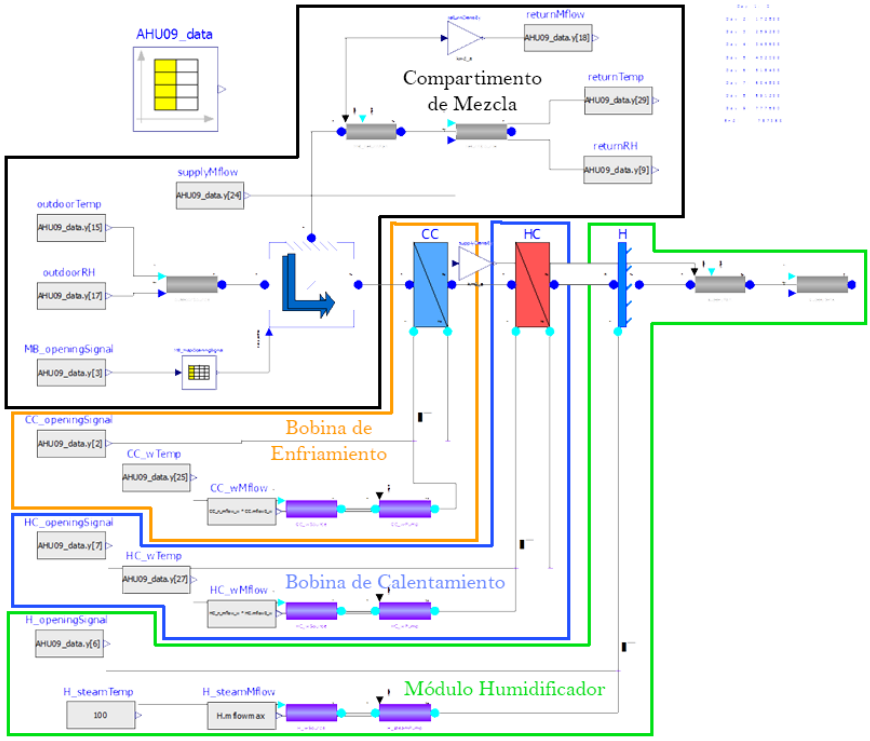


Figura 4.2 Modelo de Sistema AHU-9 Modelica

Lo que se ve representada en la anterior figura es el modelado software del sistema AHU-9. Cada uno de los bloques representativos identificables en el esquema son instancias de clases.

Los bloques grises representan objetos instancia de clases por defecto de Modelica (estas clases permiten ahorrar trabajo al programador en el modelado, al predefinir bloques gráficos representativos de cada clase por defecto). Los bloques restantes, que presentan diversas formas y colores, son componentes del AHU-9, modelados en la biblioteca HVAC_IRUSE.

A pesar de la cantidad de bloques que definen el sistema, en la figura solo se han declarado cuatro componentes básicos:

- Compartimento de Mezcla.
- Bobina de Enfriamiento.
- Bobina de Calentamiento.
- Módulo Humidificador.

Existe un último componente disponible en la biblioteca, el Módulo Mezclador de Aire Auxiliar (Mixing_Box_Supply). Este componente no forma parte del modelo completo del sistema AHU-9, y su implementación difiere de la del mezclador básico.

Además de dichos componentes, se dispone de tres grupos de bombas de líquido refrigerante con sus correspondientes válvulas conductoras (representadas por los bloques rectangulares color magenta); y dos ventiladores (bloques rectangulares de tono gris con degradado).

El modelo presenta diferencias con el sistema real AHU-9 descrito en el apartado de descripción de alto nivel. Para empezar, nuestro sistema se concibe como la interconexión de los cuatro elementos básicos mencionados.

En el modelo de alto nivel, el módulo mezclador se conecta al precalentador, éste al congelador, y el congelador al recalentador. En el modelo "Modelica", el módulo mezclador va conectado directamente al bucle de enfriamiento, éste al bucle de calentamiento, y el bucle de calentamiento al humidificador, un componente no descrito en la lista de componentes de la descripción de alto nivel del sistema (la funcionalidad del humidificador se contempla dentro del bucle de enfriamiento, no como un componente separado).

Para tratar en profundidad el modelo AHU-9 vamos a describir las conexiones que tiene cada componente con el resto de módulos del sistema, a fin de aclarar su configuración:

1. Mixing Box (Compartimento de Mezcla):
   - **Identificador:** MB

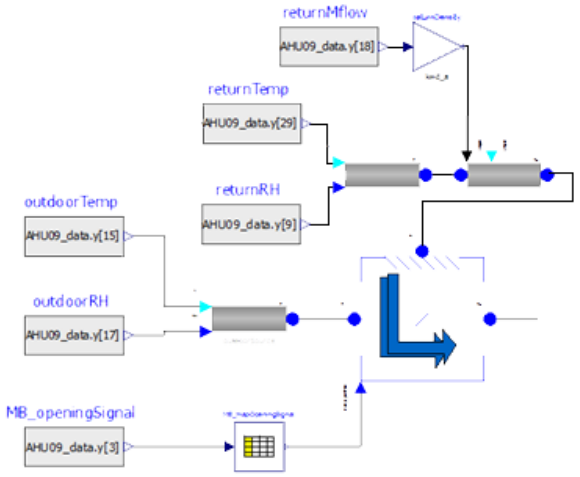


Figura 4.3 Mixing Box (Compartimento de Mezcla)

- **Funcionamiento:**
  Modela el comportamiento del módulo mezclador de aire, tomando como entradas las fuentes de aire externo y reciclado, y generando una nueva fuente de aire (MB.Po), que se conecta a la entrada del bucle de enfriamiento.

- **Conexiones:**
  - Puerto Neumático de Salida (Po) conectado al Puerto Neumático de Entrada (Pi) de CC.
  - Puerto Neumático de Entrada (Pi) conectado al Puerto Neumático de Entrada de aire externo (outdoorSource.P).
  - Puerto Neumático de Entrada (Pr) conectado al Puerto Neumático de Entrada de aire reciclado (MB_returnFan.Po).
  - Control de posición del conducto de aire (MB.damp_position) conectado a MB_mapOpeningSignal. *Se trata de una matriz interpoladora de los valores de entrada (la expresión real AHU09_data.y[3]), compuesta por una entrada y una salida*. El control de posición controla la proporción de aire externo y reciclado que recibe el sistema.

- **Módulos Dependientes:**
  - **outdoorSource.P:**
    Clase de tipo "Pneumatic_Volume" disponible en la biblioteca HVAC_IRUSE.

    Esta clase modela una fuente de entrada de aire, y contempla un doble funcionamiento como fuente y ventilador. Recibe como entradas la temperatura y la humedad relativa del aire, y ofrece como salida una instancia de la clase "Pneumatic" que sirve como conexión de entrada al módulo MB (outdoorSource.P).

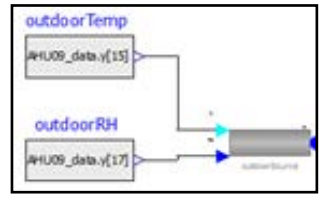


Figura 4.4 outdoorSource

    Esta instancia modela la fuente de aire que introduce en el sistema aire procedente del exterior.

    - **Conexiones del Módulo:**
    - **outdoorTemp.y (conectado a outdoorSource.Ti):**
      Clase tipo "RealExpresion". Toma valor de AHU09_data.y[15] (tabla).
    - **outdoorRH.y (conectado a outdoorSource.RHi):**
      Clase tipo "RealExpresion". Toma valor de AHU09_data.y[17] (tabla).

  - **MB_returnFan.Po:**
    Clase de tipo "Fan_Simplified" disponible en la biblioteca HVAC_IRUSE.

    Esta clase modela un modelo de ventilador que simula el flujo de aire entre los puertos de entrada y salida. Recibe como

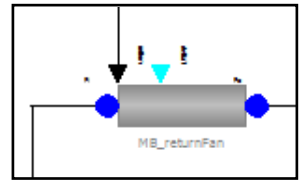


Figura 4.5 returnFan

entradas el flujo de masa del aire (mflow) y un parámetro, deltaT, que marca el incremento de temperatura. A su vez presenta dos puertos neumáticos de entrada y salida (entre los que tiene lugar el flujo de aire).

Esta instancia modela la fuente de aire que introduce en el sistema aire reciclado. Depende de otro módulo adicional independiente de MB, pero que por comodidad describiremos a continuación.

- **Conexiones del Módulo:**

➢ **returnSource.P (conectado a MB_returnFan.Pi):**
Clase tipo "Pneumatic_Volume". Instancia descrita en pag-5.

➢ **returnDensity.y (conectado a MB_returnFan.mflow):**
Clase tipo "Gain". Toma valor de AHU09_data.y[17] (tabla).
La clase "Gain"da como salida el producto de la señal de entrada por el factor "k", especificado en la instancia del objeto (1.18 en este caso).

o **returnSource.P:**
Clase de tipo "Pneumatic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela una fuente de entrada de aire, y contempla un doble funcionamiento como fuente y ventilador. Recibe como entradas la temperatura y la humedad relativa del aire, y ofrece como salida una instancia de la clase "Pneumatic" que sirve como conexión de entrada al módulo MB (returnSource.P).

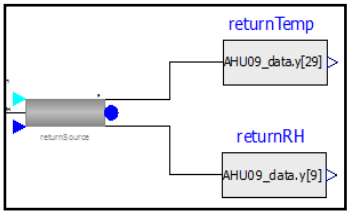


Figura 4.6 returnSource

Esta instancia modela la fuente de aire reciclado, que se introduce en el sistema.

- **Conexiones del Módulo:**

➢ **returnTemp.y (conectado a returnSource.Ti):**
Clase tipo "RealExpresion". Toma valor de AHU09_data.y[29] (tabla).
➢ **returnRH.y (conectado a returnSource.RHi):**
Clase tipo "RealExpresion". Toma valor de AHU09_data.y[9] (tabla).
➢ **MB_returnFan.Pi (conectado a returnSource.P):**
Clase tipo "Fan_Simplified". Instancia descrita en pag-5.

2. CoolingCoil (Bobina de Enfriamiento):
   1. **Identificador:** CC

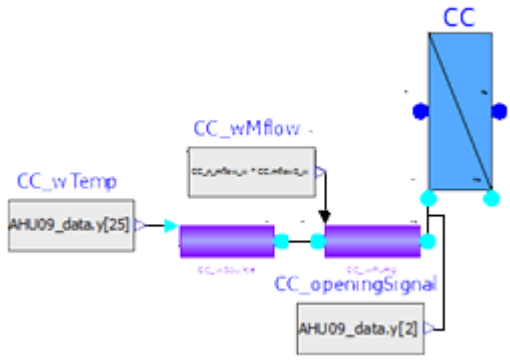


Figura 4.7 CoolingCoil

   2. **Funcionamiento:**

Modela el comportamiento de la bobina de enfriamiento, tomando como entradas la fuente de aire procedente del módulo mezclador, y el líquido refrigerante en circulación que atraviesa los puertos hidráulicos. Genera una nueva fuente de aire (CC.Po), ajustada a una temperatura de enfriamiento adecuada, que se conecta a la entrada del bucle de calentamiento.

   3. **Conexiones:**

3.1. Puerto Neumático de Salida (Po) conectado al Puerto Neumático de Entrada (Pi) de HC.

3.2. Puerto Neumático de Entrada (Pi) conectado al Puerto Neumático de Salida (Po) de MB.

3.3. Puerto Hidráulico de Entrada (Hi) conectado al Puerto Hidráulico de Salida (Ho) de CC_valve.

3.4. Puerto Hidráulico de Salida (Ho) conectado al Puerto Hidráulico de Entrada (H) de CC_HSink.

   4. **Módulos Dependientes:**
      - **CC_valve.Ho:**

Clase de tipo "Valve" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela el comportamiento de histéresis en el flujo de agua, siempre que el parámetro "hysteresis" este marcado a valor "true". Recibe como entradas un puerto hidráulico por donde circula el líquido refrigerante, y un porcentaje que marca el centrode la histéresis. Como salida presenta un puerto hidráulico que

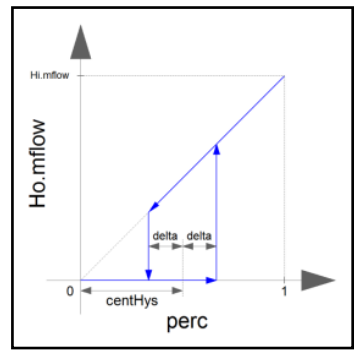


Figura 4.8 hysteresis

representa el flujo de salida del agua. Las clases "valve" solo modifican el flujo de agua entrante en el elemento, manteniendo intacto la temperatura del mismo.

Esta instancia modela la histéresis del flujo de agua refrigerante entrante a CC.

- **Conexiones del Módulo:**
  - **CC_openingSignal.y (conectado a CC_valve.perc):**
    Clase tipo "RealExpresion". Toma valor de AHU09_data.y[2] (tabla).
  - **CC_wPump.Ho (conectado a CC_valve.Hi):**
    Clase tipo "Pump_Simplified". Instancia descrita en pag-6.

- **CC_wPump.Ho:**

Clase de tipo "Pump_Simplified" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela el funcionamiento de una bomba de agua, fijando el ratio de flujo de masa entre los puertos de entrada y salida. Recibe como entradas un puerto hidráulico por donde circula el líquido refrigerante, y el flujo de masa asociado a la circulación del líquido entrante. Como salida presenta un puerto hidráulico que representa el flujo de salida del agua (con el flujo ajustado a los valores de entrada).

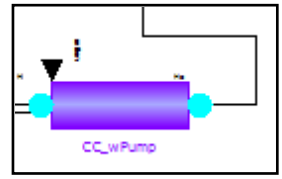


Figura 4.9 CC_wPump

Esta instancia modela la bomba de agua que dirige el líquido refrigerante a la bobina de enfriamiento.

- **Conexiones del Módulo:**
  - **CC_wMflow.y (conectado a CC_wPump.mflow):**
    Clase tipo "RealExpresion". Toma valor producto del parámetro CC_n_mflow_w multiplicado por CC.mflow0_w.
  - **CC_wSource.H (conectado a CC_wPump.Hi):**
    Clase tipo "Hidraulic_Volume". Instancia descrita en pag-6.
  - **CC_valve.Hi (conectado a CC_wPump.Ho):**
    Clase tipo "Valve". Instancia descrita en pag-6.

- **CC_wSource.H:**

Clase de tipo "Hidraulic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modelauna fuente de entrada de agua, y contempla un doble funcionamiento como fuente y ventilador (mediante el parámetro "useSource"). Recibe como entrada únicamente la

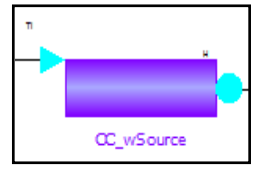


Figura 4.10 CC_wSource

temperatura del agua bombeada, y genera como salida un flujo de agua, con la temperatura ajustada al valor de entrada.

Esta instancia modela la bomba de agua que genera el líquido refrigerante de la bobina de enfriamiento.

- **Conexiones del Módulo:**
  - **CC_wTemp.y (conectado a CC_wSource.Ti):**
    Clase tipo "RealExpresion". Toma valorde AHU09_data.y[25] (tabla).
  - **CC_wPump.Hi (conectado a CC_wSource.H):**
    Clase tipo "Pump_Simplified". Instancia descrita en pag-6.

- **CC_HSink.H:**

  Clase de tipo "Hidraulic_Volume" disponible en la biblioteca HVAC_IRUSE.

  Esta clase modela una fuente de entrada de agua, y contempla un doble funcionamiento como fuente y ventilador (mediante el parámetro "useSource"). Recibe como entrada únicamente la temperatura del agua bombeada, y genera como salida un flujo de agua, con la temperatura ajustada al valor de entrada.

  Esta instancia modela el sumidero de agua que recibe el líquido refrigerante procedente de la salida de la bobina de enfriamiento, y no recibe ningún valor de temperatura.

  - **Conexiones del Módulo:**

3. HeatingCoil (Bobina de Calentamiento):
   1. **Identificador:** HC

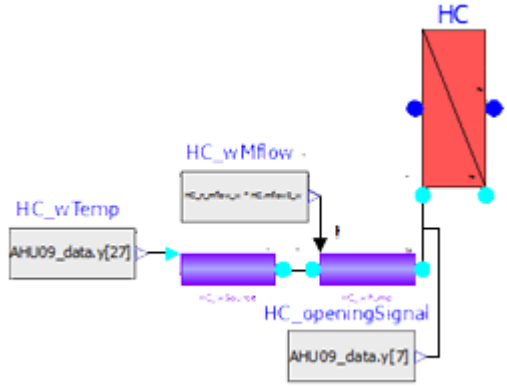


Figura 4.11 HeatingCoil

2. **Funcionamiento:**
   Modela el comportamiento del calentador de aire, tomando como entradas la fuente de aire procedente del bucle de enfriamiento, y el líquido refrigerante en circulación que atraviesa los puertos hidráulicos. Genera una nueva fuente de aire (HC.Po), ajustada a una temperatura de calentamiento adecuada, que se conecta a la entrada del humidificador.

3. **Conexiones:**
   3.1. Puerto Neumático de Salida (Po) conectado al Puerto Neumático de Entrada (Pi) de H.
   3.2. Puerto Neumático de Entrada (Pi) conectado al Puerto Neumático de Salida (Po) de CC.
   3.3. Puerto Hidráulico de Salida (Ho) conectado al Puerto Hidráulico (H) de HC_HSink.
   3.4. Puerto Hidráulico de Entrada (Hi) conectado al Puerto Hidráulico de Entrada (Ho) de HC_valve.

4. **Módulos Dependientes:**
   - **HC_valve.Ho:**
     Clase de tipo "Valve" disponible en la biblioteca HVAC_IRUSE.

     Esta clase modela el comportamiento de histéresis en el flujo de agua, siempre que el parámetro "hysteresis" este marcado a valor "true". Recibe como entradas un puerto hidráulico por donde circula el líquido calefactor, y un porcentaje que marca el centro de la histéresis. Como salida presenta un puerto hidráulico que representa el flujo de salida del agua. Las clases "valve" solo modifican el flujo de agua entrante en el elemento, manteniendo intacto la temperatura del mismo.

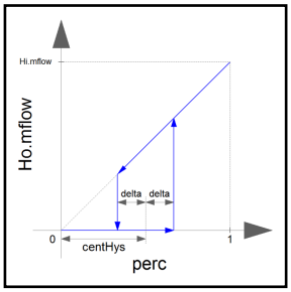


Figura 4.12 hysteresis

     Esta instancia modela la histéresis del flujo de agua calefactora entrante a HC.

     - **Conexiones del Módulo:**
     - **HC_openingSignal.y (conectado a HC_valve.perc):**
       Clase tipo "RealExpresion". Toma valorde AHU09_data.y[7] (tabla).
     - **HC_wPump.Ho (conectado a HC_valve.Hi):**
       Clase tipo "Pump_Simplified". Instancia descrita en pag-8.

   - **HC_wPump.Ho:**
     Clase de tipo "Pump_Simplified" disponible en la biblioteca HVAC_IRUSE.

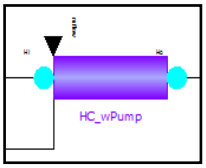


Figura 4.13 HC_wPump

     Esta clase modela el funcionamiento de una bomba de agua, fijando el ratio de flujo de masa entre los puertos

de entrada y salida. Recibe como entradas un puerto hidráulico por donde circula el líquido calefactor, y el flujo de masa asociado a la circulación del líquido entrante. Como salida presenta un puerto hidráulico que representa el flujo de salida del agua (con el flujo ajustado a los valores de entrada).

Esta instancia modela la bomba de agua que dirige el líquido calefactor a la bobina de calentamiento.

- **Conexiones del Módulo:**
- **HC_wSource.H (conectado a HC_wPump.Hi):**
  Clase tipo "Hidraulic_Volume". Instancia descrita en pag-8.
- **HC_wMflow.y (conectado a HC_wPump.mflow):**
  Clase tipo "RealExpresion". Toma valor producto del parámetro HC_n_mflow_w multiplicado por HC.mflow0_w.
- **HC_valve.Hi (conectado a HC_wPump.Ho):**
  Clase tipo "Valve". Instancia descrita en pag-7.

o **HC_wSource.H:**

Clase de tipo "Hidraulic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modelauna fuente de entrada de agua, y contempla un doble funcionamiento como fuente y ventilador (mediante el parámetro "useSource"). Recibe como entrada únicamente la temperatura del agua bombeada, y genera como salida un flujo de agua, con la temperatura ajustada al valor de entrada.

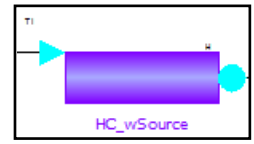


Figura 4.14
HC_wSource

Esta instancia modela la bomba de agua que genera el líquido calefactor de la bobina de calentamiento.

- **Conexiones del Módulo:**
- **HC_wTemp.y (conectado a HC_wSource.Ti):**
  Clase tipo "RealExpresion". Toma valorde AHU09_data.y[27] (tabla).
- **HC_wPump.Hi (conectado a HC_wSource.H):**
  Clase tipo "Pump_Simplified". Instancia descrita en pag-8.

o **HC_HSink.H:**

Clase de tipo "Hidraulic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela una fuente de entrada de agua, y contempla un doble funcionamiento como fuente y ventilador (mediante el parámetro "useSource"). Recibe como entrada únicamente la temperatura del agua bombeada, y genera como salida un flujo de agua, con la temperatura ajustada al valor de entrada.

Esta instancia modela el sumidero de agua que recibe el líquido calefactor procedente de la salida de la bobina de calentamiento, y no recibe ningún valor de temperatura.

- **Conexiones del Módulo:**

4. Humidifier (Módulo Humidificador):
   1. **Identificador:** H

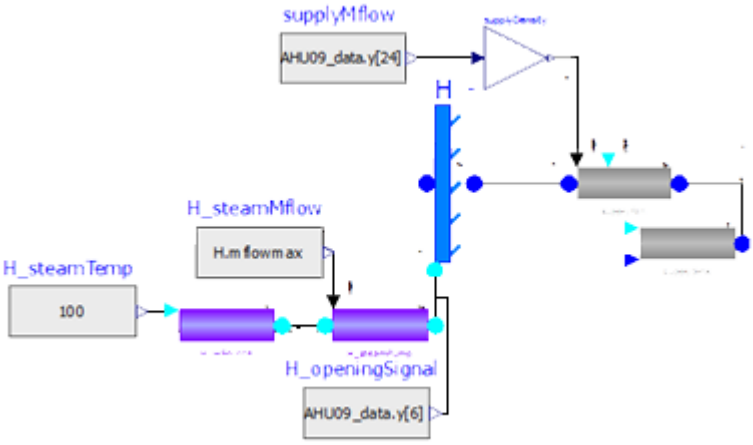


Figura 4.15 Humidifier

   2. **Funcionamiento:**

Modela el comportamiento del humidificador, tomando como entradas la fuente de aire procedente del bucle de calentamiento, y el líquido refrigerante en circulación que atraviesa los puertos hidráulicos. Genera una nueva fuente de aire (H.Po), ajustada a una temperatura de calentamiento adecuada, que se pasa al aula.

   3. **Conexiones:**
      3.1. Puerto Neumático de Salida (Po) conectado al Puerto Neumático de Entrada (Pi) de SupplyFan.
      3.2. Puerto Neumático de Entrada (Pi) conectado al Puerto Neumático de Salida (Po) de HC.
      3.3. No tiene Puerto Hidráulico de Salida (Ho) conectado.
      3.4. Puerto Hidráulico de Entrada (Hi) conectado al Puerto Hidráulico de Entrada (Ho) de H_steamPump.

   4. **Módulos Dependientes:**
      - **H_valve.Ho:**

Clase de tipo "Valve" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela el comportamiento de histéresis en el flujo de agua, siempre que el parámetro "hysteresis" este marcado a valor "true". Recibe como entradas un puerto hidráulico por donde circula el líquido refrigerante,

y un porcentaje que marca el centro de la histéresis. Como salida presenta un puerto hidráulico que representa el flujo de salida del agua. Las clases "valve" solo modifican el flujo de masa entrante en el elemento, manteniendo intacto la temperatura del mismo. Es decir, solo representa el proceso de histéresis, que solo se aplica al flujo de agua.

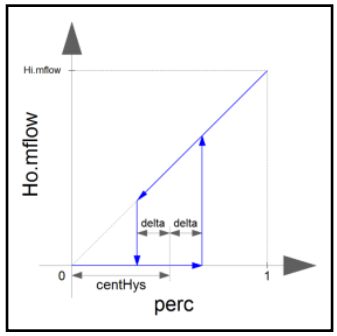


Figura 4.16 hysteresis

Esta instancia modela la histéresis del flujo de agua refrigerante entrante a H.

- **Conexiones del Módulo:**
  - **H_steamPump.Ho (conectado a H_valve.Hi):**
    Clase tipo "Pump_Simplified". Instancia descrita en pag-9.
  - **H_openingSignal.y (conectado a H_valve.perc):**
    Clase tipo "RealExpresion". Toma valorde AHU09_data.y[6] (tabla).

- **H_steamPump.Ho:**

Clase de tipo "Pump_Simplified" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela el funcionamiento de una bomba de agua, fijando el ratio de flujo de masa entre los puertos de entrada y salida. Recibe como entradas un puerto hidráulico por donde circula el líquido refrigerante, y el flujo de masa asociado a la circulación del líquido entrante. Como salida presenta un puerto hidráulico que representa el flujo de salida del agua (con el flujo ajustado a los valores de entrada).

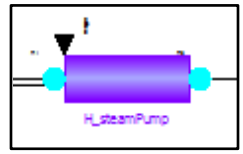


Figura 4.17 H_steamPump

Esta instancia modela la bomba de agua que dirige el líquido refrigerante al humidificador.

- **Conexiones del Módulo:**
  - **H_wSource.H (conectado a H_steamPump.Hi):**
    Clase tipo "Hidraulic_Volume". Instancia descrita en pag-10.
  - **H_valve.Hi (conectado a H_steamPump.Ho):**
    Clase tipo "Valve". Instancia descrita en pag-9.
  - **H_steamMflow.y (conectado a H_steamPump.mflow):**
    Clase tipo "RealExpresion". Toma valorde H.mflowmax.

- **H_wSource.H:**

Clase de tipo "Hidraulic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modelauna fuente de entrada de agua, y contempla un doble funcionamiento como fuente y ventilador (mediante el parámetro "useSource"). Recibe como entrada únicamente la temperatura del agua bombeada, y genera como salida un flujo de agua, con la temperatura ajustada al valor de entrada.

Figura 4.18
H_wSource

Esta instancia modela la bomba de agua que genera el líquido refrigerante del humidificador.

- **Conexiones del Módulo:**
- **H_steamTemp (conectado a H_wSource.Ti):**
  Clase tipo "RealExpresion". Toma valor100.
- **H_steamPump.Hi (conectado a H_wSource.H):**
  Clase tipo "Pump_Simplified". Instancia descrita en pag-9.

o **SupplyFan.Pi:**
Clase de tipo "Fan_Simplified" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela un modelo de ventilador que simula el flujo de aire entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del aire (mflow) y un parámetro, deltaT, que marca el incremento de temperatura. A su vez presenta dos puertos neumáticos de entrada y salida (entre los que tiene lugar el flujo de aire).

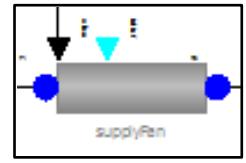


Figura 4.19
SupplyFan

Esta instancia modela la fuente de aire que introduce en las aulas. Solo altera la densidad de flujo de aire en el sistema, manteniendo intacta la temperatura. El módulo no va comunicado directamente a las clases, sino que se conecta a un ventilador.

- **Conexiones del Módulo:**
- **supplyDensity.y (conectado a supplyFan.mflow):**
  Clase tipo "Gain". Toma valor de AHU09_data.y[24] (tabla).
  La clase "Gain"da como salida el producto de la señal de entrada por el factor "k", especificado en la instancia del objeto (1.18 en este caso).
- **supplySink.P (conectado a supplyFan.Po):**
  Clase tipo "Pneumatic_Volume". Instancia descrita en pag-10.

o **SupplySink.P:**
Clase de tipo "Pneumatic_Volume" disponible en la biblioteca HVAC_IRUSE.

Esta clase modela una fuente de entrada de aire, y contempla un doble funcionamiento como fuente y ventilador. Recibe como entradas la temperatura y la

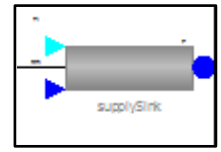


Figura 4.20
SupplySink

humedad relativa del aire, y ofrece como salida una instancia de la clase "Pneumatic" que sirve como conexión de entrada.

Esta instancia modela el ventilador de aire que introduce el aire aclimatado en las clases. No modifica ningún parámetro del flujo de aire.

- **Conexiones del Módulo:**
  - **supplyFan.Po (conectado a supplySink.P):**
    Clase tipo "Fan_Simplified". Instancia descrita en pag-10.

❖ Descripción de la Tabla AHU09_data:

Identificador de entrada	Nombre de entrada	Tipo	Conexión	Descripción
AHU09_data.y[2]	CC_openingSignal	Real	CC_valve.perc	Valor de apertura de la válvula de entrada de líquido refrigerante a CC por Hi.
AHU09_data.y[3]	MB_openingSignal	Real	MB_mapOpeningSignal	Valor de control de la proporción de aire entrante al sistema procedente del exterior y del propio sistema (aire reciclado)
AHU09_data.y[6]	H_openingSignal	Real	H_valve.perc	Valor de apertura de la válvula de entrada de líquido refrigerante a H por Hi.
AHU09_data.y[7]	HC_openingSignal	Real	HC_valve.perc	Valor de apertura de la válvula de entrada de líquido refrigerante a HC por Hi.
AHU09_data.y[9]	returnRH	Real	returnSource.RHi	Valor de humedad relativa del aire reciclado.
AHU09_data.y[15]	outdoorTemp	Real	outdoorSource.Ti	Valor de temperatura del aire exterior.
AHU09_data.y[17]	outdoorRH	Real	outdoorSource.RHi	Valor de humedad relativa del aire exterior.
AHU09_data.y[18]	returnMflow	Real	returnDensity	Valor de flujo del aire reciclado.

AHU09_data.y[24]	supplyMflow	Real	supplyDensity	Valor de flujo del aire aclimatado.
AHU09_data.y[25]	CC_wTemp	Real	CC_wSource.Ti	Valor de temperatura del líquido refrigerante entrante a CC por Hi.
AHU09_data.y[27]	HC_wTemp	Real	HC_wSource.Ti	Valor de temperatura del líquido refrigerante entrante a HC por Hi.
AHU09_data.y[29]	returnTemp	Real	returnSource.Ti	Valor de temperatura del aire reciclado.

4.4.3. Componentes del modelo de sistema AHU-9

Estudio de los bloques funcionales básicos que definen el sistema AHU-9. Análisis de los datos de entrada, salida y medidas de cada componente. Estudio y comprensión de la estructura interna de cada bloque: ecuaciones, suposiciones y comportamientos.

Se indagará en el código fuente de los elementos del paquete "Components" de la biblioteca HVAC_IRUSE. Con este análisis y el previamente realizado sobre el sistema se abarcará el estudio completo de todo el código del software del sistema.

Analizamos los valores de entrada, salida y medidas que se realizan dentro de los diferentes componentes básicos del sistema.

1. Mixing Box:

- Entradas:
  - Puerto Neumático Pi:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W – humidity ratio).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Neumático Pr:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W – humidity ratio).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático Po:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W – humidity ratio).
    - Ratio de Flujo de Masa (mflow – kg/s).

- Mediciones:
  (No hay mediciones adicionales).

- Otros Datos:
  - Posición del Amortiguador (determina fuente de flujo de aire seleccionada de entrada – [0,1]).

2. CoolCoil:

- Entradas:
  - Puerto Neumático Pi:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W – humidity ratio).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidraúlico Hi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático Po:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W - humidity).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidráulico Ho:
    - Temperatura del Aire (T -ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
  - Capacidad de enfriamiento (Qflow_coil).
  - Efectividad (método *eff-NTU*).
  - Ratio de capacidad de flujo de aire (Z).
  - Número de unidades de transferencia (NTU).
  - Resistencia térmica (R_dry).
- Otros Datos:
  (No hay otros datos adicionales).

3. HeatCoil:

- Entradas:
  - Puerto Neumático Pi:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W – humidity ratio).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidraúlico Hi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático Po:
    - Temperatura del Aire (T - ºC).
    - Fracción de Masa (W - humidity).

        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidráulico Ho:
        - Temperatura del Aire (T -ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
    - Efectividad (método *eff-NTU*).
    - Ratio de capacidad de flujo de aire (Z).
    - Número de unidades de transferencia (NTU).
    - Resistencia térmica (R).

- Otros Datos:
    - Constante de tiempo del sistema (tau).

4. Humidifier:

- Entradas:
    - Puerto Neumático Pi:
        - Temperatura del Aire (T - ºC).
        - Fracción de Masa (W – humidity ratio).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidraúlico Hi:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
    - Puerto Neumático Po:
        - Temperatura del Aire (T - ºC).
        - Fracción de Masa (W - humidity).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
    - Capacidad de calentamiento específica del aire entrante (c_ai).
- Otros Datos:
  (No hay otros datos adicionales).

4.5. Modelo Software del Sistema AHU-9 Simplificado

4.5.1. Introducción

Desglose del modelo de sistema AHU-9 Simplificado en conexiones y componentes. Interpretación de su representación virtual en Modelica.

Se hará un análisis exhaustivo del modelo "AHU_09.mo", describiendo cada uno de los componentes instanciados en el mismo y las interconexiones existentes.

El estudio del fichero busca familiarizar al lector con la notación de los componentes, reflejar las similitudes y disparidades del modelo con la descripción de alto nivel y dar una noción básica para el posterior tratamiento de los componentes, bloque a bloque.

4.5.2. Descripción del modelo de sistema AHU-9 Simplificado

El modelo AHU-9 Simplificado consiste en una revisión del modelo de sistema de ventilación "AHU_09_full.mo" el cual ajusta su implementación a los detalles técnicos descriptivos del sistema real que modela.

Esta revisión surge para evitar las discrepancias existentes entre el anterior modelo de sistema y la definición de sistema real AHU-9. Presenta un código más simplificado, en el que el número de componentes definidos es menor, toda la definición de los mismos y sus interconexiones se contemplan en un mismo fichero (no hay bibliotecas de componentes como en el caso anterior), y no se define una representación gráfica de los mismos (por lo que en no se dispone de un diagrama representativo de los componentes y sus interconexiones).

El nuevo modelo presenta una configuración pareja a la de la descripción real del sistema AHU-9. Dentro de él se modelan además otros "componentes" como puedan ser el ambiente exterior, el ambiente dentro de la sala y los conductos de refrigeración y ventilación del sistema. Además, el modelo prescinde de considerar parámetros referidos a la humedad del aire (ni se controla ni se mide el nivel de humedad, trabajamos solo con flujos y temperaturas).

A continuación se procede a la descripción de cada uno de los componentes presentes en el sistema:

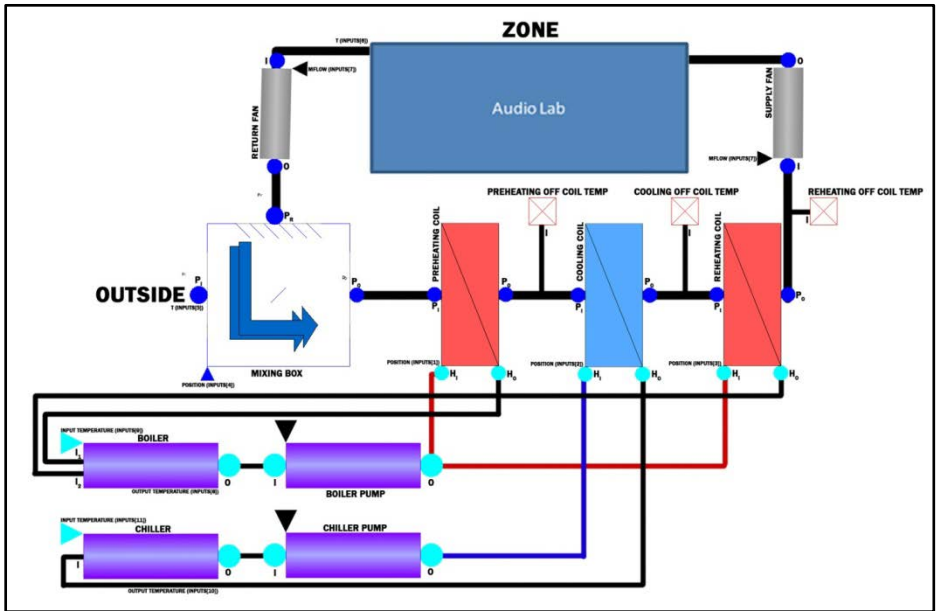


Figura 4.21 Modelo de Sistema AHU-9 Simplificado

1. Mixing Box (Compartimento de Mezcla):
   - **Identificador:** mixingBox (instancia de clase MixingBox)

   - **Funcionamiento:**
     Modela el comportamiento del módulo mezclador de aire, tomando como entradas las fuentes de aire exterior y reciclado, y generando una nueva fuente de aire (mixingBox.po), que se conecta a la entrada de la bobina de precalentamiento.

   - **Descripción de Clase:**
     - **Variables:**
       - PneumaticPort pi: modela conexión de uno de los puertos neumáticos de entrada con el exterior.
       - PneumaticPort pr: modela conexión de uno de los puertos neumáticos de entrada con la salida de aire reciclado del sistema.
       - PneumaticPort po: modela conexión del puerto neumático de salida con el puerto neumático de entrada de la bobina de precalentamiento (preheatingCoil).
       - Real position: modela la posición de la válvula de aire del compartimento que controla la entrada de aire procedente del exterior y reciclado. Toma valor de la entrada de la tabla "inputs.y[4]".

     - **Lógica del Componente:**

       La clase MixingBox basa su funcionamiento en el modelado de un flujo de aire saliente ajustado a los valores de entrada de los puertos neumáticos. Considera que el flujo de aire exterior es nulo, de manera que todo el flujo entrante al componente proviene del conducto de aire reciclado (y en dicho flujo no interviene la posición de la válvula).

       La temperatura del flujo saliente del compartimento de mezclas se estima a partir de la suma de flujos entrantes al sistema, multiplicados respectivamente por su temperatura, y divididos por el flujo de salida:

       $$po.mflow * po.T = outdoorFlow * pi.T + returnFlow * pr.T$$

   - **Conexiones:**
     - Puerto Neumático de Salida (po) conectado al Puerto Neumático de Entrada (pi) de la bobina de precalentamiento (preheatingCoil.pi).
     - Puerto Neumático de Entrada (pi) conectado al Puerto Neumático de Entrada de aire externo (outside.o).
     - Puerto Neumático de Entrada (pr) conectado al Puerto Neumático de Entrada de aire reciclado (returnfan.o).

   - **Módulos Dependientes:**
     - **outside.o:**
       Clase de tipo “Outside” disponible en el fichero "ahu-9.mo".

Esta clase modela la fuente de entrada de aire procedente del exterior. Recibe como entrada la temperatura del aire, y ofrece como salida una instancia de la clase "PneumaticPort" que sirve como conexión de entrada al módulo mixingBox (outside.o).

Esta instancia modela la fuente de aire que introduce en el sistema aire procedente del exterior.

- **Conexiones del Módulo:**
  - **outside.T:**
    Clase tipo "Real". Toma valor de "inputs.y[5]" (tabla).

- **returnfan.o:**
  Clase de tipo "Fan" disponible en el fichero "ahu-9.mo".

Esta clase modela un modelo de ventilador que simula el flujo de aire entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del aire (mflow, toma valor de "inputs.y[7]"). A su vez presenta dos puertos neumáticos de entrada y salida (entre los que tiene lugar el flujo de aire). En el ventilador la temperatura del flujo de aire entrante y saliente es la misma.

Esta instancia modela la fuente de aire que introduce en el sistema aire reciclado. Dicho flujo de aire se corresponde con el aire extraído de la sala aclimatada (zone).

- **Conexiones del Módulo:**
  - **returnfan.o (conectado a mixingBox.pr):**
    Clase tipo "PneumaticPort".
  - **returnfan.i (conectado a zone.o):**
    Clase tipo "PneumaticPort".

2. Preheating Coil (Bobina de Precalentamiento):
   - **Identificador:** preheatingCoil (instancia de clase HeatingCoil)

   - **Funcionamiento:**
     Modela el comportamiento de la bobina de precalentamiento, tomando como entrada la fuente de aire procedente del compartimento de mezcla, y generando una nueva fuente de aire (preheatingCoil.po), que se conecta a la entrada de la bobina de enfriamiento.

   - **Descripción de Clase:**
     - **Parámetros:**
       - pneumaticSpecificHeat: valor 1006.
       - hydraulicSpecificHeat: valor 4186.
       - UA: empleado para la estimación de la efectividad. Valor 6400.

- **Variables:**
  - PneumaticPort pi: modela conexión de uno de los puertos neumáticos de entrada con el puerto neumático de salida del compartimento de mezcla (mixingBox).
  - PneumaticPort po: modela conexión del puerto neumático de salida con el puerto neumático de entrada de la bobina de enfriamiento (coolingCoil).
  - HydraulicPort hi: modela conexión del puerto hidráulico de entrada con la válvula que conduce el líquido calefactor de la bomba al precalentador (preheatingCoilValve).
  - HydraulicPort ho: modela conexión del puerto hidráulico de salida con la válvula que conduce el líquido calefactor del precalentado a la bomba (gasBoiler).
  - Real eff: la clase en cuestión define una serie de variables privadas para la posterior aplicación del método de efectividad, cuyo resultado se almacena en esta variable.

- **Lógica del Componente:**

  La clase HeatingCoil basa su funcionamiento en el modelado de un flujo de aire saliente ajustado a los valores de entrada del puerto neumático de aire procedente del compartimento de mezcla, y el puerto hidráulico de líquido calefactor procedente de la bomba de líquido calefactor. Considera que siempre hay un flujo de aire y líquido (mínimo 0.001), y mantiene constante dichos flujos en entrada y salida.

  La temperatura de los flujos de agua y aire salientes del componente son estimados en base a datos de efectividad y temperatura de los conductos de entrada.

- **Conexiones:**
  - Puerto Neumático de Salida (po) conectado al Puerto Neumático de Entrada (pi) de la bobina de enfriamiento (coolingCoil.pi).
  - Puerto Neumático de Entrada (pi) conectado al Puerto Neumático de Entrada (po) del compartimento de mezcla (mixingBox.po).
  - Puerto Hidráulico de Salida (ho) conectado al Puerto Hidráulico de Entrada del calentador de agua (gasBoiler.i1).
  - Puerto Hidráulico de Entrada (hi) conectado al Puerto Hidráulico de Salida de la válvula de la bobina de precalentamiento (preheatingCoilValve.o).

- **Módulos Dependientes:**
  - **preheatingCoilValve.o:**

    Clase de tipo "HydraulicValve" disponible en el fichero "ahu-9.mo".

    Esta clase modela el comportamiento de una válvula de agua. Contempla una variable de posición que determina el porcentaje de flujo entrante que atraviesa el conducto. La válvula mantiene constante la temperatura del flujo de agua circulante por ella.

Esta instancia modela la válvula de agua que conduce el líquido calefactor de la bomba a la bobina de precalentamiento.

- **Conexiones del Módulo:**
  - **preheatingCoilValve.o (conectado a preheatingCoil.hi):**
    Clase tipo "HydraulicPort".
  - **preheatingCoilValve.i (conectado a boilerPump.o):**
    Clase tipo "HydraulicPort".

  - **preheatingCoilValve.position:**
    Clase tipo "Real". Toma valor de "inputs.y[1]" (tabla).

- **boilerPump.o:**
  Clase de tipo "HydraulicPump" disponible en el fichero "ahu-9.mo".

Esta clase modela un modelo de bomba de masa que simula el flujo de agua entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del agua (mflow, que en este caso es constante, con valor 1.5). A su vez presenta dos puertos hidráulicos de entrada y salida (entre los que tiene lugar el flujo de agua). En él la temperatura del flujo de aire entrante y saliente es la misma.

Esta instancia modela la bomba de agua que introduce el líquido calefactor en las bobinas de precalentamiento y recalentamiento. Dicho flujo de agua se corresponde con el líquido procedente del calentador (gasBoiler).

- **Conexiones del Módulo:**
  - **boilerPump.o (conectado a preheatingCoilValve.i):**
    Clase tipo "HydraulicPort".

  - **boilerPump.i (conectado a gasBoiler.o):**
    Clase tipo "HydraulicPort".

- **gasBoiler.o:**
  Clase de tipo "GasBoiler" disponible en el fichero "ahu-9.mo".

Esta clase modela un modelo de calentador de agua que simula el flujo de agua entre los dos puertos de entrada y el puerto de salida. Recibe como entradas la temperatura de entrada (inputTemperature, toma valor "inputs.y[9]") y calentamiento del agua (outputTemperature, toma valor "inputs.y[8]"). A su vez presenta dos puertos hidráulicos de entrada y uno de salida (entre los que tiene lugar el flujo de agua). En él, el flujo de agua saliente es igual a la suma de flujos de agua entrantes.

Esta instancia modela el calentador de agua que introduce el líquido calefactor en la bomba de agua. Dicho flujo de agua se corresponde con el líquido reciclado procedente de las bobinas de precalentamiento y recalentamiento.

- **Conexiones del Módulo:**
  - **gasBoiler.o (conectado a boilerPump.i):**
    Clase tipo "HydraulicPort".
  - **gasBoiler.i1 (conectado a preheatingCoil.ho):**
    Clase tipo "HydraulicPort".
  - **gasBoiler.i2 (conectado a reheatingCoil.ho):**
    Clase tipo "HydraulicPort".

- **preheatingOffCoilTemp.i:**
  Clase de tipo "TemperatureSensor" disponible en el fichero "ahu-9.mo".

  Esta clase modela el comportamiento de un sensor de temperatura del aire. Contempla un puerto neumático (i) que conecta al sistema con el flujo de aire circundante entre componentes, y del cual extrae la medida de temperatura. Contempla un algoritmo para la generación de resultados y el etiquetado de los mismos.

  Esta instancia modela el sensor que mide la temperatura del aire saliente del precalentador.

  - **Conexiones del Módulo:**
    - **preheatingOffCoilTemp.i (conectado a preheatingCoil.po):**
      Clase tipo "PneumaticPort".

3. Cooling Coil (Bobina de Enfriamiento):
   - **Identificador:** coolingCoil (instancia de clase CoolingCoil)

   - **Funcionamiento:**
     Modela el comportamiento de la bobina de enfriamiento, tomando como entrada la fuente de aire procedente de la bobina de precalentamiento, y generando una nueva fuente de aire (coolingCoil.po), que se conecta a la entrada de la bobina de recalentamiento.

   - **Descripción de Clase:**
     - **Parámetros:**
       - pneumaticSpecificHeat: valor 1006.
       - hydraulicSpecificHeat: valor 4186.
       - UA: empleado para la estimación de la efectividad. Valor 6400.

     - **Variables:**
       - PneumaticPort pi: modela conexión del puerto neumático de entrada con el puerto neumático de salida de la bobina de precalentamiento.
       - PneumaticPort po: modela conexión del puerto neumático de salida con el puerto neumático de entrada de la bobina de recalentamiento (reheatingCoil).
       - HydraulicPort hi: modela conexión del puerto hidráulico de entrada con la válvula que conduce el líquido refrigerante de la bomba al enfriador.
       - HydraulicPort ho: modela conexión del puerto hidráulico de salida con la válvula que conduce el líquido refrigerante del enfriador a la bomba.

- Real eff: la clase en cuestión define una serie de variables privadas para la posterior aplicación del método de efectividad, cuyo resultado se almacena en esta variable.

➢ **Lógica del Componente:**

La clase CoolingCoil basa su funcionamiento en el modelado de un flujo de aire saliente ajustado a los valores de entrada del puerto neumático de aire procedente del buble precalentador, y el puerto hidráulico de líquido calefactor procedente de la bomba de líquido calefactor. Considera que siempre hay un flujo de aire y líquido (mínimo 0.001), y mantiene constante dichos flujos en entrada y salida.

La temperatura de los flujos de agua y aire salientes del componente son estimados en base a datos de efectividad y temperatura de los conductos de entrada.

La definición de esta clase coincide a nivel de código fuente con la definición de la clase "HeatingCoil".

- **Conexiones:**
  - Puerto Neumático de Salida (po) conectado al Puerto Neumático de Entrada (pi) de la bobina de recalentamiento (reheatingCoil.pi).
  - Puerto Neumático de Entrada (pi) conectado al Puerto Neumático de Entrada (po) de la bobina de precalentamiento (preheatingCoil.po).
  - Puerto Hidráulico de Salida (ho) conectado al Puerto Hidráulico de Entrada del enfriador de agua (chiller.i).
  - Puerto Hidráulico de Entrada (hi) conectado al Puerto Hidráulico de Salida de la válvula de la bobina de enfriamiento (coolingCoilValve.o).

- **Módulos Dependientes:**
  - **coolingCoilValve.o:**

    Clase de tipo "HydraulicValve" disponible en el fichero "ahu-9.mo".

    Esta clase modela el comportamiento de una válvula de agua. Contempla una variable de posición que determina el porcentaje de flujo entrante que atraviesa el conducto. La válvula mantiene constante la temperatura del flujo de agua circulante por ella.

    Esta instancia modela la válvula de agua que conduce el líquido refrigerante de la bomba a la bobina de enfriamiento.

    - **Conexiones del Módulo:**
      - **coolingCoilValve.o (conectado a coolingCoil.hi):**
        Clase tipo "HydraulicPort".
      - **coolingCoilValve.i (conectado a chiller.o):**
        Clase tipo "HydraulicPort".
      - **coolingCoilValve.position:**
        Clase tipo "Real". Toma valor de "inputs.y[2]" (tabla).

- **chillerPump.o:**
  Clase de tipo "HydraulicPump" disponible en el fichero "ahu-9.mo".

  Esta clase modela un modelo de bomba de masa que simula el flujo de agua entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del agua (mflow, que en este caso es constante, con valor 1.5). A su vez presenta dos puertos hidráulicos de entrada y salida (entre los que tiene lugar el flujo de agua). En él la temperatura del flujo de aire entrante y saliente es la misma.

  Esta instancia modela la bomba de agua que introduce el líquido refrigerante en la bobina de enfriamiento. Dicho flujo de agua se corresponde con el líquido procedente del refrigerador (chiller).

  - **Conexiones del Módulo:**
    - **chillerPump.o (conectado a coolingCoilValve.i):**
      Clase tipo "HydraulicPort".
    - **chillerPump.i (conectado a chiller.o):**
      Clase tipo "HydraulicPort".

- **chiller.o:**
  Clase de tipo "Chiller" disponible en el fichero "ahu-9.mo".

  Esta clase modela un modelo de refrigerador de agua que simula el flujo de agua entre los puertos de entrada y salida. Recibe como entradas la temperatura de entrada (inputTemperature, toma valor "inputs.y[11]") y enfriamiento del agua (outputTemperature, toma valor "inputs.y[10]"). A su vez presenta dos puertos hidráulicos de entrada y salida (entre los que tiene lugar el flujo de agua). En él, el flujo de agua saliente es igual al flujo de agua entrante.

  Esta instancia modela el refrigerador de agua que introduce el líquido refrigerante en la bomba de agua. Dicho flujo de agua se corresponde con el líquido reciclado procedente de la bobina de enfriamiento.

  - **Conexiones del Módulo:**
    - **chiller.o (conectado a chillerPump.i):**
      Clase tipo "HydraulicPort".
    - **chiller.i (conectado a coolingCoil.ho):**
      Clase tipo "HydraulicPort".

- **coolingOffCoilTemp.i:**
  Clase de tipo "TemperatureSensor" disponible en el fichero "ahu-9.mo".

  Esta clase modela el comportamiento de un sensor de temperatura del aire. Contempla un puerto neumático (i) que conecta al sistema con el flujo de aire circundante entre componentes, y del cual extrae la medida de temperatura. Contempla un algoritmo para la generación de resultados y el etiquetado de los mismos.

Esta instancia modela el sensor que mide la temperatura del aire saliente del refrigerador.

- **Conexiones del Módulo:**
  - **coolingOffCoilTemp.i (conectado a coolingCoil.po):** Clase tipo "PneumaticPort".

4. Reheating Coil (Bobina de Recalentamiento):
   - **Identificador:** reheatingCoil (instancia de clase HeatingCoil)

   - **Funcionamiento:**
     Modela el comportamiento de la bobina de recalentamiento, tomando como entrada la fuente de aire procedente de la bobina de enfriamiento, y generando una nueva fuente de aire (reheatingCoil.po), que se conecta a la entrada del ventilador de aire que comunica el circuito de ventilación con la sala aclimatada.

   - **Descripción de Clase:**
     - **Parámetros:**
       - pneumaticSpecificHeat: valor 1006.
       - hydraulicSpecificHeat: valor 4186.
       - UA: empleado para la estimación de la efectividad. Valor 6400.

     - **Variables:**
       - PneumaticPort pi: modela conexión del puerto neumático de entrada con el puerto neumático de salida de la bobina de enfriamiento (coolingCoil).
       - PneumaticPort po: modela conexión del puerto neumático de salida con el puerto neumático de entrada del ventilador de la sala (supplyFan).
       - HydraulicPort hi: modela conexión del puerto hidráulico de entrada con la válvula que conduce el líquido calefactor de la bomba al recalentador (reheatingCoilValve).
       - HydraulicPort ho: modela conexión del puerto hidráulico de salida con la válvula que conduce el líquido calefactor del recalentador a la bomba (gasBoiler).
       - Real eff: la clase en cuestión define una serie de variables privadas para la posterior aplicación del método de efectividad, cuyo resultado se almacena en esta variable.

     - **Lógica del Componente:**
       La clase HeatingCoil basa su funcionamiento en el modelado de un flujo de aire saliente ajustado a los valores de entrada del puerto neumático de aire procedente del bucle de enfriamiento, y el puerto hidráulico de líquido calefactor procedente de la bomba de líquido calefactor. Considera que siempre hay un flujo de aire y líquido (mínimo 0.001), y mantiene constante dichos flujos en entrada y salida.

       La temperatura de los flujos de agua y aire salientes del componente son estimados en base a datos de efectividad y temperatura de los conductos de entrada.

- **Conexiones:**
  - Puerto Neumático de Salida (po) conectado al Puerto Neumático de Entrada (pi) de la bobina de enfriamiento (supplyFan.i).
  - Puerto Neumático de Entrada (pi) conectado al Puerto Neumático de Entrada (po) de la bobina de enfriamiento (coolingCoil.po).
  - Puerto Hidráulico de Salida (ho) conectado al Puerto Hidráulico de Entrada del calentador de agua (gasBoiler.i1).
  - Puerto Hidráulico de Entrada (hi) conectado al Puerto Hidráulico de Salida de la válvula de la bobina de recalentamiento (reheatingCoilValve.o).

- **Módulos Dependientes:**
  - **reheatingCoilValve.o:**

    Clase de tipo "HydraulicValve" disponible en el fichero "ahu-9.mo".

    Esta clase modela el comportamiento de una válvula de agua. Contempla una variable de posición que determina el porcentaje de flujo entrante que atraviesa el conducto. La válvula mantiene constante la temperatura del flujo de agua circulante por ella.

    Esta instancia modela la válvula de agua que conduce el líquido calefactor de la bomba a la bobina de recalentamiento.

    - **Conexiones del Módulo:**
      - **reheatingCoilValve.o (conectado a reheatingCoil.hi):**
        Clase tipo "HydraulicPort". Instancia descrita en pag-XX.
      - **reheatingCoilValve.i (conectado a boilerPump.o):**
        Clase tipo "HydraulicPort". Instancia descrita en pag-XX.
      - **reheatingCoilValve.position:**
        Clase tipo "Real". Toma valor de "inputs.y[3]" (tabla).

  - **boilerPump.o:**

    Clase de tipo "HydraulicPump" disponible en el fichero "ahu-9.mo".

    Esta clase modela un modelo de bomba de masa que simula el flujo de agua entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del agua (mflow, que en este caso es constante, con valor 1.5). A su vez presenta dos puertos hidráulicos de entrada y salida (entre los que tiene lugar el flujo de agua). En él la temperatura del flujo de aire entrante y saliente es la misma.

    Esta instancia modela la bomba de agua que introduce el líquido calefactor en las bobinas de precalentamiento y recalentamiento. Dicho flujo de agua se corresponde con el líquido procedente del calentador (gasBoiler).

    - **Conexiones del Módulo:**
      - **boilerPump.o (conectado a reheatingCoilValve.i):**
        Clase tipo "HydraulicPort".
      - **boilerPump.i (conectado a gasBoiler.o):**
        Clase tipo "HydraulicPort".

- **gasBoiler.o:**
  Clase de tipo "GasBoiler" disponible en el fichero "ahu-9.mo".

  Esta clase modela un modelo de calentador de agua que simula el flujo de agua entre los dos puertos de entrada y el puerto de salida. Recibe como entradas la temperatura de entrada (inputTemperature, toma valor "inputs.y[9]") y calentamiento del agua (outputTemperature, toma valor "inputs.y[8]"). A su vez presenta dos puertos hidráulicos de entrada y uno de salida (entre los que tiene lugar el flujo de agua). En él el flujo de agua saliente es igual a la suma de flujos de agua entrantes.

  Esta instancia modela el calentador de agua que introduce el líquido calefactor en la bomba de agua. Dicho flujo de agua se corresponde con el líquido reciclado procedente de las bobinas de precalentamiento y recalentamiento.

  - **Conexiones del Módulo:**
    - **gasBoiler.o (conectado a boilerPump.i):**
      Clase tipo "HydraulicPort".
    - **gasBoiler.i1 (conectado a preheatingCoil.ho):**
      Clase tipo "HydraulicPort".
    - **gasBoiler.i2 (conectado a reheatingCoil.ho):**
      Clase tipo "HydraulicPort".

- **supplyfan.o:**
  Clase de tipo "Fan" disponible en el fichero "ahu-9.mo".

  Esta clase modela un modelo de ventilador que simula el flujo de aire entre los puertos de entrada y salida. Recibe como entradas el flujo de masa del aire (mflow, toma valor de "inputs.y[7]"). A su vez presenta dos puertos neumáticos de entrada y salida (entre los que tiene lugar el flujo de aire). En el ventilador la temperatura del flujo de aire entrante y saliente es la misma.

  Esta instancia modela la fuente de aire que introduce en la sala aclimatada el aire ajustado al nivel de temperatura deseado.

  - **Conexiones del Módulo:**
    - **supplyfan.o (conectado a zone.i):**
      Clase tipo "PneumaticPort".
    - **supplyfan.i (conectado a reheatingCoil.po):**
      Clase tipo "PneumaticPort".

- **zone.i:**
  Clase de tipo "Zone" disponible en el fichero "ahu-9.mo".

  Esta clase modela el ambiente de la sala aclimatada, que simula el flujo de aire entre los puertos de entrada y salida. Recibe como entradas la temperatura del aire (T, toma valor de "inputs.y[6]"). A su vez presenta dos puertos neumáticos de

entrada y salida (entre los que tiene lugar el flujo de aire). En la sala el flujo de aire entrante y saliente es el mismo.

Esta instancia modela la fuente de aire que circula en la sala aclimatada.

- **Conexiones del Módulo:**
  - **zone.o (conectado a returnFan.i):**
    Clase tipo "PneumaticPort".
  - **zone.i (conectado a supplyFan.o):**
    Clase tipo "PneumaticPort".

- **reheatingOffCoilTemp.i:**
  Clase de tipo "TemperatureSensor" disponible en el fichero "ahu-9.mo".

Esta clase modela el comportamiento de un sensor de temperatura del aire. Contempla un puerto neumático (i) que conecta al sistema con el flujo de aire circundante entre componentes, y del cual extrae la medida de temperatura. Contempla un algoritmo para la generación de resultados y el etiquetado de los mismos.

Esta instancia modela el sensor que mide la temperatura del aire saliente del recalentador.

- **Conexiones del Módulo:**
  - **reheatingOffCoilTemp.i (conectado a reheatingCoil.po):**
    Clase tipo "PneumaticPort".

❖ Descripción de la Tabla inputs:

Identificador de entrada	Tipo	Conexión	Descripción
inputs[1]	Real	preheatingCoilValve.position	Valor de apertura de la válvula de entrada de líquido calentador a preheatingCoil por boilerPump
inputs[2]	Real	coolingCoilValve.position	Valor de apertura de la válvula de entrada de líquido refrigerante a coolingCoil por chillerPump
inputs[3]	Real	reheatingCoilValve.position	Valor de apertura de la válvula de entrada de líquido calentador a reheatingCoil por boilerPump

inputs[4]	Real	mixingBox.position	Valor de apertura de la válvula de entrada de aire a mixingBox por outside y returnFan
inputs[5]	Real	Outside.T	Valor de temperatura de aire exterior
inputs[6]	Real	Zone.T	Valor de temperatura de aire en sala aclimatada
inputs[7]	Real	supplyFan.mflow, supplyFan.mflow	Valor de flujo de aire saliente de los ventiladores
inputs[8]	Real	gasBoiler.outputTemperature	Valor de temperatura de aire de salida del calentador de agua
inputs[9]	Real	gasBoiler.inputTemperature	Valor de temperatura de aire de entrada del calentador de agua
inputs[10]	Real	chiller.outputTemperature	Valor de temperatura de aire de salida del enfriador de agua
inputs[11]	Real	chiller.inputTemperature	Valor de temperatura de aire de entrada del enfriador de agua

4.5.3. Componentes del modelo de sistema AHU-9 Simplificado

Estudio de los bloques funcionales básicos que definen el sistema AHU-9 Simplificado. Análisis de los datos de entrada, salida y medidas de cada componente. Estudio y comprensión de la estructura interna de cada bloque: ecuaciones, suposiciones y comportamientos.

Analizamos los valores de entrada, salida y medidas que se realizan dentro de los diferentes componentes básicos del sistema (en la documentación del nuevo sistema no se detallan las unidades que se emplean para cada magnitud; en el siguiente documento suponemos que las unidades de medida son equivalentes a las del modelo completo).

1. Mixing Box:

- Entradas:
  - Puerto Neumático pi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Neumático pr:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático po:

        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
  (No hay mediciones adicionales).
- Otros Datos:
  - Posición del Amortiguador (determina fuente de flujo de aire seleccionada de entrada – [0,1]).

2. Preheating Coil:

- Entradas:
  - Puerto Neumático pi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidraúlico hi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático po:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidráulico ho:
    - Temperatura del Aire (T -ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
  - Temperatura del Aire en po (T - ºC).
- Otros Datos:
  - hotSideCapacitanceRate.
  - coldSideCapacitanceRate.
  - minCapacitanceRate.
  - maxCapacitanceRate.
  - capacitanceRateRatio.
  - Efectividad (método *eff-NTU*).
  - Número de unidades de transferencia (NTU).

3. Colinging Coil:

- Entradas:
  - Puerto Neumático pi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidraúlico hi:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
  - Puerto Neumático po:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidráulico ho:
    - Temperatura del Aire (T -ºC).

        - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
    - Temperatura del Aire en po (T - ºC).
- Otros Datos:
    - hotSideCapacitanceRate.
    - coldSideCapacitanceRate.
    - minCapacitanceRate.
    - maxCapacitanceRate.
    - capacitanceRateRatio.
    - Efectividad (método *eff-NTU*).
    - Número de unidades de transferencia (NTU).

4. Reheating Coil:

- Entradas:
    - Puerto Neumático pi:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidraúlico hi:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Salidas:
    - Puerto Neumático po:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidráulico ho:
        - Temperatura del Aire (T -ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
- Mediciones:
    - Temperatura del Aire en Po (T - ºC).
- Otros Datos:
    - hotSideCapacitanceRate.
    - coldSideCapacitanceRate.
    - minCapacitanceRate.
    - maxCapacitanceRate.
    - capacitanceRateRatio.
    - Efectividad (método *eff-NTU*).
    - Número de unidades de transferencia (NTU)

4.5.4. Analisis y Estudio de las “Health Variables”

Análisis de las diferentes "health variables" detectadas en cada componente. Enumeración del conjunto de variables que definen a cada componente y selección de las "health variables" a partir de dicho conjunto, justificando su razón de ser.

A continuación se hace una revisión completa de los componentes que definen el sistema AHU-9 Simplificado. Por cada módulo detallamos las variables que definen el componente y las que consideramos como "health variables". Descartamos en la

enumeración de variables de componentes aquellas que son locales a las diferentes clases de componente, esto es, que solo tienen significado a nivel de programación:

1. **Mixing Box:**
   - Variables:
     - Puerto Neumático pi:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Puerto Neumático pr:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Puerto Neumático po:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Posición position: determina la posición de los amortiguadores para el control del flujo de aire dirigido parte al exterior, parte al interior del sistema.
   - Health Variables:
     - Posición position: única variable del módulo que modela comportamiento del mismo. Toma valores de la tabla de entrada, por lo que es susceptible de inducir fallos a través de dicho componente. Fallos en el componente pueden suponer una alteración del flujo de aire total circulante en el sistema.
   - Variables Descartadas:
     - outdoorFlow: variable interna que a través del producto del flujo entrante por la posición permite estimar el flujo de aire saliente del sistema.
     - returnFlow: variable interna que a través del producto del flujo entrante por la posición permite estimar el flujo de aire reciclado del sistema.

2. **Preheating Coil - Reheating Coil - Cooling Coil:**
   - Variables:
     - Puerto Neumático pi:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Puerto Hidraúlico hi:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Puerto Neumático po:
       - Temperatura del Aire (T - ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
     - Puerto Hidráulico ho:
       - Temperatura del Aire (T -ºC).
       - Ratio de Flujo de Masa (mflow – kg/s).
   - Health Variables:
     - Ninguna: del conjunto de variables que definen el módulo, solo los puertos modelan la estructura interna del sistema. Parte de este razonamiento se basa en que a nivel de programación, ninguna de las variables toma

valores de la tabla de entradas (es decr, no se puede inducir fallo a ningún componente).

- Variables Descartadas:
  - nonZeroHydraulicInputFlow: variable interna empleada para asegurar que el flujo de agua en el módulo nunca sea nulo.
  - nonZeroPneumaticInputFlow: variable interna empleada para asegurar que el flujo de aire en el módulo nunca sea nulo.
  - coldSideCapacitanceRate: variable interna con la que estimamos el ratio de capacitancia de calentamiento generado por el circuito hidráulico.
  - hotSideCapacitanceRate: variable interna con la que estimamos el ratio de capacitancia de enfriamiento generado por el circuito neumático.
  - NTU: variable interna empleada para el cálculo de la efectividad.
  - eff: variable interna con la que estimamos la efectividad.
  - minCapacitanceRate: variable interna con la que estimamos la menor de las capacitancias de enfriamiento y calentamiento.
  - maxCapacitanceRate: variable interna con la que estimamos la mayor de las capacitancias de enfriamiento y calentamiento.
  - capacitanceRateRatio: variable interna con la que estimamos el ratio de capacitancias.
  - heatRate: variable interna empleada para la estimación de la temperatura del flujo de salida de aire en el módulo.

3. supplyFan - returnFan:

- Variables:
  - Puerto Neumático i:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Neumático o:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Flujo mflow: determina el flujo de aire saliente del ventilador.
- Health Variables:
  - mflow: única variable del módulo que modela comportamiento del mismo. Toma valores de la tabla de entrada, por lo que es susceptible de inducir fallos a través de dicho componente. Fallos en el componente pueden suponer una alteración del flujo de aire total circulante en el sistema. Si falla un ventilador, el otro también falla, pues ambos toman la misma entrada.
- Variables Descartadas:
  - Ninguna.

4. gasBoiler:

- Variables:
  - Puerto Hidráulico i1:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidraúlico i1:

        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidraúlico o:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Temperatura outputTemperature: determina la temperatura a la que se expulsa el líquido del calentador.
- Health Variables:
    - outputTemperature: única variable del módulo que modela comportamiento del mismo. Toma valores de la tabla de entrada, por lo que es susceptible de inducir fallos a través de dicho componente. Fallos en el componente pueden suponer una alteración de la capacidad de calentamiento de las bobinas de precalentamiento y recalentamiento.

- Variables Descartadas:
    - Temperatura inputTemperature: a pesar de que toma valores de la tabla de datos, se descara por no ser empleada en la implementación del comportamiento del calentador.

5. Chiller:

- Variables:
    - Puerto Hidraúlico i:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidráulico o:
        - Temperatura del Aire (T -ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Temperatura outputTemperature: determina la temperatura a la que se expulsa el líquido del refrigerador.
- Health Variables:
    - outputTemperature: única variable del módulo que modela comportamiento del mismo. Toma valores de la tabla de entrada, por lo que es susceptible de inducir fallos a través de dicho componente. Fallos en el componente pueden suponer una alteración de la capacidad de enfriamiento de la bobina de enfriamiento.
- Variables Descartadas:
    - Temperatura inputTemperature: a pesar de que toma valores de la tabla de datos, se descara por no ser empleada en la implementación del comportamiento del refrigerador.

6. boilerPump - chillerPump:

- Variables:
    - Puerto Hidraúlico i:
        - Temperatura del Aire (T - ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).
    - Puerto Hidráulico o:
        - Temperatura del Aire (T -ºC).
        - Ratio de Flujo de Masa (mflow – kg/s).

- Flujo mflow: determina el flujo de líquido bombeado por el módulo.
- Health Variables:
  - Ninguna: mflow es la única candidata posible, pero dado que en el modelado del sistema contempla un valor de flujo constante, estos componentes no son susceptibles de generar fallos en el sistema, por lo que se descarta.
- Variables Descartadas:
  - Ninguna.

7. preheatingCoilValve - coolingCoilValve - reheatingCoilValve:

- Variables:
  - Puerto Hidraúlico i:
    - Temperatura del Aire (T - ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Puerto Hidráulico o:
    - Temperatura del Aire (T -ºC).
    - Ratio de Flujo de Masa (mflow – kg/s).
  - Posición position: determina la posición de los amortiguadores para el control del flujo de líquido dirigido a cada componente del sistema.
- Health Variables:
  - Position: única variable del módulo que modela comportamiento del mismo. Toma valores de la tabla de entrada, por lo que es susceptible de inducir fallos a través de dicho componente. Fallos en el componente pueden suponer una alteración del flujo de líquido calentador/refrigerante que recibe cada módulo, lo que supone una alteración de la capacidad de los mismos para la aclimatación del aire a la temperatura deseada.
- Variables Descartadas:
  - Ninguna.

Con el presente informe se han cubierto todos los componentes susceptibles de fallos en el sistema, y que por lo tanto presentaban posibles "health variables". Se ha descartado el estudio de los componentes representativos de la sala aclimatada y el exterior, dado que estos módulos modelan el contexto en el que nuestro sistema funciona, sin embargo no forman parte de él.

También descartamos analizar los sensores, ya que su implementación es tal que impide la presencia de fallos en dichos componentes.

4.6. Conclusiones

El sistema Air-Handling-Unit Nine (AHU-9) es un sistema de aire acondicionado instalado en la Escuela de Música de Cork. Su misión es la de aclimatar el aire procedente del exterior para ajustar la temperatura ambiente de las diferentes salas del recinto al valor deseado por los usuarios.

Su funcionamiento se resume en un bucle continuo por el que circula el aire desde el exterior hasta la sala aclimatada, atravesando una serie de componentes que modifican sus propiedades de temperatura y humedad ajustando las mismas a los valores especificados por

el usuario. Cuando el ciclo termina, parte del aire es expulsado al exterior, mientras que otra parte del mismo se recicla dentro del propio sistema, y vuelve a recorrer el bucle.

Estructuralmente el sistema se define por una serie de componentes de acondicionamiento básicos: un precalentador, un enfriador y un recalentador. Mediante estos tres elementos la temperatura del aire circulante por el circuito se ajusta a los valores deseados.

También existe un componente estructural básico, pero que no controla la temperatura del aire: el compartimento de mezcla. Este elemento se define como una cámara de aire gobernada por amortiguadores. En función de la posición de los mismos se efectúa un control de la proporción de aire entrante al sistema proveniente del exterior frente al aire reciclado del sistema.

Estos componentes se conectan a través de conductos y válvulas: los conductos llevan el aire desde el compartimento de mezcla a la sala; las válvulas conducen el líquido calefactor/refrigerante de las bombas a los componentes de acondicionamiento, de manera que estos pueden modificar la temperatura del aire por medio de estos líquidos.

La organización DXC ha puesto a nuestra disposición dos modelos del sistema AHU-9, el modelo complejo, reflejo de la estructura interna del sistema AHU-9; y el modelo simplificado, el cual reduce la complejidad estructural del modelo de sistema complejo al prescindir de las mediciones de humedad del aire en todo el sistema, y omitir ciertos módulos que ahora quedan obsoletos, como el humidificador.

Ambos modelos software han sido desarrollados en Modelica, un lenguaje de programación orientado a objeto diseñado especialmente para el modelado de sistemas complejos en base a componentes y conexiones.

Centrándonos en el modelo simplificado, el cual es el definitivo para la organización del concurso, el modelo emula un comportamiento equiparable al del sistema real en estado estacionario.

Para nuestro modelo solo son visibles los valores de temperatura asociados a cada uno de los tres componentes básicos a de aclimatación (las bobinas precalentadora, enfriadora y recalentadora). Esto nos lleva a considerar que los Posibles Conflictos asociados al sistema se definan como los subsistemas necesarios para la estimación de los valores de dichas temperaturas, al ser las únicas observaciones del sistema, de las cuales vamos a poder calcular los residuos.

A lo largo del capítulo se han descrito las configuraciones de cada modelo y las posibles variables susceptibles de inducir fallo al sistema.

Con toda esta información estamos en disposición de elaborar los PCs asociados al sistema y, en consecuencia, iniciar la labor de diagnosis.

En el siguiente capítulo se detallará todo este proceso, contemplando cómo hemos sido capaces de elaborar los PCs a partir de la descripción del sistema y como somos capaces de

reescribir el comportamiento del mismo en un código funcional más simplificado y manejable para nuestros intereses.

Capítulo 5:

Caso de Estudio

5.1. Introducción

En el siguiente capítulo procederemos a exponer toda la documentación referente a la labor realizada para la diagnosis del sistema AHU-9.

Aplicando los principios teóricos expuestos a lo largo del documento, vamos a describir los pasos seguidos para poder diagnosticar el sistema de estudio considerado. Se considerarán diferentes casos hipotéticos de funcionamiento, formulados por los propios investigadores.

El capítulo recoge la documentación, el remodelado del sistema software puesto a nuestra disposición por la organización del DXC, la aplicación de las técnicas de diagnosis basada en PCs sobre el mismo y los resultados obtenidos.

Adicionalmente se adjuntan tablas de datos, gráficas y descripciones de los modelos empleados para la diagnosis, junto con la explicación de cada uno de ellos. La idea es exponer con claridad cómo se efectuó la labor de diagnóstico justificando cada uno de los pasos seguidos y explicando las conclusiones que se derivan de los resultados.

5.2. Presentación del Sistema

5.2.1. Introducción

En el capítulo 4 ya se hizo un estudio completo del sistema AHU-9, que consideramos como nuestro caso de estudio particular. Se estudió la configuración del sistema real, su modelo de sistema, desarrollado en Modelica; y su revisión simplificada.

Entendemos el funcionamiento del sistema, su composición, las ecuaciones que rigen su comportamiento y la información que podemos extraer del mismo (las mediciones de temperatura obtenidas por los sensores).

Sin embargo, trabajar con dicho modelo para nuestro fin supone invertir un esfuerzo adicional a la hora de simular debido a la complejidad del código del sistema y el lenguaje en el que está desarrollado.

El estudio a realizar exige trabajar con un modelo al que se le puedan inducir fallos intencionadamente. Nuestra labor parte de la simulación del modelo de sistema en diversos casos hipotéticos donde falla alguno de los componentes del sistema. A partir de dicha información, y trabajando con nuestra técnica de diagnóstico, buscamos determinar si somos capaces de detectar y localizar dichos fallos.

Esto requiere trabajar con un modelo flexible que permita reconfigurar la simulación del sistema a nuestro antojo, esto es, poder inducir fallos a cualquier componente en cualquier instante de simulación.

Con dicha idea en mente, el primer paso para el diagnóstico de nuestro sistema pasa por programar el modelo de sistema en una plataforma software flexible donde podamos inducir de forma automática fallos al modelo.

5.2.2. Reescritura del sistema AHU-9 a una notación funcional

El modelo de sistema AHU-9 está programado en Modelica. Esto supone un cierto problema a la hora de trabajar con dicho modelo para el diagnóstico. Modelica es un lenguaje de programación orientado a objetos y concebido para el modelado de sistemas.

Dicho esto, y atendiendo a trabajos anteriores [Hernández2012]; se considera que Matlab es la plataforma software adecuada para desempeñar nuestra labor de diagnóstico. Matlab permite trabajar de manera flexible con datos e información. Soporta tanto programación funcional como orientada a objetos, y dispone de una gran flexibilidad en el tratamiento de ficheros de datos. En nuestro caso, los ficheros de entrada de datos son compatibles con Matlab, y en esta herramienta existe la posibilidad de programar un modelo de sistema con capacidad para inducir fallos.

Trabajar en Matlab supone el esfuerzo adicional de reescribir nuestro modelo de sistema AHU-9 a un código fuente ejecutable en dicha plataforma.

Para efectuar esta reimplementación del modelo se efectuó una labor de documentación y revisión del modelo simplificado del sistema AHU-9. Dicha labor buscaba determinar cómo reescribir el código completo del sistema en una notación funcional (sin clases) y lo más simplificada posible para poder efectuar simulaciones del mismo que generasen resultados equivalentes a los del modelo en Modelica.

Es necesario saber discernir entre lo que es “modelo de sistema” y lo que es “modelo de simulación”. Para nuestros intereses, lo que buscamos obtener es un “modelo de simulación”. Mediante el modelo de simulación buscamos disponer de un software que emule el comportamiento del sistema de estudio, pero limitado a la estimación de las magnitudes observables del mismo, esto es, las mediciones de los sensores de temperatura.

El “modelo de sistema” se configura para dar un esquema aproximado de la descripción del sistema real que abstrae; es con lo que hemos estado trabajando hasta el momento. Este modelo intenta reflejar la estructura completa del sistema y su comportamiento, que vendría a ser el ajuste de temperatura óptimo del atributo “T” de la instancia de la clase “Zone” que modela la sala aclimatada.

Dicho esto, el primer paso de la reescritura es generar el modelo de simulación a partir del modelo de sistema. Este paso se llevó a cabo realizando las siguientes tareas:

- *Reinterpretación y simplificación del sistema:*

  El primer paso para generar el modelo de simulación es analizar el modelo de sistema de partida. La idea inicial de esta tarea es entender el comportamiento del sistema y determinar los componentes del mismo que resultan innecesarios para efectuar la simulación.

  Estamos trabajando con un modelo software que emula el comportamiento real del sistema. El modelo de sistema se caracteriza por definir una configuración virtual del sistema real que modela, equivalente a la del mismo. Esto supone que en su implementación se declaren componentes que existen en el sistema real, pero que a nivel de programación resultan innecesarios.

  Para dar a entender esta idea vamos a listar los componentes despreciables de nuestro modelo de simulación, y la justificación de porqué se considera a cada uno de ellos como innecesario:

  - **Zone:** componente despreciable en el modelo de simulación dado que resulta irrelevante para la estimación de la temperatura en los sensores (la temperatura de la zona, además, coincide con la temperatura medida por el sensor de temperatura conectado al bucle de recalentamiento).

    El componente está presente en el modelo de sistema para modelar la sala aclimatada a la que el sistema dirige el flujo de aire.

  - **SupplyFan:** componente despreciable en el modelo de simulación dado que resulta irrelevante para la estimación de la temperatura en los sensores (la temperatura del ventilador, además, coincide con la temperatura medida por el sensor de temperatura conectado a la bobina de recalentamiento).

    El componente está presente en el modelo de sistema para modelar el ventilador que dirige el flujo de aire de la bobina de recalentamiento a la sala aclimatada.

  - **Conectores de realimentación de bobinas y circuito de aire:**

    El sistema en cuestión se compone de tres tipos de bobinas: bobina de precalentamiento, enfriamiento y recalentamiento. Las bobinas contemplan un doble flujo, de aire y masa, dentro de su estructura.

    En particular el flujo de masa proviene de una serie de bombas que acondicionan previamente la temperatura del líquido en circulación; y luego envían el mismo a la bobina. De esta manera se consigue inducir la temperatura deseada al flujo de aire circundante por el componente.

    La salida de masa de la bobina se conecta con las bombas de masa. Esta realimentación es despreciable, pues a nivel de programación solo nos interesa

lidiar con el flujo entrante de masa a cada bobina; y éste está programado de tal manera que es independiente de la realimentación de masa en las bombas.

Esta independencia se consigue debido a que los valores de temperatura y flujo de masa son valores de entrada al modelo de sistema, por lo que no mantienen ninguna relación con el flujo de masa que se recicla desde las bobinas a la bomba.

Esto mismo ocurre con el flujo de aire aclimatado presente en nuestro sistema. Como sabemos, nuestro sistema se podría concebir como un circuito por el que circula el aire, en el que parte de él es reciclado una vez completa todo el circuito, repitiendo el mismo recorrido.

Aunque a nivel estructural el aire reciclado podría influir en la temperatura del flujo entrante al sistema, el sistema está programado de tal manera que las propiedades de dicho flujo reciclado se concibe como una entrada de datos, lo que supone que se pueda prescindir de los conectores de aire que van de la sala al ventilador de salida.

Al final de todo este razonamiento podemos hablar de un esquema de modelo de simulación del sistema AHU-9 donde se prescinde de los elementos que a nivel de programación resultan innecesarios y solo generan redundancia en el sistema. En la siguiente figura se da una idea de lo que sería el mismo:

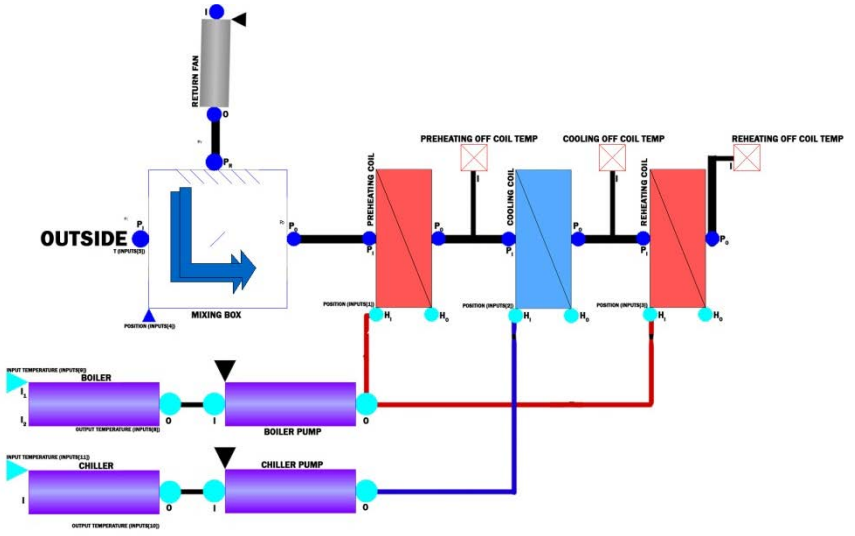


Figura 5.1 Modelo de simulación del sistema AHU-9

- *Descripción del comportamiento secuencial del sistema:*

  Definida la configuración inicial de nuestro modelo de simulación, el siguiente paso consistiría en modelar el comportamiento del mismo. Ahora trabajamos con código funcional, por lo que prescindimos de clases y componentes.

  La idea es definir una función completa que represente el comportamiento interno del modelo componente a componente. Esto supone escribir de manera ordenada las ecuaciones que rigen el ciclo del aire dentro del sistema.

  Este comportamiento se puede inferir fácilmente de la secuencia de conexiones del modelo de sistema y las ecuaciones que definen el comportamiento de cada componente.

  Además, la lectura del código del modelo de simulación revela aspectos de utilidad en la programación de los componentes del modelo, lo que hace más sencilla la reprogramación del código:

  - **Flujo Constante:** el flujo de aire que circula por el sistema es constante, desde el compartimento de mezcla hasta la bobina de recalentamiento.

    Además de ello, en el compartimento de mezcla, donde hipotéticamente el flujo de salida es una mezcla de los dos flujos de aire entrantes (exterior y reciclado); el flujo saliente es exclusivamente de aire reciclado.

    Esto hace que todas las funciones que modelan el comportamiento del sistema vayan destinadas a la estimación de temperatura del flujo de aire en cada instante, lo que simplifica su programación.

  - **Componentes equivalentes:**

    Aunque cada bucle de acondicionamiento se define como instancia de una clase propia, todas estas clases comparten una misma implementación. Es decir, las bobinas son iguales.

    Como resultado de ello, modelar cada uno de los bucles sigue un mismo elenco de ecuaciones; que se puede introducir dentro de una función genérica para la estimación de temperatura en cada bobina.

    Algo similar ocurre con las bombas de masa, que son idénticas. En el caso del calentador y el refrigerador, su composición interna es diferente dado que el calentador contempla dos puertos hidráulicos de entrada en su composición, frente al puerto de entrada único del refrigerador.

    Sin embargo, como en nuestro modelo de simulación prescindimos de dichos puertos, la estructura de ambas clases puede redefinirse de manera que sean

idénticas, trabajando con una única clase que los modele. Es decir, podemos replicar el comportamiento de los dos componentes acondicionadores del flujo de masa mediante un conjunto de ecuaciones común.

La conclusión a la que se llega con estos datos derivados de la comprensión del código fuente del modelo de sistema es que hay ciertas partes de su estructura redundantes. Se han declarado clases cuyo comportamiento es idéntico, mientras que otras solo difieren a nivel estructural por el número de puertos de entrada/salida que contemplan.

Si se eliminan estas redundancias vemos que el modelo de simulación queda mucho más simplificado, esencialmente debido a que se pueden agrupar las ecuaciones que definen el comportamiento de los componentes en funciones. Dichas funciones pueden ser empleadas por varios componentes del sistema, por lo que al final la programación del modelo se va a resumir en la conexión ordenada de los componentes para definir el comportamiento del sistema AHU-9.

A continuación exponemos las ecuaciones que rigen nuestro modelo de simulación. Lo que podemos ver es un modelo monolítico simplificado del modelo monolítico del sistema, contenido en el glosario de esta memoria.

Lo que se ha hecho para construir el modelo monolítico es ordenar las funciones del sistema de manera que su ejecución ordenada permita la estimación de todos los parámetros del sistema.

Aplicando al mismo las simplificaciones anteriormente mencionadas y sustituyendo las variables intermedias del modelo por expresiones en las que solo intervienen variables de entrada se obtiene un modelo de simulación simplificado:

```
---------------------------------------------------------------------------------
nonZeroPneumaticInputFlow = max(abs(returnFan.o.mflow), 0.001);
nonZeroHydraulicInputFlow = max(abs(boilerPump.o.position * 1.5), 0.001);
---------------------------------------------------------------------------------
hotSideCapacitanceRate = nonZeroHydraulicInputFlow * hydraulicSpecificHeat;
coldSideCapacitanceRate = nonZeroPneumaticInputFlow * pneumaticSpecificHeat;
---------------------------------------------------------------------------------
minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);
capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
---------------------------------------------------------------------------------
NTU = UA / minCapacitanceRate;
eff = effectiveness(HeatExchangerConfiguration.counterFlow, capacitanceRateRatio, NTU);
heatRate = eff * minCapacitanceRate * (gasBoiler.outputTemperature - ( (outdoorFlow * outside.T +
returnFlow * returnFan.o.T)/returnFan.o.mflow ) );
---------------------------------------------------------------------------------
preheatingOffCoilTemp.i.T = (heatRate/coldSideCapacitanceRate) + ( (outdoorFlow * outside.T +
returnFlow * returnFan.o.T)/returnFan.o.mflow );
---------------------------------------------------------------------------------
nonZeroPneumaticInputFlow = max(abs(returnFan.o.mflow), 0.001);
nonZeroHydraulicInputFlow = max(abs(chillerPump.o.position * 1.5), 0.001);
---------------------------------------------------------------------------------
hotSideCapacitanceRate = nonZeroHydraulicInputFlow * hydraulicSpecificHeat;
coldSideCapacitanceRate = nonZeroPneumaticInputFlow * pneumaticSpecificHeat;
---------------------------------------------------------------------------------
```

```
minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);
capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
---------------------------------------------------------------------------
NTU = UA / minCapacitanceRate;
eff = effectiveness(HeatExchangerConfiguration.counterFlow, capacitanceRateRatio, NTU);
heatRate = eff * minCapacitanceRate * (chiller.outputTemperature - preheatingOffCoilTemp.i.T);
---------------------------------------------------------------------------
reheatingOffCoilTemp.i.T = (heatRate/coldSideCapacitanceRate) + coolingOffCoilTemp.i.T;
---------------------------------------------------------------------------
nonZeroPneumaticInputFlow = max(abs(returnFan.o.mflow), 0.001);
nonZeroHydraulicInputFlow = max(abs(boilerPump.o.position * 1.5), 0.001);
---------------------------------------------------------------------------
hotSideCapacitanceRate = nonZeroHydraulicInputFlow * hydraulicSpecificHeat;
coldSideCapacitanceRate = nonZeroPneumaticInputFlow * pneumaticSpecificHeat;
---------------------------------------------------------------------------
minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);
capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
---------------------------------------------------------------------------
NTU = UA / minCapacitanceRate;
eff = effectiveness(HeatExchangerConfiguration.counterFlow, capacitanceRateRatio, NTU);
heatRate = eff * minCapacitanceRate * (boilerPump.o.T - coolingOffCoilTemp.i.T);
---------------------------------------------------------------------------
reheatingOffCoilTemp.i.T = (heatRate/coldSideCapacitanceRate) + coolingOffCoilTemp.i.T;
---------------------------------------------------------------------------
```

Como se puede ver, el número de ecuaciones que define el sistema es bastante pequeño; lo que ocurre es que dichas ecuaciones se repiten en varias ocasiones (tres para ser exactos, una por cada bobina).

A partir de estas funciones la programación del modelo de simulación es directa, como veremos a continuación.

- *Escritura y simplificación del modelo de simulación:*

Escribir el modelo de simulación en Matlab supone transcribir todas las ecuaciones descritas anteriormente a un único fichero ".m" con el que se modela el sistema completo.

Para facilitar el código, hemos eliminado todas las ecuaciones asociadas al cálculo de la temperatura en bucles, y las hemos introducido en una función auxiliar. De esta manera el código del sistema queda mucho más simplificado, y menos redundante.

El sistema final puede ser a su vez simplificado. En este modelo eliminamos por completo toda mención asociada a las bombas de masa, los módulos calentadores y refrigeradores, y el compartimento de mezcla, dejando solo en el sistema las ecuaciones asociadas a las bobinas.

Los componentes asociados al control de flujo de masa determinan la temperatura y ratio de dicho flujo a partir de entradas, por lo que sustituyendo su declaración por la ruta de valores de entrada podemos omitir ciertas líneas de

código redundantes (la redundancia de conectar las entradas a dichos componentes, en vez de omitir estos y usar las entradas directamente).

El compartimiento de mezcla requiere de un cierto estudio de las ecuaciones que modelan su comportamiento:

$$outdoorFlow = pr.mflow * position;$$
$$returnFlow = pr.mflow * (1 - position);$$
$$po.mflow = outdoorFlow + returnFlow;$$
$$po.mflow * po.T = outdoorFlow * pi.T + returnFlow * pr.T;$$

Analizando dichas ecuaciones, nos interesa estimar los valores de *po.mflow* y *po.T*; esto es, la temperatura y el ratio de flujo del aire saliente por el puerto hidráulico de salida *po*.

Siguiendo el orden lógico de estimación de dichas ecuaciones, el primer paso sería estimar *po.mflow*. Esta variable se estima a partir de la suma de flujos entrantes condicionada por la posición del amortiguador, que controla la proporción de aire entrante de cada tipo. Si sustituimos las variables *outdoorFlow* y *returnFlow* por su expresión, se obtiene la siguiente secuencia de expresiones:

$$po.mflow = pr.mflow * position + pr.mflow * (1 - position)$$
$$po.mflow = pr.mflow * (position + (1 - position))$$
$$po.mflow = pr.mflow$$

Ya se mencionaba anteriormente, pero ahora podemos ver justificadamente que el flujo de salida del compartimento de mezcla no es más que el flujo de aire reciclado del sistema.

Recordando lo dicho anteriormente, este flujo es constante a lo largo de todo el circuito de tratamiento del aire, por lo que el flujo reciclado es el que marca el flujo de aire en todos los componentes de nuestro sistema de simulación.

Dicho esto, la estimación de la temperatura del flujo de aire saliente se puede calcular manipulando las siguientes fórmulas:

$$po.mflow * po.T = outdoorFlow * pi.T + returnFlow * pr.T$$
$$po.mflow * po.T = (pr.mflow * position) * pi.T + (pr.mflow * (1 - position)) * pr.T$$
$$pr.mflow * po.T = (pr.mflow * position) * pi.T + (pr.mflow * (1 - position)) * pr.T$$
$$po.T = position * pi.T + (1 - position) * pr.T$$

Con esto se consigue demostrar que la temperatura del flujo de aire de salida es independiente del ratio de flujo de los flujos de aire entrantes.

Con todo ello, la conclusión final a la que se llega es que en la descripción del modelo de simulación, el compartimento de mezcla presenta también un comportamiento que permite omitir al mismo por expresiones que se reduce a una fórmula algebraica sin dinámica.

Con todo ello, nuestro modelo de simulación final se define como tres bobinas de acondicionamiento cuyo comportamiento se rige por un mismo conjunto de ecuaciones, las cuales pueden agruparse en una misma función.

Así el código fuente final de nuestro modelo de simulación es el siguiente:

```
mixing_box_po_T  = ((MixingBoxPosition * SystemFlow) * OutsideTemp + ((1 -
MixingBoxPosition) * SystemFlow) * ZoneTemp)/SystemFlow;
mixing_box_po_mflow = SystemFlow;

PreheatingTemp  = mixingBox(mixing_box_po_mflow,mixing_box_po_T,(PreheatingValve *
1.5),BoilerTemp);
preheating_po_mflow = mixing_box_po_mflow

CoolingTemp = mixingBox(preheating_po_mflow,PreheatingTemp,(CoolingValve *
1.5),ChillerTemp);
cooling_po_mflow = preheating_po_mflow;

ReheatingTemp = mixingBox(cooling_po_mflow,CoolingTemp,(ReheatingValve * 1.5),BoilerTemp);
reheating_po_mflow = Cooling_po_mflow;
```

Y la función que modela el comportamiento de las bobinas:

```
function temp = coil(pi_mflow,pi_T,hi_mflow,_hi_T)
  nonZeroPneumaticInputFlow = max(abs(pi_mflow), 0.001);
  nonZeroHydraulicInputFlow = max(abs(hi_mflow), 0.001);

  hotSideCapacitanceRate = nonZeroHydraulicInputFlow  * waterSpecificHeat;
  coldSideCapacitanceRate = nonZeroPneumaticInputFlow * airSpecificHeat;

  minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
  maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);

  capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
  NTU = UA / minCapacitanceRate;

  if capacitanceRateRatio != 1 then
    eff = (1 - exp(-NTU * (1 - capacitanceRateRatio))) / (1 - capacitanceRateRatio * exp(-NTU * (1 -
capacitanceRateRatio)));
  else
    eff = NTU / (1 + NTU);
  end;

  heatRate = eff * minCapacitanceRate * (hi_T - pi_T);
  temp = (heatRate/coldSideCapacitanceRate) + pi_T;
end
```

Este modelo va a ser el empleado para la generación de experimentos a partir de su simulación; y para la posterior construcción de los PCs.

Como puede verse, si se efectúa un estudio completo de los sistemas reales con los que trabajamos y sus modelos software, es posible simplificar los mismos hasta casos donde solo trabajamos modelando aquellas partes del sistema cuyas estimaciones nos resultan de interés.

Se trata de un esfuerzo dedicado a simplificar los modelos y trabajar de manera más eficiente; el cual se ve recompensado una vez se llega a nuestra etapa final de simulación del modelo y generación de PCs, donde la mayor simplificación del modelo supone una reducción considerable de esfuerzo en la labor de diagnóstico.

En nuestro caso ese esfuerzo nos lleva a trabajar con un modelo de apenas 20 líneas frente a las 257 que presenta el modelo de sistema simplificado.

5.3. Ecuaciones del Sistema Completo y de los PCs

Aplicando los conocimientos teóricos expuestos referentes a la identificación de PCs asociados a un sistema, se ha detectado que el AHU-9 presenta un total de tres posibles conflictos.

Cada uno de estos posibles conflictos va asociado a la estimación de los valores de temperatura medidos por cada uno de los tres sensores que componen el sistema a estudiar. A nivel de implementación, la definición de cada PC se asocia con las tres etapas diferentes del ciclo de tratamiento del aire, representadas en los tres bucles de aclimatamiento que definen la estructura interna del sistema.

Además de la definición de cada PC, también hay que indicar los componentes asociados a los mismos; esto es, los elementos de nuestro sistema que se consideran candidatos de estar funcionando bajo fallo cuando el PC al que corresponden se activa.

En esta sección procedemos a definir cada uno de los PCs contemplados, su MEM y los componentes candidatos a fallo asociados a los mismos:

- **PC1:**

  El PC1 permite calcular los residuos asociados a la discrepancia entre observaciones y estimaciones del sensor de temperatura conectado al bucle de precalentamiento.

  Su estructura interna se define con las siguientes ecuaciones:

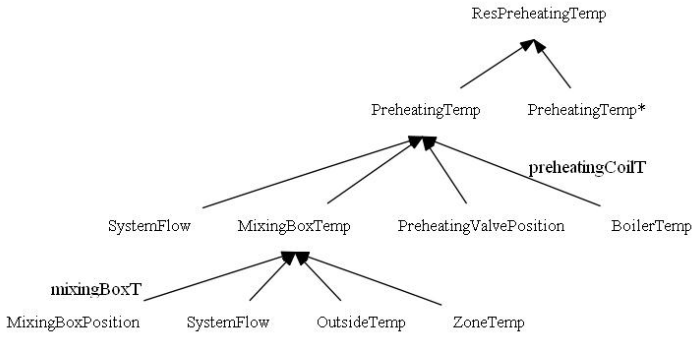


Figura 5.2 RBD asociada al PC1

- **PC2:**

El PC2 permite calcular los residuos asociados a la discrepancia entre observaciones y estimaciones del sensor de temperatura conectado al bucle de precalentamiento.

Su estructura interna se define con las siguientes ecuaciones:

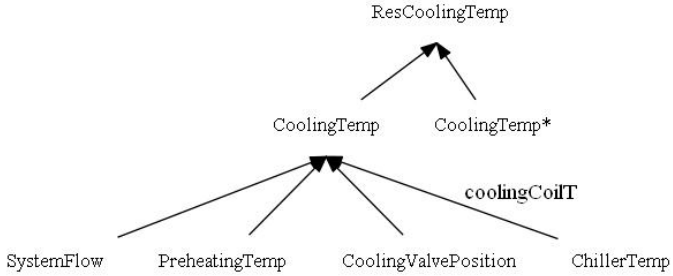


Figura 5.3 RBD asociada al PC2

- **PC3:**

El PC3 permite calcular los residuos asociados a la discrepancia entre observaciones y estimaciones del sensor de temperatura conectado al bucle de recalentamiento.

Su estructura interna se define con las siguientes ecuaciones:

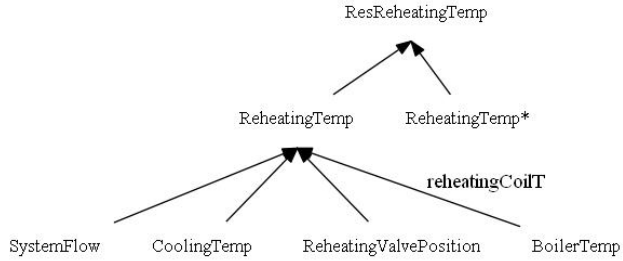


Figura 5.4 RBD asociada al PC3

Para terminar se muestra la matriz de firmas teóricas. En ella indicamos por cada PC los parámetros de fallo asociados (marcados con un 1):

Matriz de Firmas Teóricas										
	MB Valve	PC Valve	PC UA	PC Sensor	CC Valve	CC UA	CC Sensor	RC Valve	RC UA	RC Sensor
PC1	1	1	1	1	0	0	0	0	0	0
PC2	0	0	0	1	1	1	1	0	0	0
PC3	0	0	0	0	0	0	1	1	1	1

5.4. Proceso de Diagnóstico del sistema AHU-9:

5.4.1. Detección de fallos

La detección de fallos en el sistema se hace a partir de los PCs. Los PCs permiten efectuar los pasos necesarios marcados por la DBM, permitiendo la detección, aislamiento e identificación de fallos.

Desde la introducción de los PCs hemos explicado que la activación de los mismos se hace en base a la estimación de residuos entre las observaciones extraídas del modelo de sistema real. Cuando el residuo supera un cierto umbral (ajustado para distinguir entre errores y ruido en los sensores) el PC se activa y consideramos el conjunto de componentes asociados al PC como posibles candidatos a sufrir fallo.

En presencia de ruido, una vez activado un residuo se suele aplicar una lógica de decisión para considerar si es una activación real o un pico en la señal debido al ruido. Para evitar este problema se puede usar un test estadístico en lugar de la simple comparación del valor del residuo con un umbral o bien complementarlo con alguna heurística. En nuestro caso vamos a emplear test estadístico *z-test*.

Empleamos el *ztest* sobre el residuo obtenido de la diferencia entre valor estimado por la red de cada PC para las variables observadas y el valor experimental obtenido, generado por el modelo real de sistema.

Mediante el *ztest* comprobamos si dos intervalos de una señal siguen una misma distribución (esto es, mismos parámetros de media y desviación típica). La activación de los PCs tiene lugar cuando el *ztest* verifica que dichos intervalos no responden a una misma distribución.

La estimación de los parámetros de media μ y desviación típica σ atiende a sus fórmulas tradicionales:

$$\hat{\mu}_{N_2}(k) = \frac{1}{N_2} \sum_{i=k-N_2+1}^{k} r(i)$$

$$\hat{\sigma}^2_{N_1}(k) = \frac{1}{N_1 - 1} \sum_{i=k-N_1+1}^{k} (r(i) - \mu_{N_1}(k))^2$$

El empleo del *ztest* exige considerar una serie de tamaños muestrales de intervalos de señal, el *delay* o retraso entre dichos intervalos, y el nivel de significación en la toma de decisión del test. En nuestro caso dichos valores son:

- **Longitud del primer intervalo (*N1*):** 40
- **Longitud del segundo intervalo (*N2*):** 20
- **Retraso entre intervalos (*delay*):** 10
- **Alfa (α):** 0.05

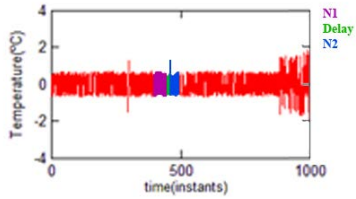


Figura 5.5 Ejemplo de ventana deslizante definida por N1-Delay-N2

5.4.2. Escenarios de fallo

Llegados a este punto estamos en condiciones de efectuar la labor de diagnóstico del sistema. A continuación vamos a exponer los diferentes escenarios de simulación considerados y los fallos detectados. En ellos se ha contemplado tanto el funcionamiento del sistema en modo nominal como el funcionamiento con fallo.

Trabajaremos con escenarios nominales y de fallo en los que se contemplarán cada uno de los posibles fallos asociados a componentes de cada PC. Cada fallo se repetirá en diferentes escenarios, y con diferentes magnitudes de error (5%, 20% y 50% en nuestro caso).

En el modo nominal, el sistema trabaja de forma correcta, en ausencia de fallos, y con entrada variable. Estas simulaciones permiten al investigador ajustar los parámetros del *ztest* para efectuar la labor de diagnóstico.

En los modos de fallo buscamos comprobar si las técnicas de diagnóstico aplicadas son efectivas. Estos modos son los relevantes para nuestra labor, y son los que se reflejarán en nuestra memoria.

Lo relevante es determinar para cada escenario la detección del fallo por los PC, el instante de activación de los mismos y los componentes seleccionados como candidatos a fallo. Esta será la información que se expondrá a continuación. Consideramos oportuno también el adjuntar gráficos orientativos de los resultados a modo de apoyo visual para el lector, de manera que se justifiquen de manera gráfica las conclusiones expuestas en la memoria.

La simulación del sistema contemplará un total de 2882 instantes de tiempo. Utilizaremos para los ejemplos visuales un mismo fichero de entrada, puesto a nuestra disposición por parte de la organización de la DXC. Para el resto de simulaciones utilizaremos ficheros con entradas variables, para garantizar que los resultados del diagnóstico son fiables.

La diagnosis del sistema se realizará por medio de Matlab. Hemos implementado un software sencillo para la generación automática de experimentos y diagnosis. Introduciendo al software las entradas, el sistema es capaz de trabajar con el modelo de AHU-9, induciendo configuraciones de fallo al mismo, para la generación de diferentes escenarios de funcionamiento.

Cada uno de estos escenarios sirve como entrada a su vez al módulo de diagnóstico del software, que opera con los PCs para la generación de residuos y la aplicación del *ztest* sobre los mismos.

La información de código software y resultados experimentales se adjunta a la memoria en los anexos y el soporte CD disponible al final de la misma.

- **Escenario de Funcionamiento en Modo Nominal:**

  El modo de funcionamiento en modo nominal se emplea para ajustar los parámetros del *ztest*. Como veremos, para ciertos escenarios de fallo, los parámetros del test no sirven para detectar el fallo y hay que reajustarlos. Cuando se escogen los N1, N2 y el delay hay que asegurarse que éstos no activan los PCs en modo nominal.

  A continuación vamos a exponer los resultados obtenidos de la simulación del sistema en modo nominal, para las tres variables de salida:

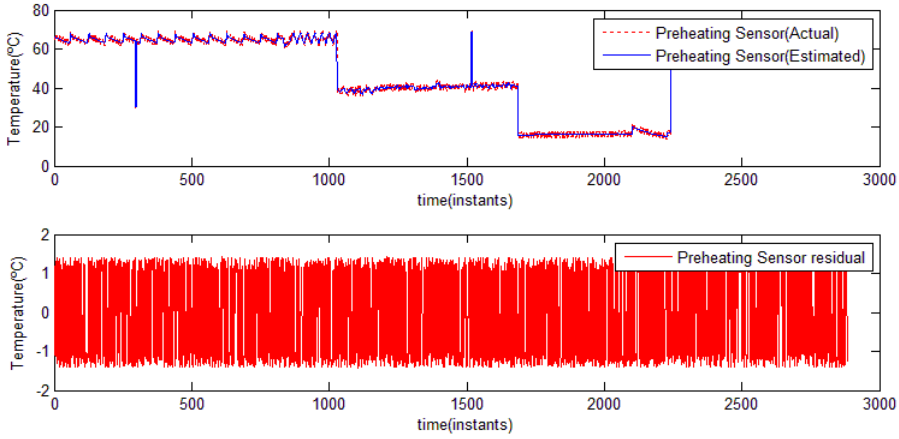


Figura 5.6 Escenario Nominal (Preheating Sensor)

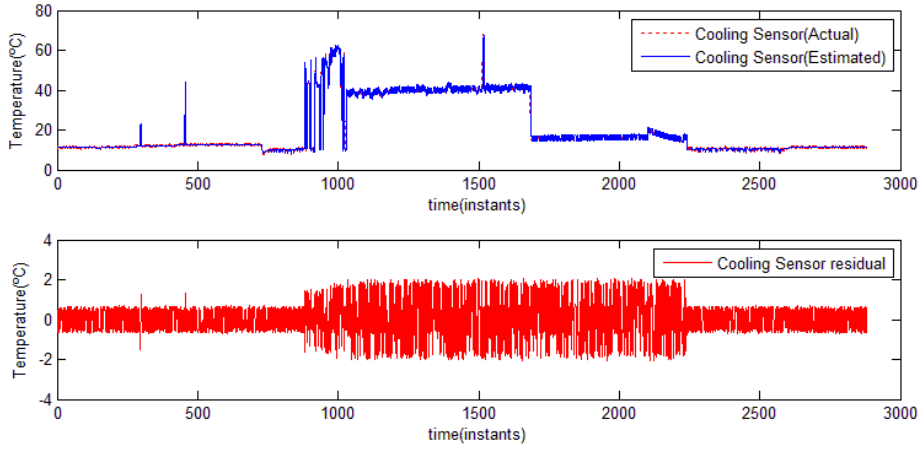


Figura 5.7 Escenario Nominal (Cooling Sensor)

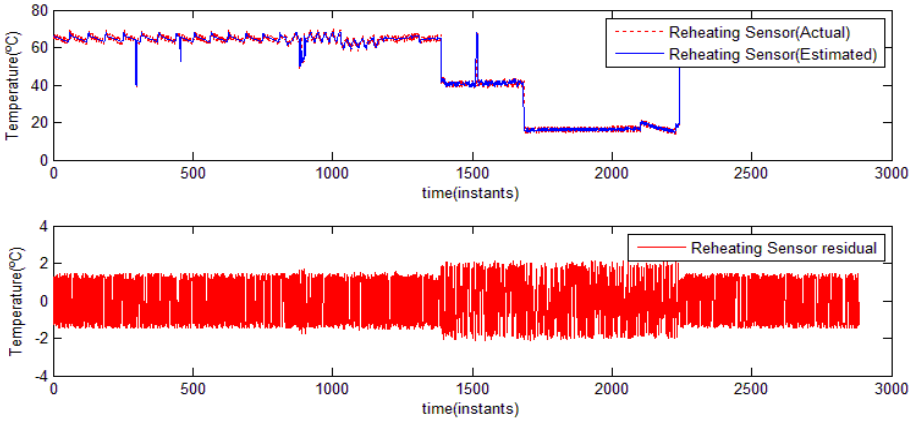


Figura 5.8 Escenario Nominal (Reheating Sensor)

Lo que vemos en las gráficas es el resultado de superponer la salida real de los sensores del sistema AHU-9 con las estimaciones del modelo. Las gráficas inferiores muestran el residuo obtenido de la diferencia de valores entre estimación y medición.

Como puede verse, en el funcionamiento nominal los residuos obtenidos son debidos a los sensores. Estos residuos alcanzan un valor máximo de 2ºC aproximadamente; además hablamos de un promedio de variación de temperatura de unos 65ºC, lo cual lleva asociado que estamos tratando con residuos con valores entorno al 4% de dicha variación. Esto puede suponer que a la hora de probar fallos del 5% de magintud pueda no ser posible detectar los mismos. Con estos valores y los parámetros por defecto considerados para el diagnóstico se ha comprobado que ningún PC se activa.

- **Escenario de Fallo 1: Fallo en Válvulas Hidráulicas CC_Valve**

Los fallos en válvulas hidráulicas ocurren cuando la posición de las válvulas que conducen el líquido calefactor o refrigerante a los bucles se atascan. Se trata de fallos permanentes.

El nivel de posicionamiento de válvulas se modela en torno a un cierto porcentaje; y en el funcionamiento nominal del sistema, dicho porcentaje es siempre del 0% (válvula totalmente abierta).

A continuación vamos a exponer los resultados de un escenario de fallo asociados a la válvula del bucle de enfriamiento; con tres magnitudes de fallo diferentes. Todas estas configuraciones tienen en común que el instante en el que se induce el fallo es el 1000. Según la matriz de firmas teóricas, el PC activado debería ser PC2:

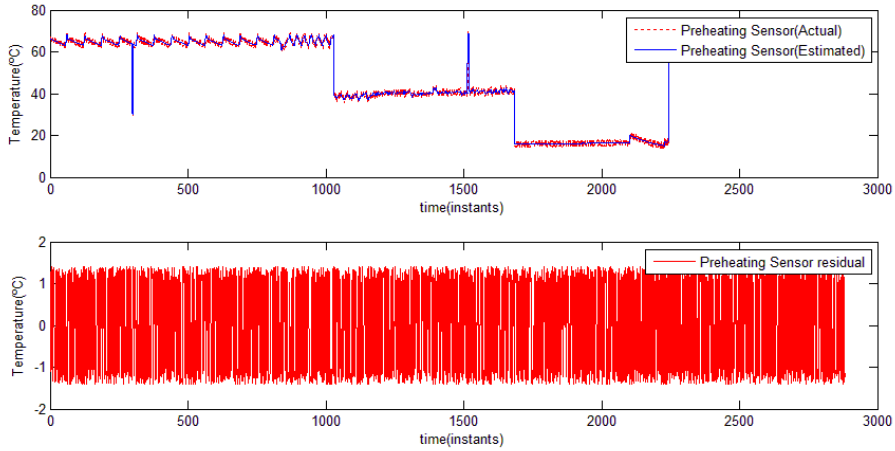


Figura 5.9 Escenario de Fallo CC Valve 5% (Preheating Sensor – PC1)

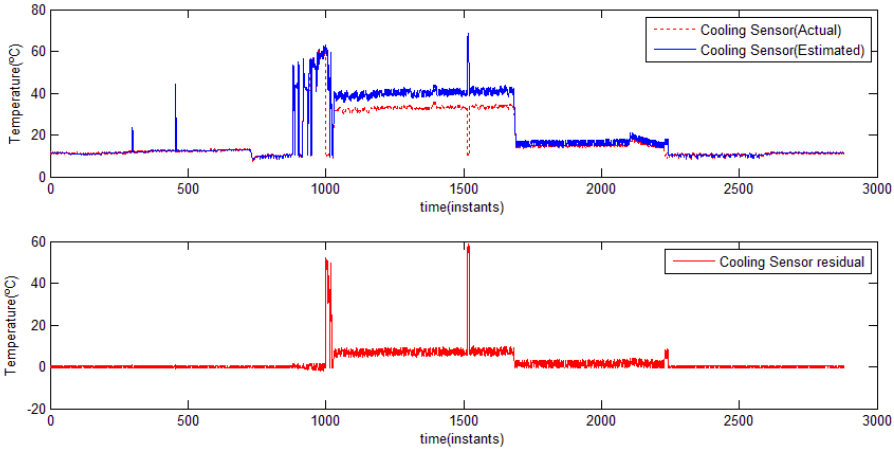


Figura 5.10 Escenario de Fallo CC Valve 5% (Cooling Sensor – PC2)

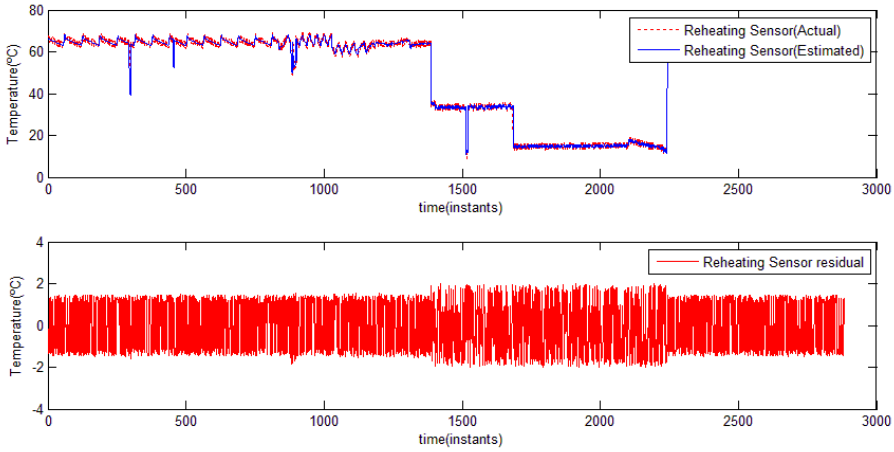


Figura 5.11 Escenario de Fallo CC Valve 5% (Reheating Sensor – PC3)

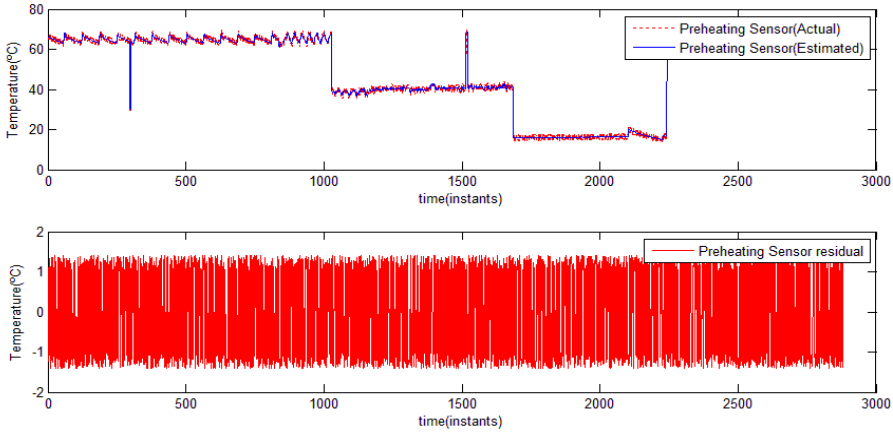


Figura 5.12 Escenario de Fallo CC Valve 20% (Preheating Sensor – PC1)

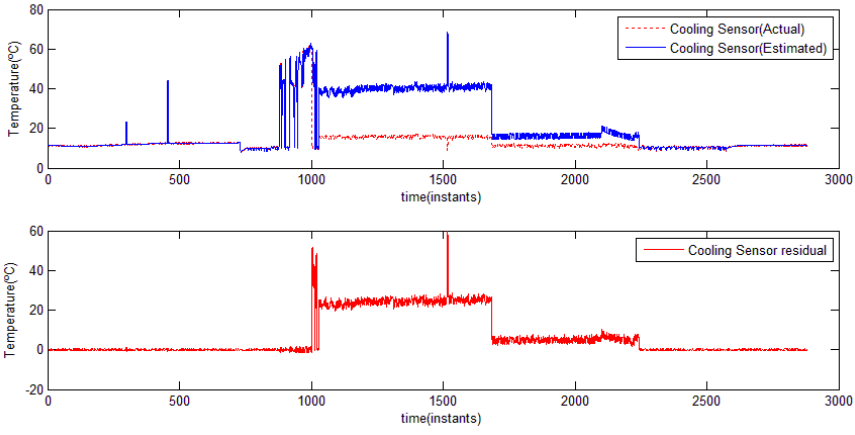


Figura 5.13 Escenario de Fallo CC Valve 20% (Cooling Sensor – PC2)

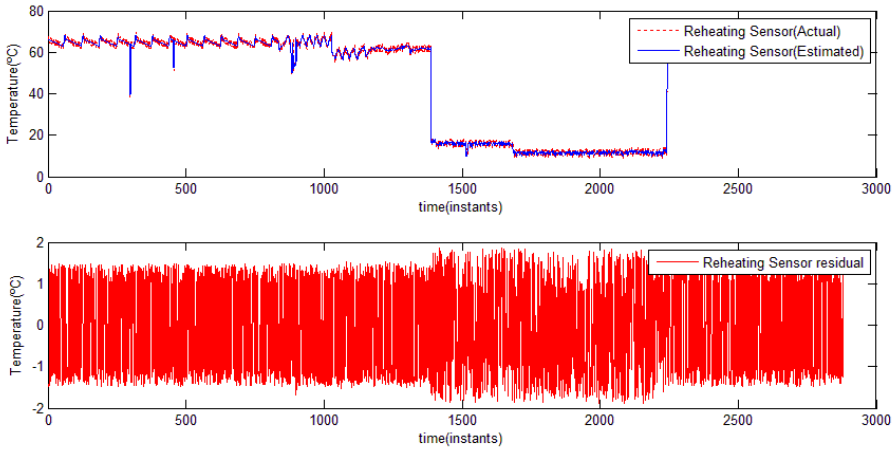


Figura 5.14 Escenario de Fallo CC Valve 20% (Reheating Sensor – PC3)

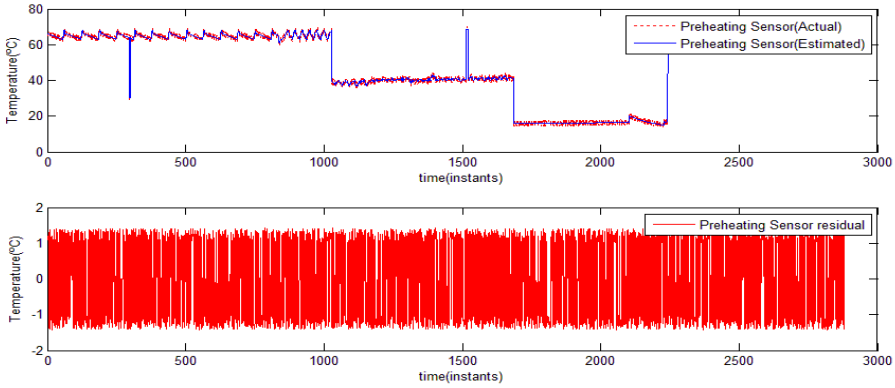


Figura 5.15 Escenario de Fallo CC Valve 50% (Preheating Sensor – PC1)

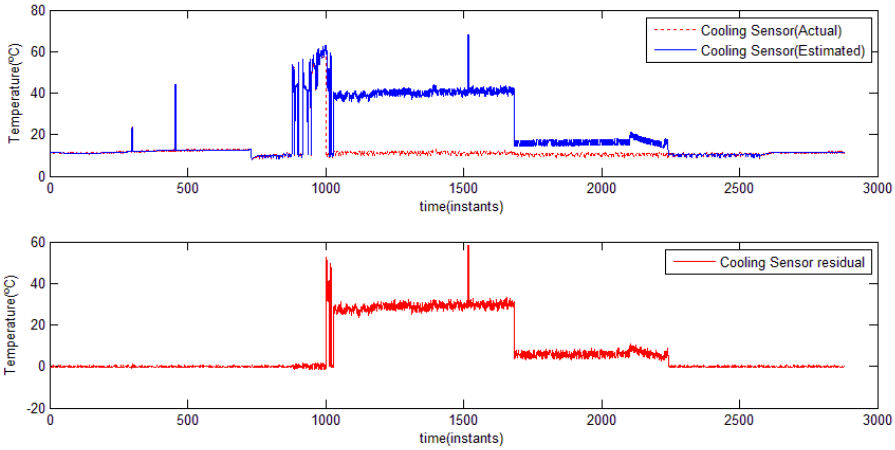


Figura 5.16 Escenario de Fallo CC Valve 50% (Cooling Sensor – PC2)

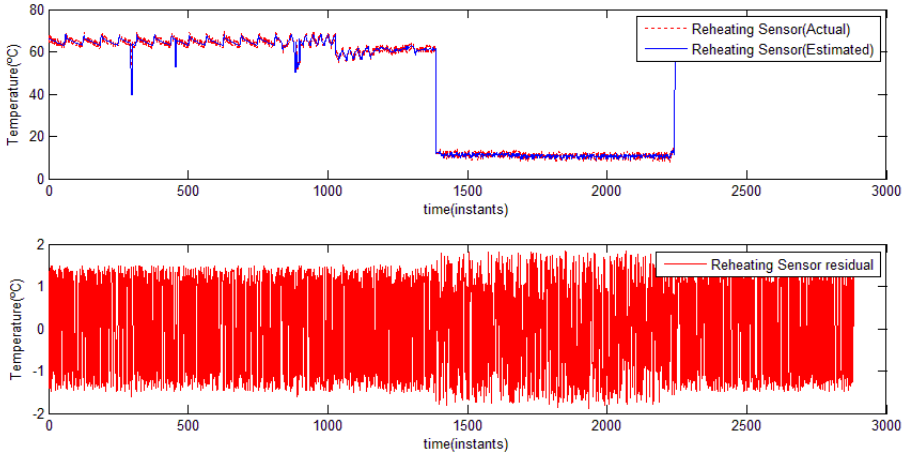


Figura 5.17 Escenario de Fallo CC Valve 50% (Reheating Sensor – PC3)

Como se puede ver, a partir del instante 1000 el valor de los residuos de PC2 comienza a crecer debido a la configuración de fallo. Cuanto mayor es la magnitud del fallo, el valor de los residuos aumenta. Al atascar las válvulas, el flujo de masa que circula por las mismas varía. Este flujo es indispensable para alterar la temperatura del flujo de aire en los bucles.

Al disminuir el flujo, la capacidad del mismo para calentar/enfriar dentro del bucle se ve reducida, lo que provoca que el cambio de temperatura en los sensores sea más "suave", como ocurre en las gráfica presente.

Empleando el mismo procedimiento para el resto de válvulas, la diagnosis ha dado como resultado la siguiente información:

Escenario	Instante de fallo	Magnitud de fallo	Posibles Conflictos Activados (instante)	Componentes Candidatos a Fallo
Válvula de Bucle de Precalentamiento atascada	1000	5%	PC1 (1031)	MB_Valve, **PC_Valve**, PC_UA, PC_Sensor
Válvula de Bucle de Precalentamiento atascada	1000	20%	PC1 (1032)	MB_Valve, **PC_Valve**, PC_UA, PC_Sensor
Válvula de Bucle de Precalentamiento atascada	1000	50%	PC1 (1031)	MB_Valve, **PC_Valve**, PC_UA, PC_Sensor

Válvula de Bucle de Enfriamiento atascada	1000	5%	PC2 (1001)	**CC_Valve**, CC_UA, CC_Sensor
Válvula de Bucle de Enfriamiento atascada	1000	20%	PC2 (1001)	**CC_Valve**, CC_UA, CC_Sensor
Válvula de Bucle de Enfriamiento atascada	1000	50%	PC2 (1001)	**CC_Valve**, CC_UA, CC_Sensor
Válvula de Bucle de Recalentamiento atascada	1000	5%	PC3 (1032)	**RC_Valve**, RC_UA, RC_Sensor
Válvula de Bucle de Recalentamiento atascada	1000	20%	PC3 (1391)	**RC_Valve**, RC_UA, RC_Sensor
Válvula de Bucle de Recalentamiento atascada	1000	50%	PC3 (1049)	**RC_Valve**, RC_UA, RC_Sensor

Como puede verse, la detección de fallos sufre de un pequeño retardo de tiempo en cada escenario. Si se observan las gráficas, puede verse que en parte esto se debe al tiempo que tarda el residuo en alcanzar un valor significativo (el cual es mayor en el caso de los escenarios asociados a las válvulas de precalentamiento y recalentamiento); mientras que por otra parte, el retardo está motivado por los intervalos de muestreo de señal considerados y su retraso en el z-test.

Los PCs activados en cada escenario son los adecuados, y entre los candidatos a fallo siempre está presente el que está sujeto a fallo. Para estos escenarios la labor de diagnóstico se ha efectuado correctamente.

- **Escenario de Fallo 2: Fallo en Sensores**

Los fallos en sensores suponen situaciones en las que los medidores de temperatura de cada bucle generan un valor que falsean la temperatura real de los mismos. Son casos aislados en los que el componente falla y se arroja un resultado que no se ajusta a la realidad.

El fallo en los sensores se modela sumando al valor generado por el sensor una cierta cantidad. En nuestro caso la magnitud a sumar se hace atendiendo a porcentaje del fallo aplicado al valor de la magnitud medida por el sensor en el instante de fallo.

A continuación vamos a exponer los resultados un escenario de fallo en el sensor del bucle de precalentamiento; con tres magnitudes de fallo diferente. Todas estas configuraciones tienen en común que el instante en el que se induce el fallo es el 500. Según la matriz de firmas teóricas, los PCs PC 1 y PC2 deberían activarse:

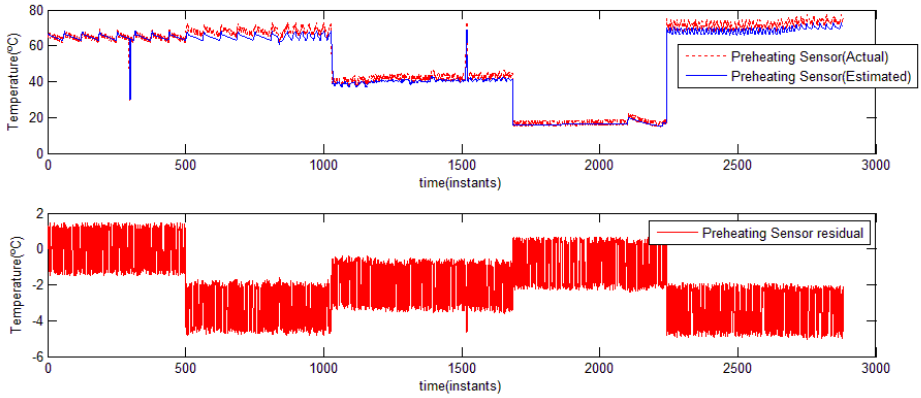


Figura 5.18 Escenario de Fallo PC Sensor 5% (Preheating Sensor – PC1)

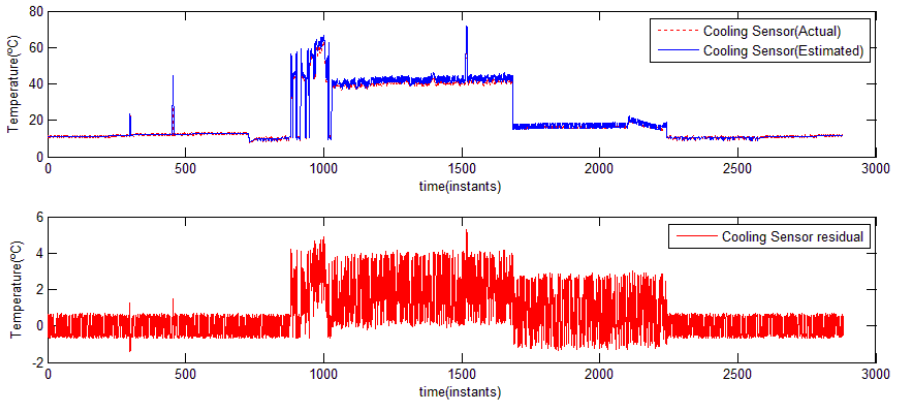


Figura 5.19 Escenario de Fallo PC Sensor 5% (Cooling Sensor – PC2)

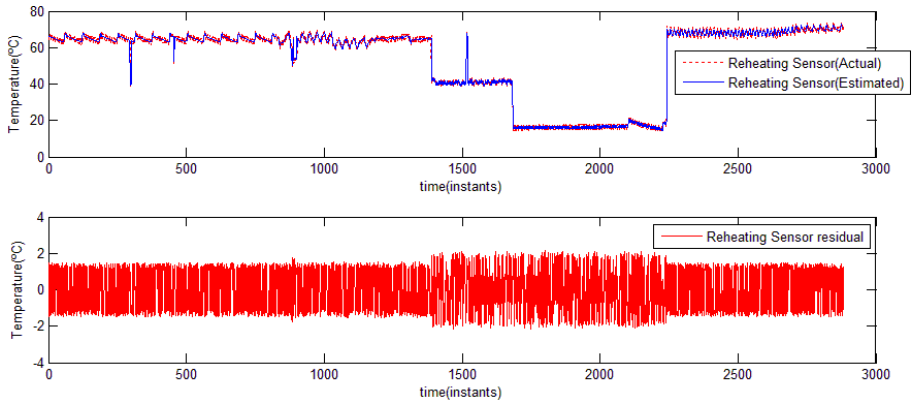


Figura 5.20 Escenario de Fallo PC Sensor 5% (Reheating Sensor – PC3)

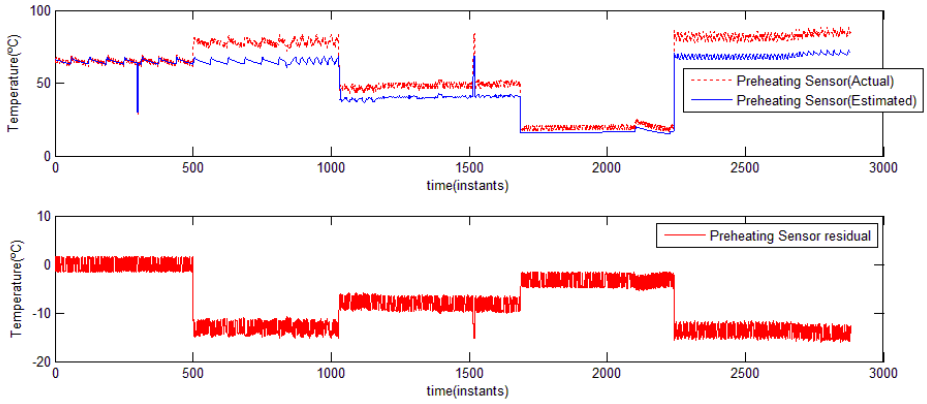


Figura 5.21 Escenario de Fallo PC Sensor 20% (Preheating Sensor – PC1)

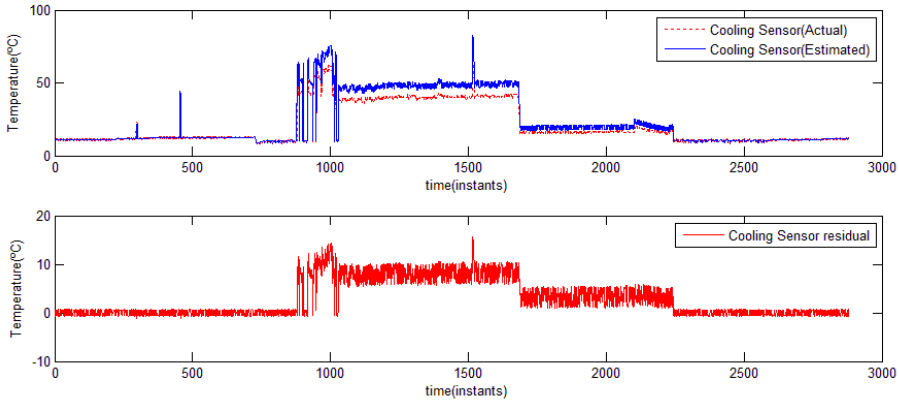


Figura 5.22 Escenario de Fallo PC Sensor 20% (Cooling Sensor – PC2)

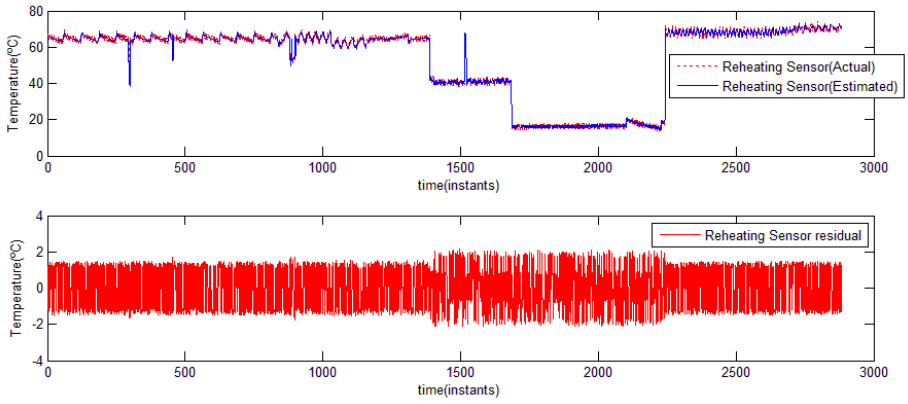


Figura 5.23 Escenario de Fallo PC Sensor 20% (Reheating Sensor – PC3)

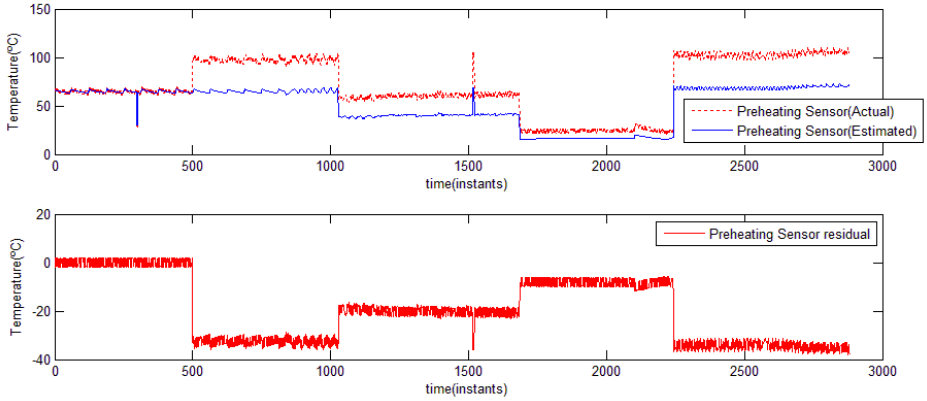


Figura 5.24 Escenario de Fallo PC Sensor 50% (Preheating Sensor – PC1)

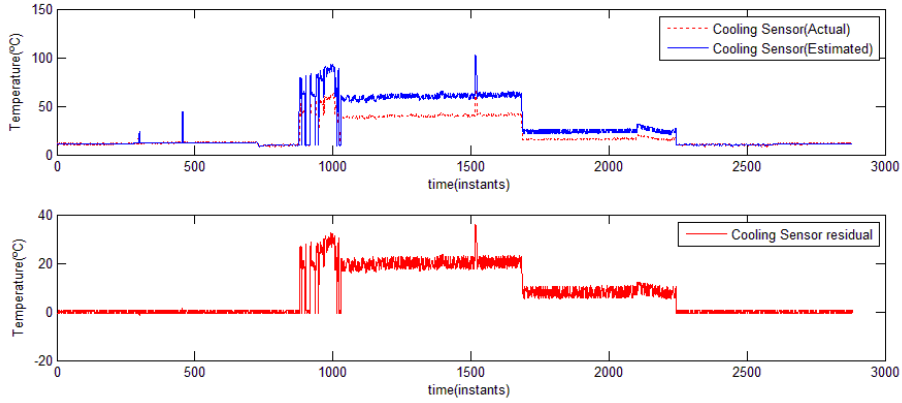


Figura 5.25 Escenario de Fallo PC Sensor 50% (Cooling Sensor – PC2)

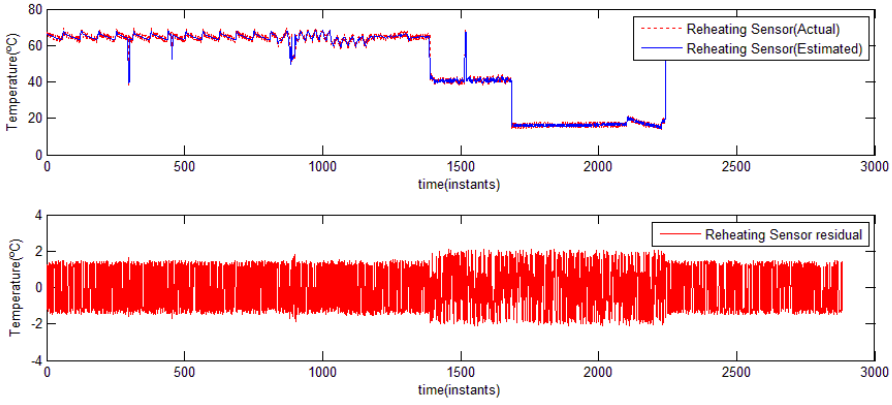


Figura 5.26 Escenario de Fallo PC Sensor 50% (Reheating Sensor – PC3)

En la figura anterior puede verse escenarios de simulación donde se ha inducido al sistema fallos en el sensor del bucle de precalentamiento. Esta configuración altera la observación de la temperatura en el componente, enmascarando su funcionamiento correcto.

La diagnosis ha dado como resultado la siguiente información:

Escenario	Instante de fallo	Magnitud de fallo	Posibles Conflictos Activados (instante)	Componentes Candidatos a Fallo
Sensor del Bucle de Precalentamiento averiado	500	5%	PC1 (513) PC2 (892)	**PC_Sensor**,[MB_Valve,CC_Valve], [MB_Valve,CC_AU], [MB_Valve,CC_Sensor], [PC_UA,CC_Valve], [PC_UA,CC_AU], [PC_UA,CC_Sensor], [PC_Valve,CC_Valve], [PC_Valve,CC_AU], [PC_Valve,CC_Sensor]
Sensor del Bucle de Precalentamiento averiado	500	20%	PC1 (504) PC2 (883)	**PC_Sensor**,[MB_Valve,CC_Valve], [MB_Valve,CC_AU], [MB_Valve,CC_Sensor], [PC_UA,CC_Valve], [PC_UA,CC_AU], [PC_UA,CC_Sensor], [PC_Valve,CC_Valve], [PC_Valve,CC_AU], [PC_Valve,CC_Sensor]
Sensor del Bucle de Precalentamiento averiado	500	50%	PC1 (502) PC2 (881)	**PC_Sensor**,[MB_Valve,CC_Valve], [MB_Valve,CC_AU], [MB_Valve,CC_Sensor], [PC_UA,CC_Valve], [PC_UA,CC_AU], [PC_UA,CC_Sensor], [PC_Valve,CC_Valve], [PC_Valve,CC_AU], [PC_Valve,CC_Sensor]
Sensor del Bucle de Enfriamiento averiado	500	5%	PC3 (1420)	RC_Valve, RC_UA, **RC_Sensor**
Sensor del Bucle de Enfriamiento averiado	500	20%	PC2 (507) PC3 (889)	**CC_Sensor**,[CC_Valve,RC_Valve], [CC_Valve,RC_AU], [CC_Valve,RC_Sensor], [CC_UA,RC_Valve], [CC_UA,RC_AU], [CC_UA,RC_Sensor], [CC_Valve,RC_Valve], [CC_Valve,RC_AU],

				[CC_Valve,RC_Sensor]
Sensor del Bucle de Enfriamiento averiado	500	50%	PC2 (505) PC3 (884)	**CC_Sensor**,[CC_Valve,RC_Valve], [CC_Valve,RC_AU], [CC_Valve,RC_Sensor], [CC_UA,RC_Valve], [CC_UA,RC_AU], [CC_UA,RC_Sensor], [CC_Valve,RC_Valve], [CC_Valve,RC_AU], [CC_Valve,RC_Sensor]
Sensor del Bucle de Recalentamiento averiado	500	5%	PC3 (516)	RC_Valve, RC_UA, **RC_Sensor**
Sensor del Bucle de Recalentamiento averiado	500	20%	PC3 (504)	RC_Valve, RC_UA, **RC_Sensor**
Sensor del Bucle de Recalentamiento averiado	500	50%	PC3 (501)	RC_Valve, RC_UA, **RC_Sensor**

Como puede verse, para los escenarios considerados, la respuesta de la diagnosis es la correcta, activando los PCs adecuados en un tiempo razonable de cara al retardo entre la aparición del fallo y la activación del PC.

Los fallos en sensores en nuestro sistema también pueden generar situaciones en las que más de un PC se active. Las mediciones de los sensores son estimaciones de los PCs, pero también variables de entrada para los mismos. Si estos valores se ven alterados, es posible que el fallo en cuestión active tanto el PC que estima el valor de la variable observada, como el PC que emplea la misma de entrada

Tomemos como referencia la figura 5.23, asociada a un fallo del 50% en el sensor de precalentamiento. Como puede verse, una vez se introduce el fallo la discrepancia entre estimación y observación se vuelve visible.

La medición errónea del sensor sirve como entrada para el PC asociado al bucle de enfriamiento. Esto provoca que a la larga, la introducción de dichos valores alterados modifique el valor de las estimaciones del PC, lo que genera una nueva discrepancia que activa al mismo.

Esta situación ya se había contemplado previamente en el estudio del sistema, y es la que nos motiva a considerar en el conjunto de candidatos asociados a cada bucle, el componente sensor de temperatura del bucle anterior al que están conectados dentro de la configuración del sistema.

- **Escenario de Fallo 3: PC UA**

Los fallos en el parámetro UA ocurren cuando los bucles sufren averías internas, que derivan en una reducción de su capacidad para transferir calor (dicha propiedad solo se ve afectada negativamente en caso de avería, reduciendo su magnitud). Se trata de fallos permanentes.

Los parámetros UA asociados a cada bucle son todos iguales, derivado de que el modelado del sistema considera a dichos componentes idénticos. Inducir fallo en este parámetro supone reducir dicha magnitud en un tanto por ciento.

A continuación vamos a exponer los resultados un escenario de fallo en el sensor del bucle de recalentamiento; con tres magnitudes de fallo diferente. Todas estas configuraciones tienen en común que el instante en el que se induce el fallo es el 1000:

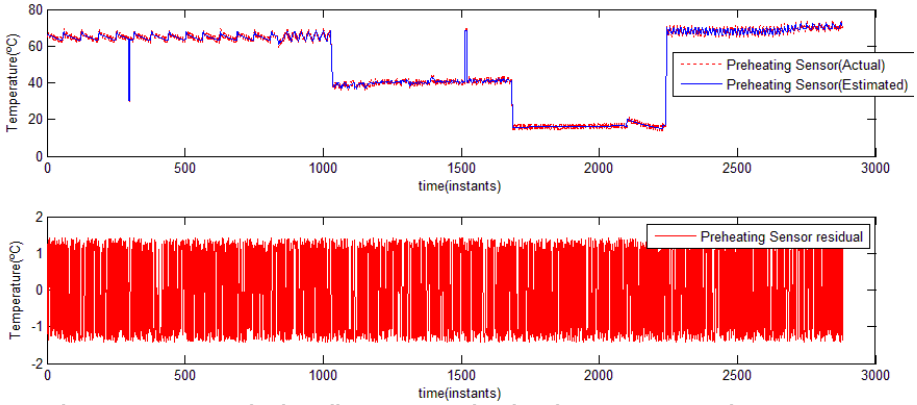


Figura 5.27 Escenario de Fallo PC UA 5% (Preheating Sensor – PC1)

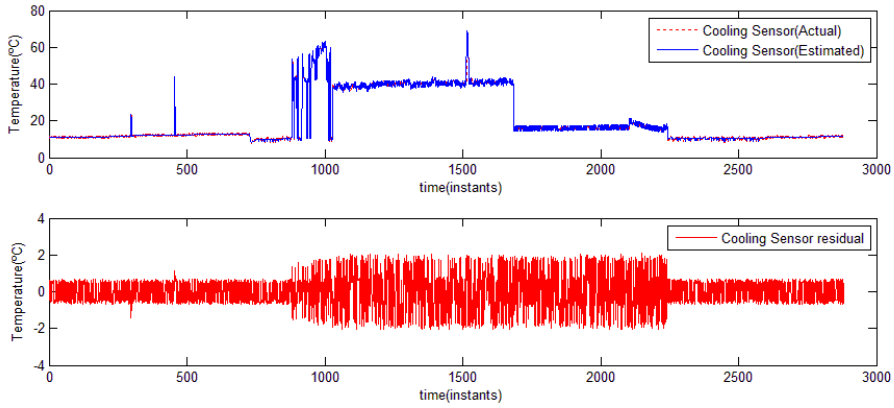


Figura 5.28 Escenario de Fallo PC UA 5% (Cooling Sensor – PC2)

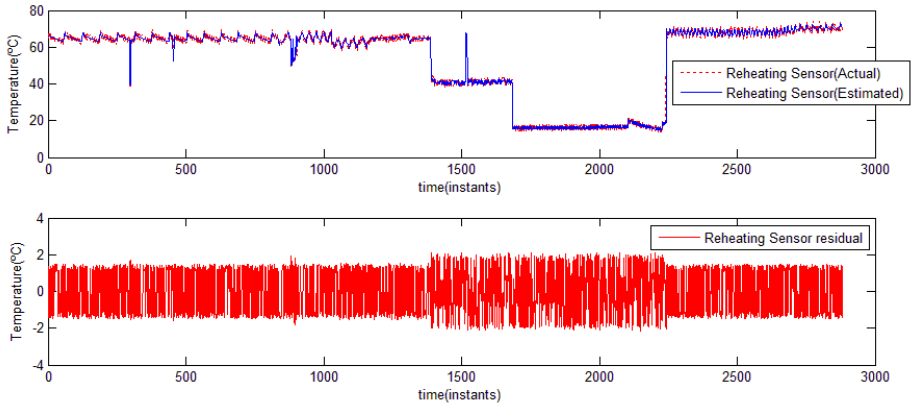


Figura 5.29 Escenario de Fallo PC UA 5% (Reheating Sensor – PC3)

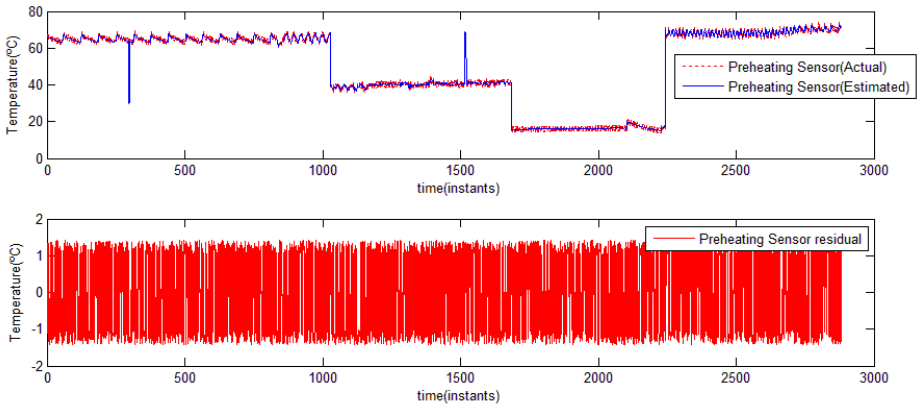


Figura 5.30 Escenario de Fallo PC UA 20% (Preheating Sensor – PC1)

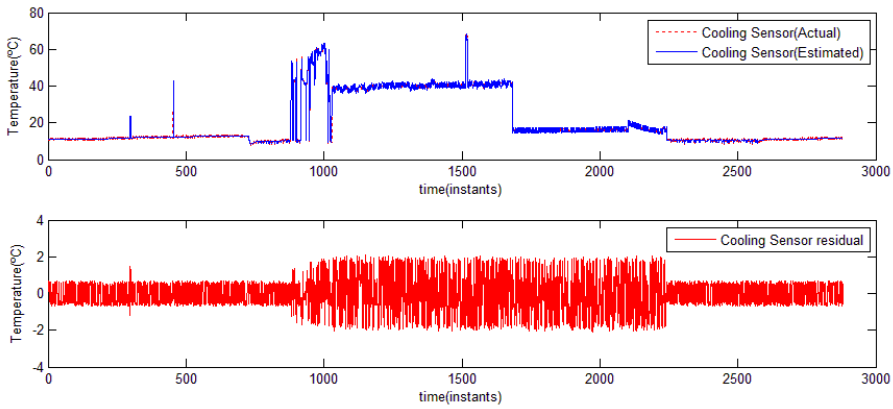


Figura 5.31 Escenario de Fallo PC UA 20% (Cooling Sensor – PC2)

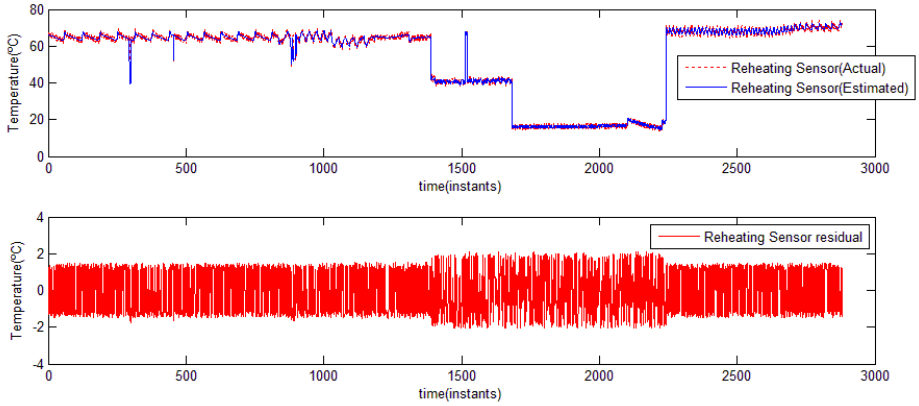


Figura 5.32 Escenario de Fallo PC UA 20% (Reheating Sensor – PC3)

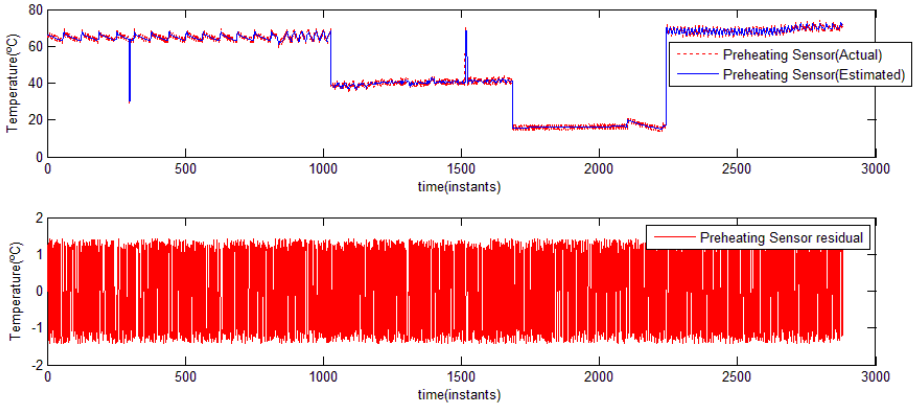


Figura 5.33 Escenario de Fallo PC UA 50% (Preheating Sensor – PC1)

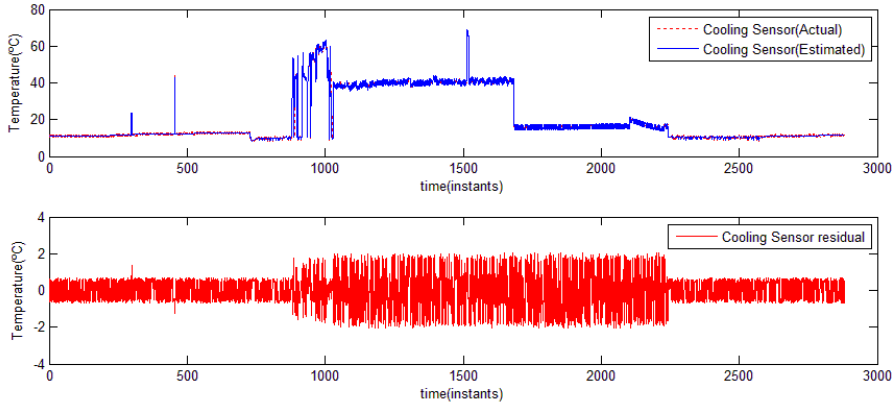


Figura 5.34 Escenario de Fallo PC UA 50% (Cooling Sensor – PC2)

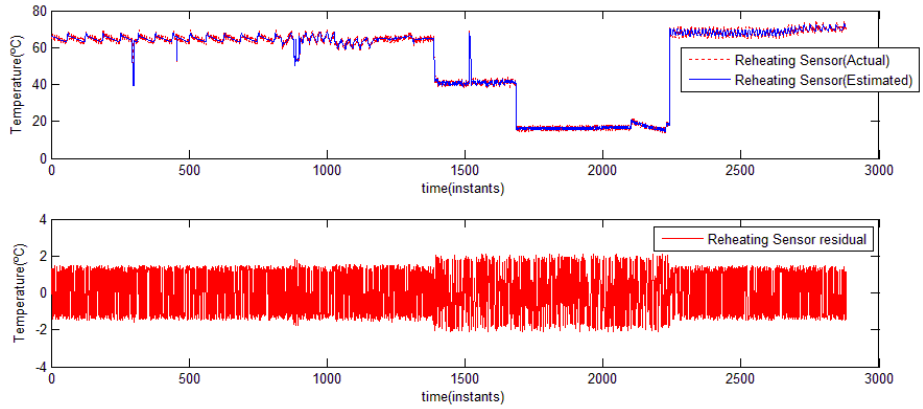


Figura 5.35 Escenario de Fallo PC UA 50% (Reheating Sensor – PC3)

El fallo en el parámetro UA de los sistemas ha resultado ser imperceptible. Para fallos con una magnitud del 5%, 20% y 50% los residuos generados resultan estar muy próximos a los del modo nominal, lo que hace que el sistema no los considere como fallos.

Es posible que se den situaciones de fallo don las consecuencias del mismo sean imperceptibles a ojos de la técnica de diagnóstico. En el caso de los fallos en el parámetro UA estamos frente a dichos casos particulares.

- **Escenario de Fallo 4: Fallo en Válvula Neumático del Compartimento de Mezclado MB Valve**

Los fallos en el compartimento de mezclado ocurren cuando la posición de la válvula que conduce el flujo de aire reciclado y exterior a los bucles se atasca. Se trata de fallos permanentes.

El nivel de posicionamiento de la válvula se modela entorno a un cierto porcentaje.

A continuación vamos a exponer los resultados de tres escenarios de fallo; cada uno de ellos con una magnitud de apertura de la válvula diferente. Todas estas configuraciones tienen en común que el instante en el que se induce el fallo es el 1000. Según la matriz de firmas teóricas solo debería activarse PC1:

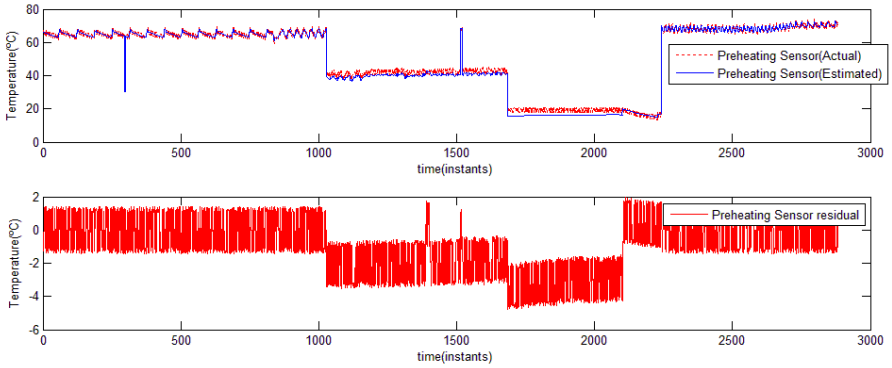


Figura 5.36 Escenario de Fallo PC MB 5% (Preheating Sensor – PC1)

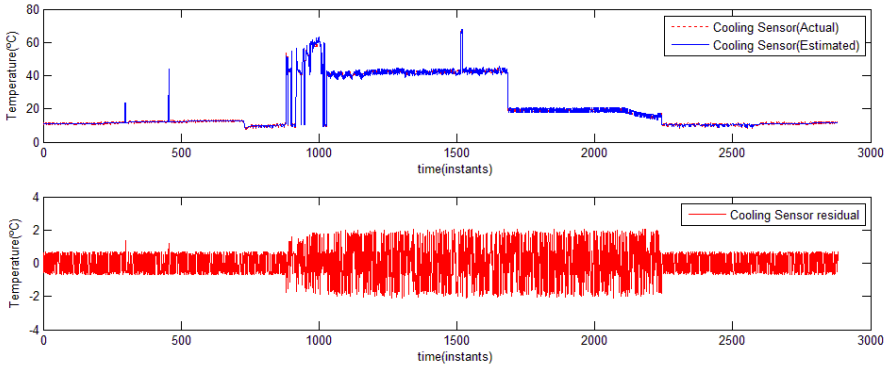


Figura 5.37 Escenario de Fallo PC MB 5% (Cooling Sensor – PC2)

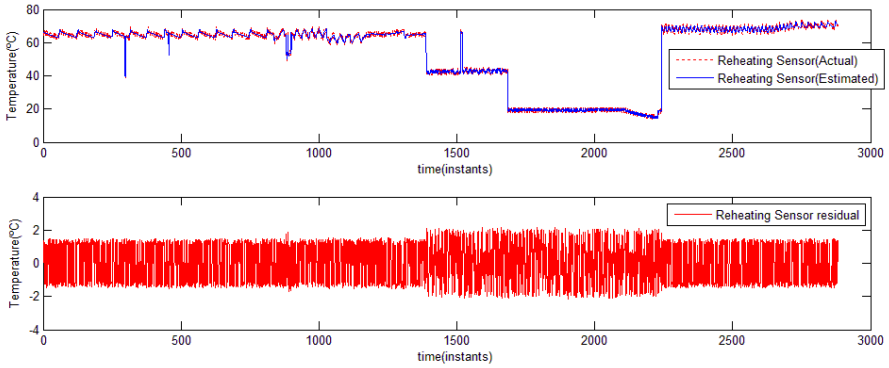


Figura 5.38 Escenario de Fallo PC MB 5% (Reheating Sensor – PC3)

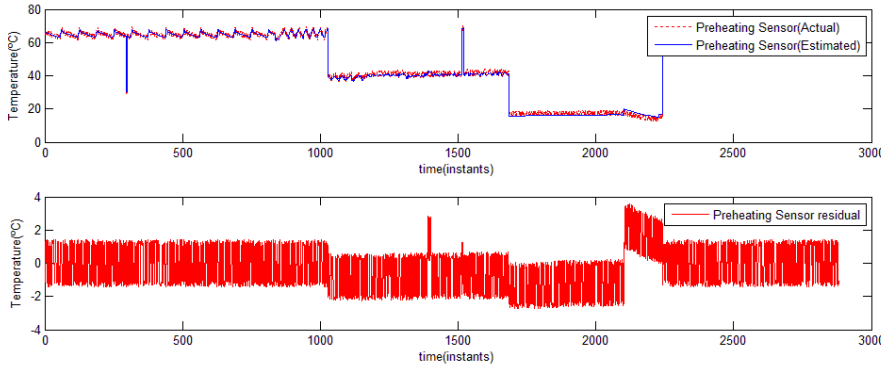


Figura 5.39 Escenario de Fallo PC MB 20% (Preheating Sensor – PC1)

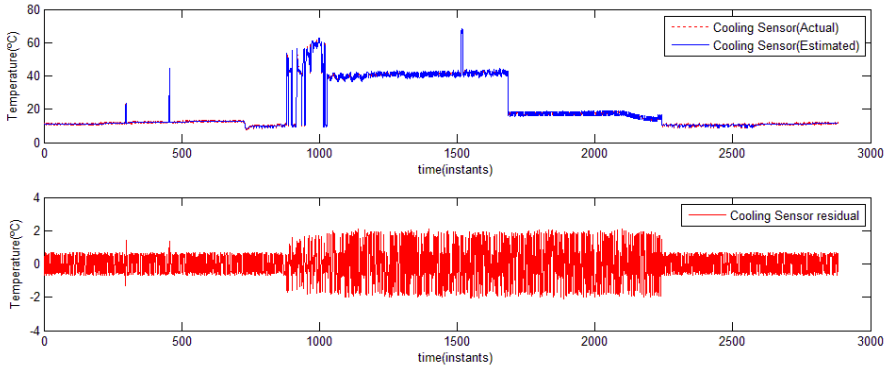


Figura 5.40 Escenario de Fallo PC MB 20% (Cooling Sensor – PC2)

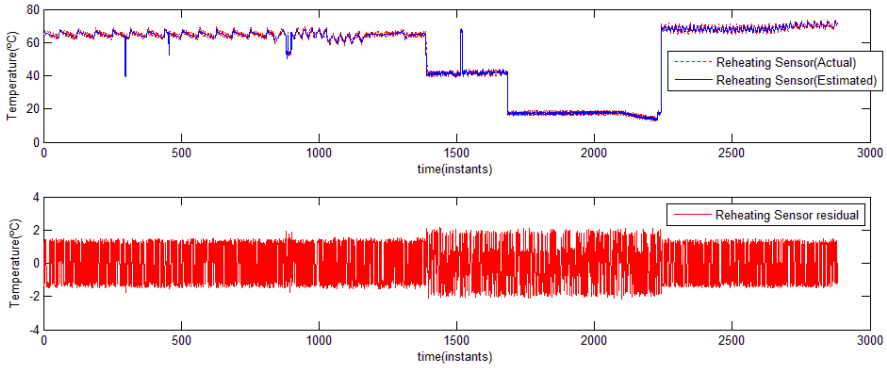


Figura 5.41 Escenario de Fallo PC MB 20% (Reheating Sensor – PC3)

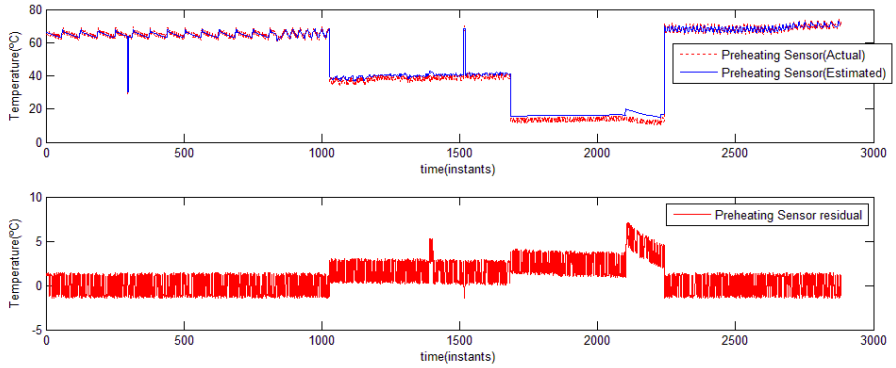


Figura 5.42 Escenario de Fallo PC MB 50% (Preheating Sensor – PC1)

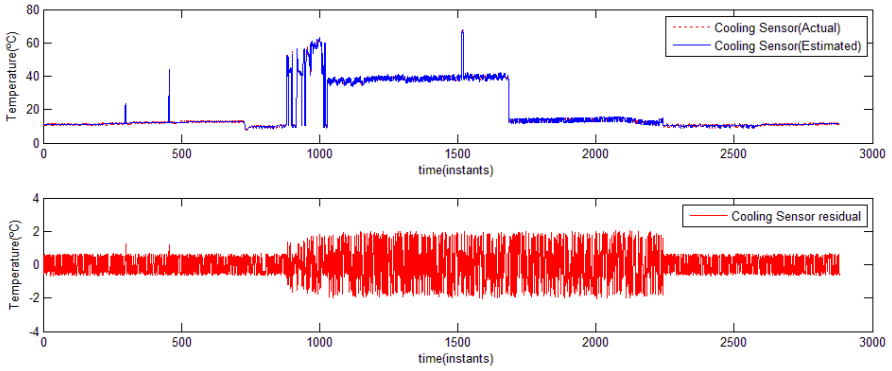


Figura 5.43 Escenario de Fallo PC MB 50% (Cooling Sensor – PC2)

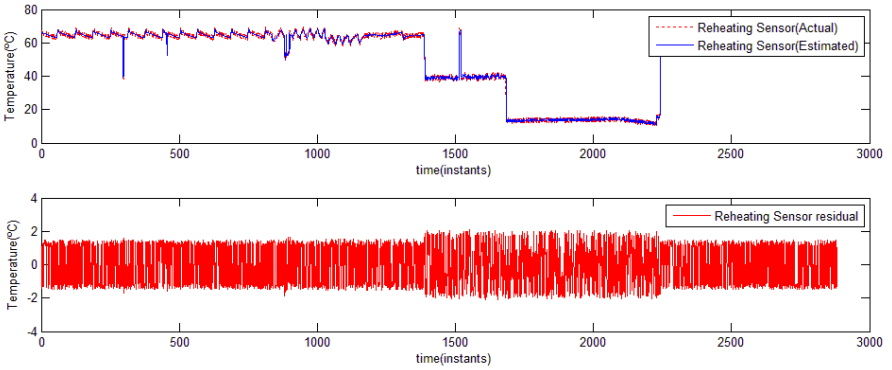


Figura 5.44 Escenario de Fallo PC MB 50% (Reheating Sensor – PC3)

Los fallos en la posición de los amortiguadores del compartimento suponen una alteración de la temperatura del flujo de aire entrante al sistema. Al alterar la proporción de flujos de aire reciclado y externo, la temperatura resultante del sistema está condiciona por la de los dos flujos entrantes.

En cada escenario, la magnitud del posicionamiento del amortiguador no presenta ninguna relación de proporcionalidad con la temperatura del flujo de aire.

La diagnosis ha dado como resultado la siguiente información:

Escenario	Instante de fallo	Magnitud de fallo	Posibles Conflictos Activados (instante)	Componentes Candidatos a Fallo
Amortiguador del Compartimento de Mezclado averiado (5%)	1000	5%	PC1 (2110)	**MB_Valve**, PC_Valve, PC_UA, PC_Sensor
Amortiguador del Compartimento de Mezclado averiado (20%)	1000	20%	PC1 (2120)	**MB_Valve**, PC_Valve, PC_UA, PC_Sensor
Amortiguador del Compartimento de Mezclado averiado (50%)	1000	50%	PC1 (2122)	**MB_Valve**, PC_Valve, PC_UA, PC_Sensor

5.5. Análisis de Resultados y Conclusiones

En este capítulo hemos documentado nuestra labor de diagnóstico. Aplicando los principios teóricos recogidos a lo largo de toda la memoria hemos sido capaces de efectuar un diagnóstico correcto de un sistema real en un amplio conjunto de hipotéticos escenarios de funcionamiento.

Para efectuar dicha labor de diagnóstico hemos implementado un sencillo sistema de simulación en Matlab. Trabajando con un modelo del sistema AHU-9 simplificado y diseñado por nosotros, nuestro software es capaz de emular el comportamiento del sistema AHU-9 definido por su diseño en Modelica, y de manipular los PCs considerados para la estimación de residuos y la activación de los mismos.

Se han considerado una serie de posibles escenarios de fallos para nuestros experimentos. Por cada componente susceptible de fallo, hemos realizado tres experimentos con magnitudes de fallo del 5%, 10% y 50%.

Cada experimento ha generado un comportamiento particular del sistema que ha sido explicado y razonado:

- Los fallos en válvulas hidráulicas suponen alterar el flujo de masa circulante por el circuito, necesario para que los bucles de aclimatamiento se ajusten a la temperatura deseada. Cuando se reduce el flujo, se reduce la capacidad de aclimatamiento del bucle,

al recibir una menor circulación de líquido con la que hacer más efectiva la transferencia de calor entre éste y el componente.

- Los fallos en sensores enmascaran el comportamiento correcto del sistema, ofreciendo datos de temperatura erróneos que no se corresponden con los reales. Para la configuración de nuestro circuito, este error activa varios PCs, ya que los valores de los sensores condicionan tanto discrepancias en las observaciones como en las entradas de los PCs.

- Los fallos en los parámetros UA de los bucles han resultado ser los menos significativos. Un fallo en el parámetro UA de un bucle supone de tratar con una avería del componente que reduce su capacidad para la transferencia de calor. Por defecto se trabaja con un UA de 320*20. En los escenarios de fallo considerados, hasta que no se ha trabajado con una magnitud de fallo considerable (50%), no se han detectado fallos.

  Esta situación puede deberse al hecho particular de que el UA con la que estemos trabajando sea relativamente pequeña para la propia expresión de efectividad con la que trabajamos, lo que provoca que la modificación del parámetro en cuestión no sea significativa para nuestro método de diagnóstico. Así, nuestros PCs, con los parámetros de *ztest* considerados, no son capaces de tratar con errores de AU.

- Los fallos en la posición del amortiguador del Compartimento de Mezclado alteran la temperatura del flujo en circulación por el sistema. Esta alteración del flujo está condicionada por la temperatura de los flujos entrantes, y la proporción de cada uno que emplea nuestro sistema. El uso en mayor o menor medida de uno de los flujos determinará que la temperatura del flujo resultante se aproxime a la temperatura de dicho flujo, distanciada en base a la temperatura del flujo restante, que bien puede aumentar o reducir dicha temperatura. Este es el único de los fallos cuyo resultado nos resulta impredecible, al estar condicionado por el valor de las entradas exclusivamente, y ser independiente del comportamiento interno del sistema.

La conclusión de los resultados es que se han podido diagnosticar todos los escenarios de fallos posibles, salvo los correspondientes al parámetro UA. Se han planteado unas hipótesis de comportamiento de sistema por cada tipo de fallo que se han cumplido en el caso práctico.

A su vez se ha hecho frente a un escenario de fallo particular el cual presentaba unos síntomas imperceptibles para el sistema de diagnóstico, lo que ha presentado un caso de fallo indetectable.

También se ha comprobado que para otros escenarios, como el del compartimento de mezcla; existen fallos en los sistemas cuya detección puede alargarse considerablemente en el tiempo, al estar influenciados por factores externos que determinan, una vez el fallo está presente, cuando es perceptible o no.

Con todo esto damos por finalizada nuestra labor de investigación. Para cerrar la memoria ofrecemos un capítulo final de conclusiones, donde se hace una breve revisión de

toda la labor desempeñada a lo largo del desarrollo del proyecto, con formulación de conclusiones y vías de trabajo futuras.

Capítulo 6:

Conclusiones

6.1. Introducción

En este capítulo presentamos las conclusiones a las que hemos llegado una vez finalizado el trabajo; y expondremos diversos comentarios acerca del mismo.

Adicionalmente haremos mención a posibles líneas de trabajo futuras sobre la herramienta de diagnóstico generada, donde detallaremos posibles mejoras las cuales podrían llevarse a cabo en posteriores revisiones.

6.2. Conclusiones

La principal conclusión a la que podemos llegar después de realizar la labor de diagnóstico del sistema AHU-9 es que se ha conseguido dar respuesta a los objetivos clave planteados inicialmente:

- **Se ha efectuado una labor de diagnóstico del sistema Air Handling Unit-Nine aplicando la técnica de Diagnóstico Basado en Consistencia con Posibles Conflictos. A esta conclusión se llega tras haber tenido resultado las siguientes tareas:**

  - Se ha efectuado un estudio completo del sistema, comprendiendo su funcionamiento así como los posibles componentes del mismo susceptibles de sufrir fallos.

  - Se ha conseguido reescribir satisfactoriamente un modelo simplificado del sistema capaz de reproducir su comportamiento real. A partir del mismo ha sido posible su factorización para la generación de los Posibles Conflictos empleados posteriormente para la diagnosis.

  - Se han generado una serie de posibles escenarios de funcionamiento del sistema para los cuales se ha realizado la diagnosis. Hemos sido capaces de determinar en cada situación el fallo y un correcto conjunto de componentes candidatos a fallo. Esta labor se ha conseguido automatizar con éxito, de manera que se han abarcado todos los casos posibles considerados como significativos para la labor de diagnóstico.

  - Adicionalmente se ha implementado un software auxiliar de apoyo al investigador, donde la generación de escenarios de funcionamiento y el cálculo de residuos, junto con la aplicación de la técnica de diagnóstico se efectúan automáticamente.

El proceso de diagnóstico ha supuesto una labor desempeñada gradualmente, condicionada por los avances en la organización del concurso.

La organización ha puesto a nuestra disposición los modelos del sistema y ficheros de entrada para la simulación de los mismos. A partir de esta información hemos sido capaces de desarrollar nuestra labor.

Cabe destacar la tarea subyacente de colaboración que se ha realizado con el equipo de ingenieros encargados de la elaboración del modelo. Nuestros estudios del sistema y su

modelado software han supuesto la elaboración de informes donde se detallaron anomalías del modelo y discrepancias estructurales con el sistema real.

Estos puntos han sido expuestos en el capítulo del estudio del sistema que acontece en la presente memoria. Esta colaboración ha sido la que ha motivado el desarrollo de un modelo simplificado del sistema, más correcto que el inicial, y acorde a la especificación del AHU-9 original.

Además del estudio del sistema, parte de la carga de trabajo del proyecto se ha destinado al estudio de las técnicas de diagnosis y la aplicación de las mismas sobre el sistema en cuestión. Comprender el mecanismo de la DBC usando PCs requiere dominar toda una serie de conceptos asociados al campo de la Inteligencia Artificial. El estudio supone entender el método de razonamiento de Sistemas Expertos en su variante orientada a modelos.

A su vez, efectuar una labor eficiente y eficaz de diagnóstico requiere de trabajar con herramientas teóricas que exigen de su comprensión.

Trabajar con un Filtro de Partículas para la estimación del estado interno del sistema requiere comprender el fundamento de esta herramienta, basada en el Método de Monte Carlo, el estudio de sus diferentes implementaciones y la determinación del filtro más adecuado a la situación a la que se hace frente.

Algo similar ocurre con el uso del *z-test* frente al uso de parámetros umbral en la detección de fallos y la activación de PCs. Aplicar el test estadístico supone entender el fundamento teórico del mismo, su utilidad; y saber ajustar los parámetros de números de muestras y periodos de muestreo para obtener una detección eficiente y precisa de fallos.

Al final, toda la labor documental se ha aplicado a un caso práctico y real. Hemos sido capaces de trasladar un modelo de sistema complejo a una notación sencilla y manipulable por los investigadores. Hemos eliminado redundancias en el código y depurado un sistema industrial hasta el punto de definirlo en base a nuestros intereses.

Además de ello, se ha conseguido replicar el comportamiento del mismo en un lenguaje especializado para la simulación; y sobre el cual se ha podido aplicar la técnica de diagnóstico.

Adicionalmente se ha hecho una breve labor de ingeniería software al desarrollar una herramienta con la que se ha automatizado toda la labor de diagnóstico. Se ha implementado una pequeña aplicación con la que el investigador solo tiene que definir los escenarios de funcionamiento e introducir las entradas necesarias para diagnosticar el sistema.

Se trata de una labor opcional, y llevada a cabo por iniciativa propia del investigador. Esta propuesta favorecerá a los lectores del documento la interacción con el sistema; además de facilitar el estudio del AHU-9 a posteriores investigadores que revisen o vuelvan a abordar este problema particular.

6.3. Líneas de trabajo futuras

Nuestra labor de diagnóstico se asocia a un caso muy particular de problema, como es el de diagnosticar un sistema AHU-9 como el que está instalado en la Universidad de Cork.

Sin embargo sí que se pueden considera ciertas líneas de investigación a partir de nuestra labor:

- ✓ **Integración del estudio en el DXC-13':** el último paso de la labor sería el de ingresar nuestro proyecto dentro del concurso de diagnóstico DXC-13' para poner a prueba nuestra técnica de diagnóstico.

  Ahora mismo el sistema efectúa una diagnosis offline por medio de la plataforma Matlab. La participación en el concurso supone adaptar el modelo para que sea capaz de trabajar con datos "online" procedentes de un servidor externo, y trasladar el sistema a una plataforma compatible con las especificaciones del concurso.

  Esta labor depende íntegramente de la organización del concurso, que debe ser responsable de oficializar la participación en esta prueba y de proporcionar los medios necesarios para integrar la herramienta en su ámbito de competición.

- ✓ **Diseño de un software conversor de modelos:** para un entorno de trabajo industrial Modelica representa un lenguaje de programación ideal.

  Mediante Modelica, cualquier usuario sin conocimientos de ingeniería software es capaz de crear programas robustos y correctos, donde la funcionalidad del software es tan buena como el fundamento teórico sobre el que se asienta su modelado.

  Si en el entorno de trabajo de un investigador dedicado al diagnóstico de sistemas, está continuamente trabajando con modelos Modelica de sus sistemas; el estudio de los mismos puede suponer un esfuerzo considerable (y más si trata con sistemas tan complejos como bien pudieran ser motores o instalaciones eléctricas).

  Lo ideal sería crear un software capaz de tratar con modelos Modelica, reduciendo la complejidad de los mismos hasta el punto de tratar exclusivamente con las ecuaciones que modelan el comportamiento de los componentes básicos.

  Se trataría de un software de manipulación de modelos, a modo de conversor. Su salida además, podría conectarse a otras herramientas como la de diagnóstico automático ya presente en el grupo de investigación (PCsDX). Con ello se conseguiría automatizar la diagnosis de sistemas complejos desde su modelado nativo.

La labor de implementación requeriría de disponer de un ingeniero software familiarizado con nuestra técnica de diagnóstico; además de dominar perfectamente el lenguaje Modelica y la plataforma Matlab, a la que se efectuaría la conversión del modelo (elegimos Matlab por trabajar con una notación sencilla, y por ofrecer funcionalidad para convertir su código a otros lenguajes de programación populares, como C).

Capítulo 7:

Referencias

Bibliografía:

- ✓ **[Alonso2012] Alonso, C., Pulido, B., Bregón, A., Moya, N y Rubio, D. "Automatic generation of Dynamic Bayesian Networks for hybrid systems fault diagnosis".** 22nd International Workshop on Principles of Diagnosis (DX-2012). Año 2012.

- ✓ **[Arulampalam2002]Arulampalam, M.S., Maskell, S. y Gordon, N."A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayseian Tracking".***IEEE TRANSACTIONS ON SIGNAL PROCESSING, Vol. 50:174-188, Nº 2, Año 2002.*

- ✓ **[Balakrishnan98] Balakrishnan, K., Honavar, V. "Intelligent diagnosis systems".** *Journal of Intelligent Systems, Vol. 8:239-290*. Año 1998.

- ✓ **[Biswas2011] Biswas, G., Alonso, C., Pulido, B., Bregón, A., Moya, N y Rubio, D. "Automatic generation of Dynamic Bayesian Networks for hybrid systems fault diagnosis".** 22nd International Workshop on Principles of Diagnosis (DX-2011). Año 2011.

- ✓ **[Bregón2007] Bregón, A. "Trabajo de Investigación – Doctorado en Informática".** Trabajo de Investigación Tutelado. *Escuela Técnica Superior de Ingeniería Informática. Universidad de Valladolid, Año 2010.*

- ✓ **[Calleja&Rubio2010] Calleja, E. y Rubio, D. "Herramienta para el Cálculo de Posibles Conflictos con Ciclos".** Proyecto Fin de Carrera. *Escuela Técnica Superior de IngenieríaInformática.Universidad de Valladolid, Año 2010.*

- ✓ **[Console2000] L.Console. "Model-based diagnosis history and state of the art"**. MONET Summer School en Bertinoro, Italia. http://monet.aber.ac.uk/. Mayo 2000.

- ✓ **[Console&Kleer92] W. Hamscher, L.Console, J. de Kleer, Eds., "Reading in Model Based Diagnosis"**. San Mateo, CA: Morgan-Kaufmann. Año 1992.

- ✓ **[DX2013] DX-2013 "The 24th International Workshop on Principles of Diagnosis: DX-2013".**Espacio web de la organización del congreso de principios de diagnosis DX13'; donde se recoge la documentación del evento. http://www.dx-2013.org/ (Última Visita: 2/9/2012)

- ✓ **[DXC2013] DXC-2013 "The 4th International Diagnostic Competition: DXC-2013".**Espacio web de la organización de conjunto de diagnosis DXC13'; donde se recoge la documentación de las normas del concurso, plazos y documentación sobre los sistemas a diagnosticar. http://www.dxc-2013.org/ (Última Visita: 2/9/2012)

- ✓ **[Koller&Lerner2001]Koller, D. y Lerner, U."Sampling in Factored Dynamic Systems".** Capítulo en "Sequential Monte Carlo Methods in Practice". Editorial Springer, Año 2011.

- ✓ **[Kumar2008] Kumar, A., Ould, B. "Model-based Process Supervision: A Bond Graph Approach (Advances in Industrial Control)".** ISBN 1848001584. Editorial Springer. *Año 2008.*

- ✓ **[Kuipiers94] Kuipers, B. "Qualitative Reasoning. Modeling and Simulation with Incomplete Knowledge".** MIT Press, Cambridge. *Año 1994.*

- ✓ **[Lajo2007] Lajo, R. "Herramienta de Cálculo del Conjunto de Posibles Conflictos".** Proyecto Fin de Carrera.*Escuela Técnica Superior de Ingeniería Informática.Universidad de Valladolid, Año 2007.*

- ✓ **[MatWorks2012] MatWorks "Matlab: Product Documentation".**Espacio web de Matlab donde se recoge la documentación de todas las bibliotecas de la plataforma Matlab.http://www.mathworks.es/help/techdoc/ (Última Visita: 2/9/2012)

- ✓ **[MatWorks2012] MatWorks "Matlab: CreatingGraphica lUser Interfaces R2012a"**http://www.mathworks.com/help/pdf_doc/matlab/buildgui.pdf (Última Visita: 2/9/2012)

- ✓ **[MatWorks2012] MatWorks "Matlab: Answers".**Espacio web de Matlab destinado a la ayuda mutua entre usuario de la plataforma Matlab por medio de foros de consulta.http://www.mathworks.es/matlabcentral/answers/ (Última Visita: 2/9/2012)

- ✓ **[Modelica2000] Modelica "ModelicaTM – A Unified Object-Oriented Language for Physical Systems Modeling, Versión 1.4".** Espacio web Modelica destinado a la formación básica de los usuarios en el lenguaje de programación Modelica. https://modelica.org/documents/ModelicaTutorial14.pdf (Última Visita: 2/9/2012)

- ✓ **[Moya2010] Moya, N. "Factorización de Redes Bayesianas Dinámicas mediante Posibles Conflictos y propuesta de aplicación a la Diagnosis Basada en Consistencia".** Trabajo de Investigación Tutelado.*Escuela Técnica Superior de Ingeniería Informática.Universidad de Valladolid, Año 2010.*

- ✓ **[OpenModelica2013] OpenModelica "User Documentation".** Espacio web OpenModelica destinado a la formación básica de los usuarios en el manejo de la herramienta OpenModelica y el desarrollo de modelos de sistemas y bibliotecas de componentes. https://modelica.org/documents/ModelicaTutorial14.pdf (Última Visita: 2/9/2012)

- ✓ **[Pulido&Alonso2004] B. Pulido and C. Alonso-González. "Possible conflicts: a compilation technique for consistency-based diagnosis".** En Part B: Cybernetics, IEEE Transactions on Systems, Man, and Cybernetics, 34(5):2192–2206, Oct. 2004.

Capítulo 8:

Anexos

8.1. Introducción

En los anexos dispone de toda la información adicional de la memoria de investigación, a saber:

- **Descripción del Software de Diagnóstico Automático.**
- **Instrucciones de Uso del Software de Diagnóstico Automático.**
- **Código fuente del modelo de sistema AHU-9.**
- **Documentación Software del Módulo Filtro de Partículas**
- **Datos de Simulación del sistema AHU-9.**

8.2. Descripción del software de Diagnóstico Automático

8.2.1. Introducción

El software de Diagnóstico Automático del sistema AHU-9 se define como un conjunto de scripts programados en la plataforma Matlab, que permiten al usuario generar automáticamente experimentos del sistema AHU-9 y el diagnóstico de los mismos.

El sistema se compone de un directorio con una serie de ficheros ".m" con los que se implementa el funcionamiento completo del software.

En este apartado procedemos a definir la lista de ficheros del directorio y la funcionalidad que tienen. En la siguiente sección se diseñará un tutorial de la herramienta donde se describirá cómo modificar estos ficheros para ajustarlos a las necesidades del investigador.

8.2.2. Listado de Ficheros

- **Directorio "ModeloCompleto":**

  - *effectiveness.m* : fichero Matlab que implementa la función genérica de efectividad para la transferencia de calor entre cuerpos. Esta función es empleada por el modelo *ahu9.m* para la simulación de su comportamiento en modo nominal y con fallos inducidos en el parámetro UA.

  - *effectivenessPC.m* : fichero Matlab que implementa la función genérica de efectividad para la transferencia de calor entre cuerpos. Esta función es empleada por los PCs del modelo AHU-9 para la simulación de su comportamiento.

  - *ahu9.m* : fichero Matlab que implementa el modelo de simulación del sistema AHU-9. Define las funciones de transferencia de flujo y calor entre componentes, y ofrece como salida los valores observados por los sensores de temperatura asociados a cada bobina.

- *noise.m* : fichero Matlab que implementa la función genérica empleada para inducir ruido a la salida de los sensores.

- *inputgenerator.m* : fichero Matlab que implementa la función para la generación de ficheros de entrada variables para la simulación de escenarios del sistema AHU-9 a partir de un fichero de entrada.

  Esta función trabaja con la temperatura de los flujos de aire del exterior y reciclado, además de los compartimentos donde se aclimata el líquido entrante a las bobinas, realizando modificaciones aleatorias controladas de las mismas para la creación de nuevos ficheros de entrada.

- *simulation.m* : fichero Matlab que implementa la función generadora de experimentos de simulación del sistema AHU-9 para un determinado fichero de entrada y un escenario de funcionamiento.

- *automaticdiagnosis.m* : fichero Matlab que implementa el interfaz de generación de experimentos y diagnóstico. Trabaja con un fichero de entrada y una lista de ficheros de configuración de escenarios.

  Con esta información, la función invoca por cada configuración de escenario a la función *simulation*, que genera el experimento particular. De este modo, podemos generar todos los escenarios contemplados en una única ejecución del programa.

  Los resultados se almacenan en ficheros de texto, además de generar un *.fig* con la representación gráfica de los experimentos obtenidos.

  Una vez generados los escenarios, la función invoca a *simulationPCs* con cada uno de los ficheros de escenarios de simulación. Así se evalúan los residuos de cada experimento y se determina la activación de PCs en cada escenario.

  De esta manera el usuario se limita a realizar la selección de ficheros de entrada y la configuración de escenarios en una lista; y el sistema efectúa la generación de experimentos y la diagnosis asociada a cada caso de manera automática.

- *ztest_code.m* : fichero Matlab que implementa la función necesaria para aplicar el *ztest* a la señal de residuos obtenida en la labor de diagnosis, de manera que es capaz de detectar la activación del PC asociado a la señal de residuo.

- *PC1*, *PC2*, *PC3* : ficheros Matlab que implementan los modelos de PCs asociados al sistema AHU-9 para aplicar la técnica de diagnosis basada en modelos por medio de PCs.

- *simulationPCs.m* : fichero Matlab que implementa el interfaz de diagnóstico. Trabaja con los ficheros de experimento y entrada asociados al experimento para generar la señal de residuo a partir de la estimación de variables observadas por los PCs. Con la señal generada se aplica el *ztest* para el diagnóstico del sistema.

  Como resultado, el programa informa por fichero de texto plano de la activación de los PCs y el instante en el que ocurre. Además se ofrece una representación gráfica de la señal de residuos y del contraste entre observaciones y estimación para facilitar la comprensión de los resultados.

8.3. Instrucciones de Uso del software de Diagnóstico Automático

El software de diagnóstico automático está diseñado para facilitar la labor de investigación del sistema AHU-9, automatizando ciertas tareas que, de hacerse manualmente, podría resultar repetitivas y tediosas.

El uso del software permite considerar todos los escenarios de simulación que se deseen y efectuar el diagnóstico sobre los mismos. La diagnosis requiere de un proceso de simulación y aplicación del *ztest* el cual exige de tiempos de ejecución muy elevados para finalizar (motivados por la implementación particular del *ztest* y su característica particular de efectuar varias llamadas recursivas dentro de su definición). Cuando ejecute el software tenga en cuenta que el diagnóstico de un escenario de funcionamiento requiere de más de una hora de funcionamiento continuo.

Si desea efectuar un estudio intensivo del sistema, considerando numerosos escenarios de funcionamiento, y trabajando con diferentes ficheros de entrada, la labor de diagnóstico puede alcanzar tiempos del orden de días.

Para llevar a cabo dicha labor se requieren dos ficheros:

- **Fichero de Entradas:** los ficheros de entrada son tablas de datos, necesarias para la simulación del sistema AHU-9 y la generación de escenarios de funcionamiento.

  Por defecto se dispone de un fichero de entradas *in.txt* proporcionado por la organización del DXC13'. Puede recurrir a realizar experimentos con dicho fichero. Dispone del script *inputgenerator.txt* con el que, usando como entrada dicho fichero, y determinando un porcentaje de variación; puede generar nuevos ficheros de entrada donde las temperaturas de entrada al sistema varían aleatoriamente en el porcentaje especificado por el usuario.

  Si desea crear sus propios ficheros de entrada solo debe crear un nuevo fichero donde los datos por línea estén tabulados, y se respete la organización de columnas del fichero de entradas de la organización.

Si desea saber más de dicha organización solo debe consultar el capítulo de estudio del sistema presente en esta memoria, donde se define cada una de las columnas la tabla "input". De esta manera sabrá a qué magnitud de que componente se corresponden los valores de cada columna.

- **Fichero de Lista de Configuraciones:** el fichero de lista de configuraciones consiste en un fichero de texto plano, donde cada línea va asociada a una dirección de un fichero de configuración de escenario de funcionamiento.

```
.\FallosMBPosition\configurationMB5.txt
.\FallosMBPosition\configurationMB20.txt
.\FallosMBPosition\configurationMB50.txt
.\FallosSensores\configurationCS5.txt
.\FallosSensores\configurationCS20.txt
.\FallosSensores\configurationCS50.txt
.\FallosSensores\configurationPS5.txt
.\FallosSensores\configurationPS20.txt
.\FallosSensores\configurationPS50.txt
.\FallosSensores\configurationRS5.txt
.\FallosSensores\configurationRS20.txt
.\FallosSensores\configurationRS50.txt
.\FallosUA\configurationCUA5.txt
.\FallosUA\configurationCUA20.txt
.\FallosUA\configurationCUA50.txt
.\FallosUA\configurationPUA5.txt
.\FallosUA\configurationPUA20.txt
.\FallosUA\configurationPUA50.txt
.\FallosUA\configurationRUA5.txt
.\FallosUA\configurationRUA20.txt
.\FallosUA\configurationRUA50.txt
.\FallosValvePosition\configurationCV5.txt
.\FallosValvePosition\configurationCV20.txt
.\FallosValvePosition\configurationCV50.txt
.\FallosValvePosition\configurationPV5.txt
.\FallosValvePosition\configurationPV20.txt
.\FallosValvePosition\configurationPV50.txt
.\FallosValvePosition\configurationRV5.txt
.\FallosValvePosition\configurationRV20.txt
.\FallosValvePosition\configurationRV50.txt
```

Figura 8.1 Ejemplo de fichero de Lista de Configuraciones

Un fichero de configuración no es más que un script Matlab donde se definen los parámetros asociados a un escenario de fallo. En dichos ficheros el usuario contempla el instante en el que se induce fallo al sistema, los componentes a los que se les induce el fallo, y la magnitud de dicho fallo.

Su diseño es sencillo. Basta con declarar una serie de variables preestablecidas por el sistema con el valor deseado para que el script sea compatible con la herramienta. El elenco de variables contempladas por el sistema es el siguiente:

- ✓ *failInstant*: instante en el que tiene lugar el fallo modelado en el sistema. Asegúrese de considerar un instante de tiempo comprendido dentro del rango de instantes de su experimento.

- ✓ *MixingBoxPosition*: posición del amortiguador del compartimento de mezclado. Especifica en forma de porcentaje el grado de apertura del amortiguador una vez queda atascado. El argumento toma valores entre 0 y 100.

- ✓ *PreheatingSensor*: Especifica el porcentaje relativo de error del sensor asociado a la bobina de precalentamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la magnitud adicional de temperatura que el sensor suma error a los valores de temperatura medidos. Dicha magnitud es el porcentaje del

valor de fallo introducido al sistema aplicado al valor de temperatura medido en cada instante.

- ✓ *CoolingSensor*: especifica el porcentaje relativo de error del sensor asociado a la bobina de enfriamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la magnitud adicional de temperatura que el sensor suma error a los valores de temperatura medidos. Dicha magnitud es el porcentaje del valor de fallo introducido al sistema aplicado al valor de temperatura medido en cada instante.

- ✓ *ReheatingSensor*: especifica el porcentaje relativo de error del sensor asociado a la bobina de recalentamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la magnitud adicional de temperatura que el sensor suma error a los valores de temperatura medidos. Dicha magnitud es el porcentaje del valor de fallo introducido al sistema aplicado al valor de temperatura medido en cada instante.

- ✓ *PreheatingUA:* especifica el porcentaje relativo de error del parámetro UA asociado a la bobina de precalentamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la proporción en la que se ve reducido el parámetro UA del componente.

- ✓ *CoolingUA:* especifica el porcentaje relativo de error del parámetro UA asociado a la bobina de enfriamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la proporción en la que se ve reducida el parámetro UA del componente.

- ✓ *ReheatingUA:* especifica el porcentaje relativo de error del parámetro UA asociado a la bobina de recalentamiento. Especificando un valor entre 0 y 1 (asociado al porcentaje de error), se determina la proporción en la que se ve reducida el parámetro UA del componente.

- ✓ *PreheatingValve:* posición de la válvula de la bobina de precalentamiento. Especifica en forma de porcentaje el grado de apertura de la válvula una vez queda atascada. El argumento toma valores entre 0 y 100.

- ✓ *CoolingValve:* posición de la válvula de la bobina de enfriamiento. Especifica en forma de porcentaje el grado de apertura de la válvula una vez queda atascada. El argumento toma valores entre 0 y 100.

- ✓ *ReheatingValve:* posición de la válvula de la bobina de recalentamiento. Especifica en forma de porcentaje el grado de apertura de la válvula una vez queda atascada. El argumento toma valores entre 0 y 100.

```
% Fallo en los sensores 1 y 2:
% ============================
failInstant = 1000;
CoolingPosition = 5;
```

Figura 8.2 Ejemplo de escenario de fallo

Con los dos ficheros de entrada listos, la ejecución del script es automática:

1. **Ejecute el script "automaticdiagnosis.m".**
2. **Seleccione tantos ficheros de entrada como desee (mantenga pulsado el botón *ctrl* para seleccionar varios ficheros a la vez).**
3. **Seleccione el fichero de lista de configuraciones.**
4. **Espere a que termine la ejecución del script.**

8.4. Código Fuente del modelo de sistema AHU-9:

8.4.1. Introducción

Para facilitar la comprensión del capítulo de la memoria asociado a la descripción del modelo de sistema AHU-9, se ha considerado adjuntar en nuestra memoria el código fuente de dicho sistema.

Se ha omitido la inclusión del código del sistema complejo debido a la extensión del mismo en líneas de código, y su dificultad para ser interpretado.

El código que aquí se presenta está asociado a las bibliotecas del modelo de sistema AHU-9 Simplificado:

8.4.2. Código Fuente

PneumaticPort

```
connector PneumaticPort
  Real T;
  Real mflow;
end PneumaticPort;
```

HydraulicPort

```
connector HydraulicPort
  Real T;
  Real mflow;
end HydraulicPort;
```

AirHandlingUnitMode

```
type AirHandlingUnitMode = enumeration(on , off );
```

HeatExchangerConfiguration

```
type HeatExchangerConfiguration = enumeration(crossFlow, counterFlow , parallelFlow );
```

effectiveness

```
function effectiveness
  input HeatExchangerConfiguration configuration;
  input Real C;
  // capacity rate ratio
  input Real NTU;
  // number of heat transfer units
  output Real eff;
algorithm
  if configuration == HeatExchangerConfiguration.crossFlow then
    eff:=1 - exp(((exp(-C * NTU ^ 0.78) - 1) * NTU ^ 0.22) / C);
  elseif configuration == HeatExchangerConfiguration.parallelFlow then
    eff:=(1 - exp(-NTU * (1 + C))) / (1 + C);

  elseif configuration == HeatExchangerConfiguration.counterFlow then
    if C <> 1 then
      eff:=(1 - exp(-NTU * (1 - C))) / (1 - C * exp(-NTU * (1 - C)));
    else
      eff:=NTU / (1 + NTU);
    end if;
  else

  end if;
end effectiveness;
```

Fan

```
model Fan
  PneumaticPort i;
  PneumaticPort o;
  Real mflow;
equation
  o.T = i.T;
  o.mflow = mflow;
end Fan;
```

GasBoiler

```
model GasBoiler
  HydraulicPort i1;
  HydraulicPort i2;
  HydraulicPort o;
  Real inputTemperature;
  Real outputTemperature;
equation
  o.T = outputTemperature;
  o.mflow = i1.mflow + i2.mflow;
end GasBoiler;
```

Chiller

```
model Chiller
  HydraulicPort i;
  HydraulicPort o;
  Real inputTemperature;
  Real outputTemperature;
equation
  o.T = outputTemperature;
  o.mflow = i.mflow;
end Chiller;
```

HydraulicPump

```
model HydraulicPump
  HydraulicPort i;
  HydraulicPort o;
  parameter Real mflow;
equation
  o.T = i.T;
  o.mflow = mflow;
end HydraulicPump;
```

HeatingCoil

```
model HeatingCoil
  PneumaticPort pi;
  PneumaticPort po;
  HydraulicPort hi;
  HydraulicPort ho;
  parameter Real pneumaticSpecificHeat;
  parameter Real hydraulicSpecificHeat;
  Real nonZeroHydraulicInputFlow;
  Real nonZeroPneumaticInputFlow;
  Real coldSideCapacitanceRate;
  Real hotSideCapacitanceRate;
  Real NTU;
  Real eff;
  Real minCapacitanceRate;
  Real maxCapacitanceRate;
  Real capacitanceRateRatio;
  Real heatRate;
  parameter Real UA;
  // thermal conductance times area
equation
  nonZeroPneumaticInputFlow = max(abs(pi.mflow), 0.001);
  nonZeroHydraulicInputFlow = max(abs(hi.mflow), 0.001);
  hotSideCapacitanceRate = nonZeroHydraulicInputFlow * hydraulicSpecificHeat;
  coldSideCapacitanceRate = nonZeroPneumaticInputFlow * pneumaticSpecificHeat;
  minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
  maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);
  capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
  NTU = UA / minCapacitanceRate;
  eff = effectiveness(HeatExchangerConfiguration.counterFlow, capacitanceRateRatio, NTU);
  heatRate = eff * minCapacitanceRate * (hi.T - pi.T);
  heatRate = hotSideCapacitanceRate * (hi.T - ho.T);
  heatRate = coldSideCapacitanceRate * (po.T - pi.T);
  po.mflow = pi.mflow;
  ho.mflow = hi.mflow;
end HeatingCoil;
```

CoolingCoil

```
model CoolingCoil
  PneumaticPort pi;
  PneumaticPort po;
  HydraulicPort hi;
  HydraulicPort ho;
  parameter Real pneumaticSpecificHeat;
  parameter Real hydraulicSpecificHeat;
  Real nonZeroHydraulicInputFlow;
  Real nonZeroPneumaticInputFlow;
  Real coldSideCapacitanceRate;
  Real hotSideCapacitanceRate;
  Real NTU;
  Real eff;
  Real minCapacitanceRate;
  Real maxCapacitanceRate;
  Real capacitanceRateRatio;
  Real heatRate;
  parameter Real UA;
  // thermal conductance times area
equation
  nonZeroPneumaticInputFlow = max(abs(pi.mflow), 0.001);
  nonZeroHydraulicInputFlow = max(abs(hi.mflow), 0.001);
  hotSideCapacitanceRate = nonZeroHydraulicInputFlow * hydraulicSpecificHeat;
  coldSideCapacitanceRate = nonZeroPneumaticInputFlow * pneumaticSpecificHeat;
  minCapacitanceRate = min(coldSideCapacitanceRate, hotSideCapacitanceRate);
  maxCapacitanceRate = max(coldSideCapacitanceRate, hotSideCapacitanceRate);
  capacitanceRateRatio = minCapacitanceRate / maxCapacitanceRate;
  NTU = UA / minCapacitanceRate;
  eff = effectiveness(HeatExchangerConfiguration.counterFlow, capacitanceRateRatio, NTU);
  heatRate = eff * minCapacitanceRate * (hi.T - pi.T);
  heatRate = hotSideCapacitanceRate * (hi.T - ho.T);
  heatRate = coldSideCapacitanceRate * (po.T - pi.T);
  po.mflow = pi.mflow;
  ho.mflow = hi.mflow;
end CoolingCoil;
```

HydraulicValve

```
model HydraulicValve
  HydraulicPort i;
  HydraulicPort o;
  Real position;
equation
  o.T = i.T;
  o.mflow = position * i.mflow;
end HydraulicValve;
```

MixingBox

```
model MixingBox
  PneumaticPort pi;
  // outdoor
  PneumaticPort pr;
  // return
  PneumaticPort po;
  // output
  Real position;
  Real outdoorFlow;
  Real returnFlow;
equation
  outdoorFlow = pr.mflow * position;
  returnFlow = pr.mflow * (1 - position);
  po.mflow = outdoorFlow + returnFlow;
  po.mflow * po.T = outdoorFlow * pi.T + returnFlow * pr.T;
  pi.mflow = 0;
end MixingBox;
```

Main

```
model Main
  parameter Real waterSpecificHeat = 4186;
  parameter Real airSpecificHeat = 1006;
  Fan returnFan;
  Outside outside;
  MixingBox mixingBox;
  Chiller chiller;
  HydraulicPump chillerPump(mflow = 1.5);
  GasBoiler gasBoiler;
  HydraulicPump boilerPump(mflow = 1.5);
  HydraulicValve preheatingCoilValve;
  HeatingCoil preheatingCoil(hydraulicSpecificHeat = waterSpecificHeat, pneumaticSpecificHeat = airSpecificHeat, UA = 320 * 20);
  HydraulicValve coolingCoilValve;
  CoolingCoil coolingCoil(hydraulicSpecificHeat = waterSpecificHeat, pneumaticSpecificHeat = airSpecificHeat, UA = 320 * 20);
  HydraulicValve reheatingCoilValve;
  HeatingCoil reheatingCoil(hydraulicSpecificHeat = waterSpecificHeat, pneumaticSpecificHeat = airSpecificHeat, UA = 320 * 20);
  Fan supplyFan;
  Zone zone;
  TemperatureSensor preheatingOffCoilTemp(name = "preheatingOffCoilTemp");
  TemperatureSensor coolingOffCoilTemp(name = "coolingOffCoilTemp");
  TemperatureSensor reheatingOffCoilTemp(name = "reheatingOffCoilTemp");
  // Input data:
  Sources.CombiTimeTable inputs(tableOnFile = true, tableName = "tab1", columns = 2:12, fileName = "c:~");
  // AHU state:
  inner AirHandlingUnitMode AHUMode;
algorithm
  preheatingCoilValve.position:=inputs.y[1] / 100;
  coolingCoilValve.position:=inputs.y[2] / 100;
  reheatingCoilValve.position:=inputs.y[3] / 100;
  mixingBox.position:=inputs.y[4] / 100;
  outside.T:=inputs.y[5];
  zone.T:=inputs.y[6];
  supplyFan.mflow:=inputs.y[7];
  returnFan.mflow:=inputs.y[7];
  gasBoiler.outputTemperature:=inputs.y[8];
  gasBoiler.inputTemperature:=inputs.y[9];
  chiller.outputTemperature:=inputs.y[10];
  chiller.inputTemperature:=inputs.y[11];
  if supplyFan.mflow > 0.5 then
    AHUMode:=AirHandlingUnitMode.on;
  else
    AHUMode:=AirHandlingUnitMode.off;
  end if;
equation
  connect(outside.o,mixingBox.pi);
  connect(returnFan.o,mixingBox.pr);
  connect(mixingBox.po,preheatingCoil.pi);
  connect(preheatingCoil.po,coolingCoil.pi);
  connect(coolingCoil.po,reheatingCoil.pi);
  connect(reheatingCoil.po,supplyFan.i);
  connect(supplyFan.o,zone.i);
  connect(zone.o,returnFan.i);
  connect(preheatingCoil.po,preheatingOffCoilTemp.i);
  connect(coolingCoil.po,coolingOffCoilTemp.i);
  connect(reheatingCoil.po,reheatingOffCoilTemp.i);
  connect(gasBoiler.o,boilerPump.i);
  connect(chiller.o,chillerPump.i);
  connect(boilerPump.o,preheatingCoilValve.i);
  connect(preheatingCoilValve.o,preheatingCoil.hi);
  connect(preheatingCoil.ho,gasBoiler.i1);
  connect(chillerPump.o,coolingCoilValve.i);
  connect(coolingCoilValve.o,coolingCoil.hi);
  connect(coolingCoil.ho,chiller.i);
  connect(boilerPump.o,reheatingCoilValve.i);
  connect(reheatingCoilValve.o,reheatingCoil.hi);
  connect(reheatingCoil.ho,gasBoiler.i2);
end Main;
```

Outside

```
model Outside
   PneumaticPort o;
   Real T;
equation
   o.T = T;
end Outside;
```

Zone

```
model Zone
  PneumaticPort i;
  PneumaticPort o;
  Real T;
equation
  o.T = T;
  o.mflow = i.mflow;
end Zone;
```

TemperatureSensor

```
model TemperatureSensor
  parameter String name;
  PneumaticPort i;
  outer AirHandlingUnitMode AHUMode;
initial algorithm
  Files.remove("C:/Users/Modelica/Modelica Models/AHU-9 Simple/ahu-9.out");
algorithm
  if AHUMode == AirHandlingUnitMode.on then
    Streams.print(String(time) + ": " + name + " = " + String(i.T), "C:/Users/Modelica/Modelica Models/
  else

  end if;
end TemperatureSensor;
```

8.5. Documentación Software del Módulo Filtro de Partículas

8.5.1. Introducción

En los capítulos introductorios de la memoria de investigación introdujimos el concepto de Filtro de Partículas y su utilidad para nuestro estudio.

El FP permite la estimación del estado interno de nuestro sistema, o lo que es lo mismo, el valor asociado a las variables de estado de nuestro sistema de estudio.

Aunque en nuestro sistema no se contemplaron variables de estado, al ser un sistema que no necesita de la definición de diferentes instantes de tiempo para modelar su comportamiento; se consideró el desarrollo de software para la integración del FP en el inicio nuestra investigación.

Desde un principio se disponía de una implementación de un FP genérico denominado “ISIS”, ya que fue desarrollado en el laboratorio ISIS de la Universidad de Vanderbilt bajo la supervisión de los profesores Gautam Biswas y Xenofon Kotsoukos; diseñado para ser ejecutado bajo plataforma Matlab; y el PCsDX, una herramienta para el diagnóstico automático de sistemas, también diseñada para Matlab [Hernández2012].

Con estas herramientas disponibles, nuestra tarea inicial fue la de desarrollar un módulo para la aplicación del FP a la labor de diagnosis desempeñada por la herramienta.

Este apartado del anexo se dedicará a realizar una breve documentación software sobre el Módulo FP implementado para el sistema, de manera que se refleje la labor de desarrollo software realizada.

El contenido del siguiente apartado revisa la documentación software de la memoria asociada al software PCsDX, señalando las modificaciones a la misma realizadas para documentar la inclusión del nuevo módulo al sistema.

8.5.2. Documento Software

1. Análisis:

En este apartado vamos a declarar los nuevos objetivos del sistema e incluir los nuevos requisitos contemplados por el módulo de FP. Dado que estamos trabajando en la etapa de análisis, los cambios en la documentación van a ser mínimos, pero en ellos se reflejará la inclusión del FP en nuestro sistema.

a. Objetivos:

El objetivo del Módulo de Filtro de Partículas es el de compatibilizar la labor de diagnóstico automático realizado por la herramienta PCsDX con la implementación del FP genérico "*ISIS*".

- ***OBJ-1: Implementación de un módulo de Filtro de Partículas ejecutable en la herramienta PCsDX:***

  Compatibilizar la labor de diagnóstico del software PCsDX con la aplicación de un FP para la estimación del estado interno de los sistemas entrantes. Con ello se busca ofrecer al usuario la posibilidad de efectuar una diagnosis donde la estimación realizada por los PCs aplique la funcionalidad de esta herramienta teórica, y no sea solo una simulación

b. Requisitos del Sistema:

Recogiendo el documento de requisitos del sistema PCsDX, la funcionalidad del módulo de diagnosis vendría recogida en el requisito funcional RF-9:

RF-9	Efectuar Diagnosis
Descripción	El sistema deberá realizar la diagnosis del sistema en un escenario dado, detectando la presencia de fallo, y localizando y aislando al mismo.
Prioridad	Alta
Estado	Aprobado

En este requisito se define la necesidad del sistema de efectuar la labor de diagnóstico. Sin embargo no se especifica su mecanismo de diagnóstico ni el uso del FP.

Estamos trabajando con documentación de análisis, por lo que está abordándose de manera muy genérica la solución al problema de diagnóstico automático. Hablar en estas líneas de FP sería entrar en detalles más propios del diseño o la implementación. Sin embargo, como el módulo se basa en compatibilizar el funcionamiento de dos herramientas, resulta apropiado incluir un requisito no funcional adicional donde se refleje dicha característica:

RNF-11*	Compatibilidad
Descripción	El sistema deberá ser compatible con el FP ISIS para efectuar la labor de diagnóstico.
Prioridad	Alta
Estado	Aprobado

c. *Casos de Uso:*

La nueva funcionalidad del sistema no representa ningún caso de uso nuevo, sino que se ve incluida dentro de la definición del caso de uso *UC-0004 "EfectuarDiagnosis"*, donde se especifica que el sistema efectúa la labor de diagnóstico.

Este caso de uso debe ser modificado acorde al nuevo significado del caso de uso. Vamos a redefinir la descripción del caso añadiendo un paso adicional donde el usuario del sistema tenga la opción de configurar el método de estimación o simulación del comportamiento necesario para el diagnóstico según sus necesidades. En nuestro caso particular esto supone entre alternar el uso de la funcionalidad de diagnóstico con y sin FP, pero en un futuro sería probable que se introdujesen nuevos FP, de manera que existirían diversos tipos de configuraciones.

UC-0004	EfectuarDiagnosis	
Dependencias	Actor Usuario(ACT-0001)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario desee efectuar la labor de diagnosis de un sistema concreto*	
Precondición	Resultados del escenario de experimento asociado al Modelo de Sistema a diagnosticar disponible, Modelo de Simulación de PCs, en notación de Estados o DBN, disponible	
Secuencia normal	Paso	Acción
	1	El caso de uso comienza cuando el actor Usuario(ACT-0001) *selecciona EfectuarDiagnosis*
	2	El sistema *solicita el escenario de sistema a cargar*
	3	El actor Usuario(ACT-0001) *introduce el escenario de sistema a diagnosticar*
	4*	El actor Usuario(ACT-0001) *introduce la configuración del diagnóstico*
	5	El sistema *efectúa la diagnosis*
	6	El sistema devuelve los resultados
Postcondición	Se obtiene el diagnóstico del sistema	
Excepciones	Paso	Acción
	3	Si el actorUsuario(ACT-0001)*declina la carga del escenario el sistema se detiene y el caso de uso queda sin efecto*
Comentarios	Requisitos:RF9,RF10,RF11,RF12 A la par que el sistema efectúa la diagnosis va almacenando los resultados con el fin de poder atender a las posteriores consultas efectuadas en los casos de uso UC-0006 y UC-0007	

d. Modelo Estructural del Sistema:

El modelo estructural de nuestro sistema se mantiene inalterado. El diagrama de clases inicial contemplaría el FP dentro de su clase *SistemaDiagnosis*. Llegados a la etapa de diseño e implementación se detallarán las clases que intervienen en esta nueva funcionalidad.

Por el momento, descartamos hablar del FP dado que estamos hablando de un software ya existente. Nuestra aportación al proyecto son las clases que permiten la compatibilización del FP con el software de diagnóstico.

2. Diseño:

El diseño de nuestro sistema es otro de los aspectos que se mantiene prácticamente inalterado. El sistema PCsDX se concebió bajo unos principios de

extensibilidad que se respetaron a lo largo de todo su proceso de elaboración. De esta propiedad deriva el hecho de que extender la funcionalidad básica del sistema no suponga efectuar ningún cambio significativo en su arquitectura lógica.

La inclusión del módulo FP no supone modificar la arquitectura lógica de capas de nuestro sistema, ni su diseño físico. Para reflejar los cambios en diseño del nuevo sistema basta con reflejar en la documentación el paquete software asociado al FP y cómo se integra el mismo en la arquitectura.

El FP es un software auxiliar compuesto por una serie de clases con funciones que modelan dicha herramienta teórica. Nuestra herramienta trabajará de tal manera que invocará al filtro de forma totalmente ajena a su implementación. Estamos hablando de un paquete software de utilidades.

Este paquete se encuadra dentro de la capa de utilidades, la cual es subyacente al resto de capas, y proporciona funcionalidades adicionales tales como conexión a SGBD o las herramientas necesarias para la visualización del interfaz gráfico.

Dicho esto, solo hay que reflejar la inclusión de dicho paquete en la arquitectura. En lo que representa al modelo de despliegue, y el diagrama de clases de diseño, con sus correspondientes diagramas de interacción, el sistema se mantiene sin cambios, dado que el paquete software introducido no pertenece a las capas de presentación, lógica de negocio ni modelo de datos.

a. *Diseño Lógico de la Arquitectura de Aplicación:*

El nuevo diseño lógico de la Arquitectura de Aplicación debe contemplar la integración del módulo FP del sistema, y las dependencias del mismo con el resto de paquetes.

Dicho esto, el nuevo diagrama de paquetes resulta ser de la siguiente forma:

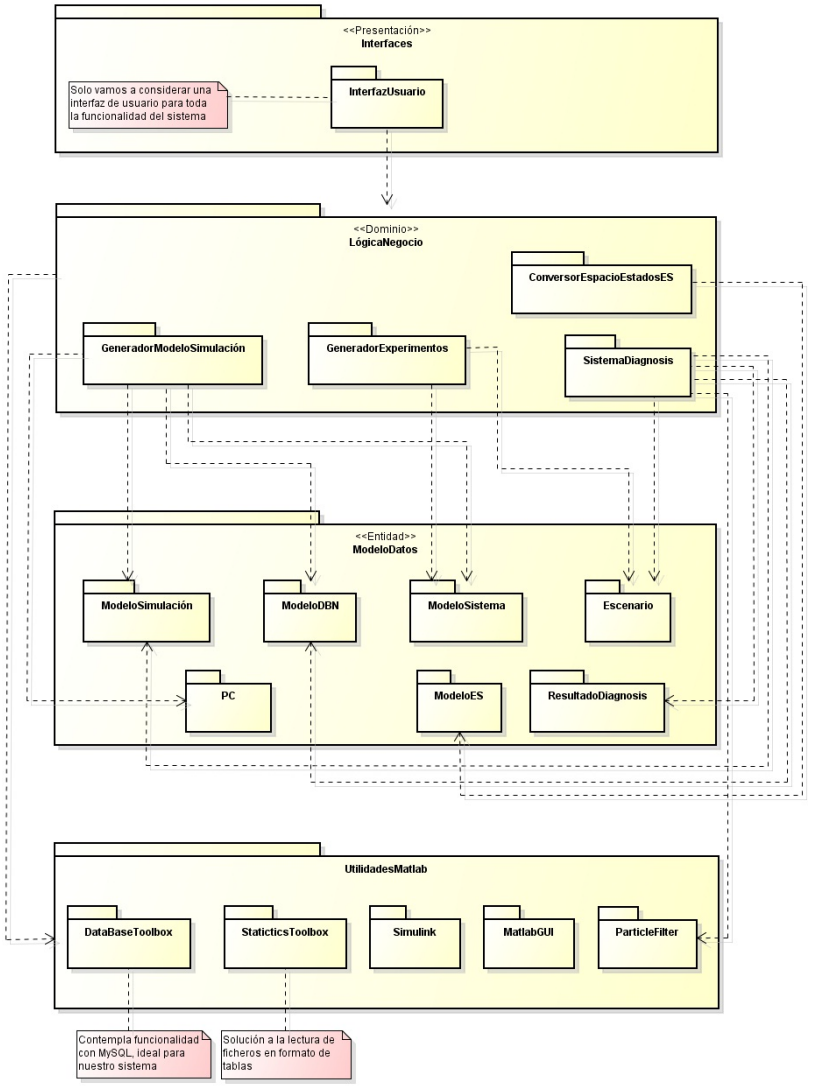


Figura 8.3 Diagrama de Paquetes

Con el diagrama de paquetes redefinido solo falta incluir la definición del paquete *ParticleFilter*:

- ***ParticleFilter:*** contempla la funcionalidad necesaria para la ejecución de un Filtro de Partículas genérico en el proceso de simulación de PCs.

b. *Diagrama de Clases de Diseño:*

El diagrama de clases de diseño no sufre ninguna alteración con la inclusión del módulo de FP, sin embargo, la dependencia de dicho módulo

con la clase *SistemaDiagnosis* motiva a que se tenga que redefinir las características de dicha clase:

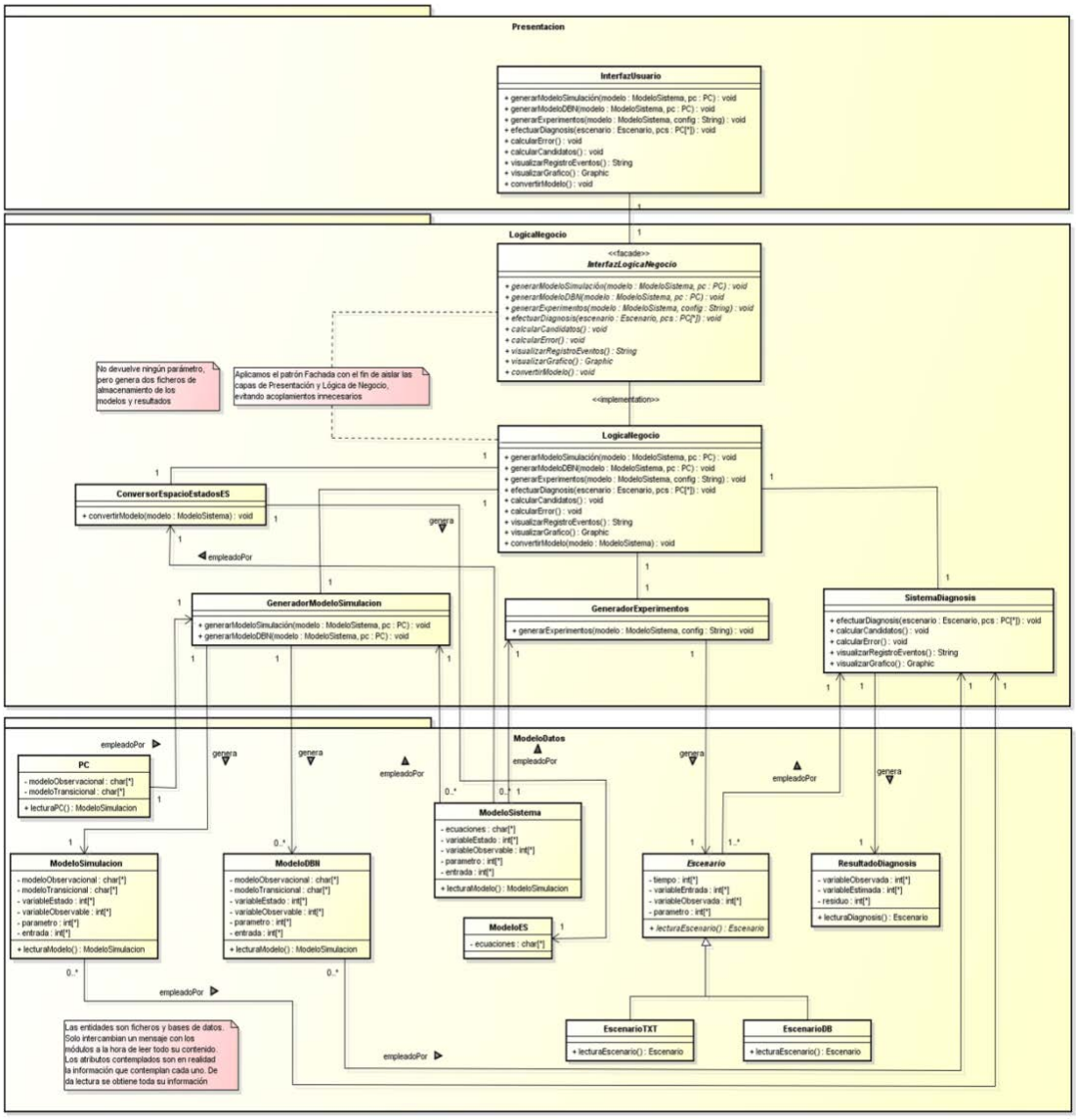


Figura 8.4 Diagrama de Clases de Diseño

SistemaDiagnosis	
Responsabilidades:	
Implementa la lógica de Diagnosis de sistemas a partir de un escenario concreto y los PCs asociados al sistema a diagnosticar; almacena los resultados en un fichero con la sintaxis apropiada	
Atributos:	
Operaciones:	
efectuarDiagnosis(escenario : Escenario, pcs : PC[*], configuration: String) : void	
Objetivo:	Implementa el servicio de efectuar la labor de diagnosis de un sistema representado en un escenario calculado previamente, recurriendo al método de FP seleccionado en la configuración de la diagnosis
Entradas:	Escenario sobre el que efectuar la diagnosis y PCs contemplados en el sistema. Configuración de la diagnosis
Salidas:	No retorna parámetros. Genera un fichero donde almacena la solución. Notifica a la clase subyacente en caso de error
calcularCandidatos() : string	
Objetivo:	Implementa el servicio de efectuar la detección de candidatos a fallo en la labor de diagnosis
Entradas:	Ninguna, las entradas necesarias para la diagnosis ya deben haber sido introducidas previamente para el paso inicial de efectuar diagnosis
Salidas:	Retorna la lista de candidatos a fallo. Notifica a la clase subyacente en caso de error.
calcularError() : double	
Objetivo:	Implementa el servicio de efectuar la labor de cálculo de los residuos de la diagnosis del sistema
Entradas:	Ninguna, las entradas necesarias para la diagnosis ya deben haber sido introducidas previamente para el paso inicial de efectuar diagnosis
Salidas:	Retorna los valores residuales calculados
visualizarRegistroEventos() : String	
Objetivo:	Retornar la información suficiente como para que la clase subyacente sea capaz de mostrar el Log de la diagnosis al usuario
Entradas:	Ninguna, la información del registro queda recogida por la aplicación conforme diagnostica el sistema en cuestión
Salidas:	Cadena de texto donde se detalla el registro, será mostrada por pantalla. Notifica a la clase subyacente en caso de error
visualizarGrafico() : Graphic	
Objetivo:	Retornar la información suficiente como para que la clase subyacente sea capaz de generar el gráfico con los resultados de la diagnosis
Entradas:	Ninguna, la información del gráfico queda recogida por la aplicación conforme diagnostica el sistema en cuestión
Salidas:	Gráfico donde se detallan los resultados, será mostrado por pantalla. Notifica a la clase subyacente en caso de error

3. Implementación:

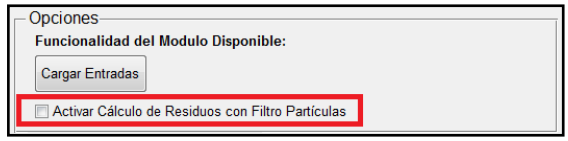
La inclusión del FP ha supuesto la realización de ciertos cambios en las capas de presentación y lógica de negocio de nuestro sistema. Vamos a documentar brevemente dichos cambios para dar a conocer la labor de programación que fue necesaria llevar a cabo para la integración del módulo en nuestro sistema:

1. *Interfaz de Usuario:*

La inclusión de la opción de simular PCs recurriendo al FP requiere de modificar el interfaz de manera que sea visible dicha opción, y que se permita seleccionar la misma.

La solución a este problema ha sido la de añadir un "checkbox" cuya selección habilite la simulación con FP. Se trata de una solución AD-HOC adecuada para nuestro caso particular. También se podrían haber planteado soluciones de carácter más extensible, como una lista múltiple, de cara a mostrar varios tipos de simulación de PCs diferentes y permitir al usuario seleccionar el que más le convenga.

Como en este punto del desarrollo solo disponemos de una configuración particular de simulación, y nuestra labor está enfocada a la investigación, consideramos que esta solución es la más adecuada. Además, el cambio de tipo de selección no es una labor muy costosa, por lo que la solución "checkbox" no supone problemas de cara a futuras revisiones del código



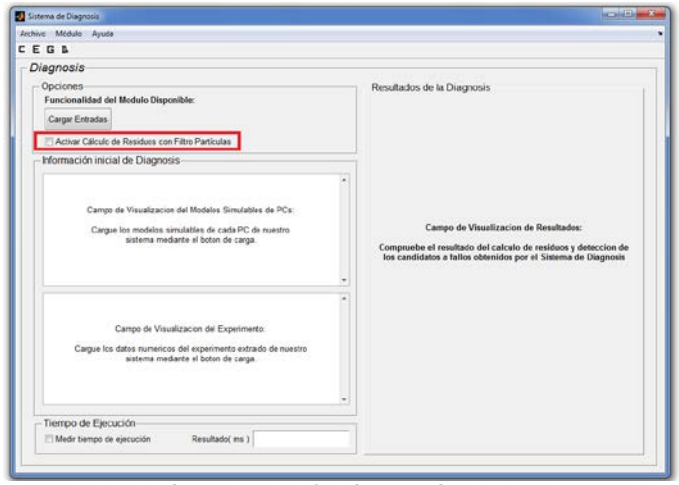


Figura 8.5 Interfaz de Usuario

2. *Lógica de Negocio:*

La capa de lógica de negocio se ve modificada fruto de la nueva dependencia entre el sistema de diagnosis y el módulo Filtro de Partículas.

La labor de diagnóstico automático se sigue definiendo en la misma sucesión de etapas: estimación de residuos y cálculo de candidatos. Sin embargo ahora en la estimación del comportamiento previo al cálculo de los valores de los residuos estamos recurriendo al Filtro de Partículas.

Este módulo funciona con entradas particulares. Cada sistema debe adaptar su notación estándar a una compatible con el Filtro de Partículas. Tradicionalmente, el uso de este módulo exigía al usuario conocer la notación de simulación necesaria para ejecutar el filtro, y reescribir sus sistemas acordes a su notación.

Dicha labor es tediosa y puede mecanizarse. Dado que queremos integrar el módulo del filtro en nuestro sistema, la modificación a realizar en la lógica de negocio es la de implementar una funcionalidad adicional que permita la conversión de modelos a notación compatible con el filtro.

En lo que respecta a las salidas, los resultados del filtro son compatibles con los de la herramienta, por lo que no hace falta contemplar ninguna modificación referida a dicho aspecto.

Se han diseñado dos funciones adicionales en nuestro sistema:

- *generadorFiltro:* genera un filtro genérico ISIS asociado a un modelo de sistema concreto para su simulación.

  Esta función es la encargada de generar un fichero auxiliar con el cual el sistema va a invocar la simulación del modelo una vez se proceda a efectuar el cálculo de residuos. La idea es utilizar un modelo de entrada adaptado a la notación del FP para crear un script donde se ajusten los parámetros del filtro (instantes, número de partículas, etc).

  Mediante dicho script se efectúa una simulación completa de los PCs y se obtiene la estimación de los nodos discrepancia para la obtención de los residuos.

- *procesadorModelos:* trabaja con los ficheros de modelos de sistema entrantes en notación DBN para adaptar los mismos a una notación compatible con la del filtro.

  Genera ficheros auxiliares que luego emplea el generador del filtro para incluir el modelo de sistema entrante en la descripción del filtro.

Estas dos funciones se han añadido a la clase *SistemaDiagnosis*. Los ficheros auxiliares que generan son los que invocan a la funcionalidad del paquete software asociado al FP.

Se ha modificado también la función de cálculo de residuos, la cual se ejecuta en un modo normal y un modo FP en función de un parámetro de entrada, que el sistema asocia al checkbox de la opción de FP.

De esta manera se logra una integración completa del filtro en el sistema. Si se desea eliminar la misma basta con borrar las funciones añadidas y los puntos de ejecución donde se invoca a dicha funcionalidad.

Por otro lado, para extender la funcionalidad del sistema a otros filtros, basta con definir nuevas funciones de generadores de filtro con una notación que se ajuste al filtro en cuestión. Con ello es posible integrar nuevos filtros sin modificar ninguna de las características ya disponibles en el software.

4. Pruebas:

    1. Metodología del Documento de Pruebas**:**

        Para la elaboración de los casos de prueba utilizaremos las “técnicas de pruebas de caja negra”.

        En los casos de prueba sobre la funcionalidad del sistema, se realizarán dos pruebas por cada requisito de análisis contemplado, acorde a la metodología RUP (prueba positiva y prueba negativa). En nuestro caso, las pruebas se harán exclusivamente en referencia al requisito *RF-9 Efectuar Diagnosis*, empleando el FP (dado que el resto de comprobaciones a realizar ya vienen recogidas en la memoria del software PCsDX).

        En los casos de prueba complementarios a las especificaciones de los requisitos se determinarán un elenco de situaciones alternativas a la de funcionamiento correcto para ver como el sistema es capaz de responder ante ellas.

        Si se detecta alguna anomalía relevante dentro del conjunto de pruebas que determinan el correcto funcionamiento de la lógica interna del sistema; el caso de uso correspondiente al fallo será comentado en dos instancias: la instancia de fallo inicial y la posterior instancia del caso de prueba una vez corregido el error.

    2. Sistema de Pruebas**:**

        Para efectuar cada uno de los casos de prueba que se contemplarán se empleará un mismo modelo de sistema, el cual se viene recogiendo en ejemplos desde el inicio del documento; y un mismo hardware.

Se ha detectado que en otros sistemas Matlab se han dado fallos de compatibilidad en el manejo de objetos. Con el fin de aclarar el contexto funcional empleado para redactar el documento de pruebas, a continuación se expone una lista con la configuración software y hardware empleada para el testeo de la aplicación:

- **Hardaware:**
  - Procesador Intel Core 2 Duo CPU P8400 A 2.27 GHz.
  - Memoria RAM de 4.00 GB

- **Software:**
  - Sistema Operativo Windows 7 Ultimate (x32)
  - Matlab Version 7.8.0 (R2009a)

- **Modelo de Sistema:**

  Modelo de Sistema de Tres Tanques Interconectados.

1. Resultados de los Casos de Prueba**:**

   A continuación se presenta una lista con los diversos casos de prueba considerados y los resultados de los mismos. Estos serán presentados en forma de tabla con los siguientes campos:

CP-0i	Nombre del Caso de Prueba
Objetivo	Intenciones que se persiguen al efectuar la prueba
Prueba	Explica en qué consiste la prueba
Salida Esperada	Detalla la salida considerada en los requisitos funcionales que debería obtenerse al efectuar la prueba
Salida Obtenida	Detalla la salida obtenida del sistema al efectuar la prueba
Resultado	Conclusión de haber efectuado la prueba

CP-1	Calcular Residuos (con Filtro de Partículas)
Objetivo	Comprobar que el sistema es capaz de efectuar el cálculo de residuos a partir de la información entrante, teniendo activada la opción de FP.
Prueba	Ejecutamos la opción "Calcular Residuos" previa carga de un conjunto de entradas correctas, , teniendo activada la opción de FP previamente.
Salida Esperada	El sistema genera los residuos y los muestra por pantalla en forma de tabla. Adjunta a la estimación de los residuos, se mostrará una tabla con la estimación de cada nodo discrepancia. Se habilitará la opción
Salida Obtenida	Salida Esperada.
Resultado	La aplicación funciona correctamente.

CP-2	Calcular Residuos con Entradas Erróneas (con Filtro de Partículas)
Objetivo	Comprobar que el sistema es capaz de responder apropiadamente cuando las entradas para el cálculo de residuos son erróneas, teniendo activada la opción de FP.
Prueba	Ejecutamos la opción "Calcular Residuos" previa carga de un conjunto de entradas erróneas, teniendo activada la opción de FP previamente.
Salida Esperada	El sistema se mantendrá sin cambios, permitiendo al usuario repetir la acción y se mostrará un mensaje de error notificando el motivo de fallo. No se habilitará ninguna opción del modelo.
Salida Obtenida	Salida Esperada.
Resultado	La aplicación detecta el fallo e informa del mismo al usuario.

5. Conclusiones:

Integrar el FP en la herramienta PCsDX requiere de una labor de ingeniería software escasa, motivada por el carácter extensible de dicho software.

En esta ocasión, hemos aprovechado las propiedades de la arquitectura lógica de capas y los sistemas modulares para extender la funcionalidad básica de nuestro sistema sin apenas modificar su estructura interna.

Disponíamos por defecto de una implementación del FP; por lo que nuestra labor se limitaba a crear una funcionalidad adicional que sirviese de "puente" entre la herramienta software y el filtro de manera que se compatibilizasen las entradas y salidas de ambos.

Dicha labor requiere comprender las entradas a cada sistema y ser capaces de generar los ficheros de modelos de sistemas con una notación compatible y acorde a cada módulo funcional.

Se trata de una labor nada trivial, dada la complejidad del filtro. A pesar de ello los resultados del trabajo han sido positivos. Se ha conseguido implementar dicha compatibilidad, integrando el FP en la herramienta sin producir ningún cambio en su estructura interna.

Si se desea trabajar con otras posibles implementaciones del filtro, en un futuro; bastará con implementar la funcionalidad que compatibilice las entradas y salidas con dicho filtro, para incluir el mismo en la herramienta.

8.6. Datos de Simulación del sistema AHU-9

Dado que se ha realizado una cantidad considerable de experimentos sobre el sistema AHU-9, y se han diagnosticado cada uno de dichos escenarios de funcionamiento, resulta oportuno incluir en el proyecto todos los datos obtenidos en la labor de investigación.

Dichos datos están contenidos dentro del directorio *Datos* del CD adjunto a la memoria de investigación. En el dispone de una lista de ficheros asociados a los resultados de la simulación de escenarios de funcionamiento y el diagnóstico de los mismos.

Existen cinco ficheros de entrada diferentes: *in.txt, in10%.txt, in30%.txt, in50%.txt e in70%.txt*. Por cada uno de estos ficheros existe un fichero experimental asociado al modo nominal y cada uno de los modos de fallo contemplados.

Puede identificar el fichero asociado a la entrada deseada en el escenario escogido por el nombre de ficheros. Hemos concatenado para cada caso el nombre del fichero de entrada con el nombre de la configuración del escenario de funcionamiento. De esa manera no necesitará consultar los datos del fichero para reconocer de qué experimento trata cada archivo.

Por último, los resultados del diagnóstico se almacenan en ficheros con terminaciones *residuo* y *diagnóstico*. En ellos dispone de los residuos generados en la diagnosis y el momento de activación de cada PC, respectivamente. Estos ficheros se identifican de la misma manera que los experimentales, por lo que consultando el nombre del fichero sabe a qué escenario se asocian los resultados del diagnóstico.

Los datos están almacenados en ficheros de texto plano, con las entradas tabuladas. Esto se ha hecho así de cara a que el lector tenga acceso a la información de los mismos desde cualquier dispositivo. Para facilitar el trabajo con los ficheros, cada tabla tiene adjunto un *.fig* donde se almacena la representación gráfica de los datos que contiene. Esta extensión es solo compatible con Matlab, pero permite su exportación a cualquier formato de imagen conocido.

Los datos están también preconfigurados para ser compatibles con la función *importdata* de Matlab. De esta manera el usuario puede cargar las tablas de manera automática en una matriz, especificando solamente la ruta del fichero que contiene la tabla.

Con esta propiedad se busca facilitar la manipulación de los datos, y permitir al usuario el exportar los mismos en otros formatos, como CSV o entrada a una BD, de manera que pueda elegir el soporte técnico de manipulación de datos que desee.