UNIVERSIDAD COMPLUTENSE DE MADRID

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ParaTrough: Librería en Modelica para el modelado y simulación de plantas termosolares de colectores cilindro-parabólicos

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

ParaTrough: Simulación y Modelización Termosolar

Autor: Juan Antonio Romera Cabrerizo

Director: Alfonso Urquía Moraleda

Curso 2014-2015

Convocatoria de Septiembre

Universidad Complutense de Madrid

Universidad Nacional de Educación a Distancia

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

ParaTrough: Librería en *Modelica* para el modelado y simulación de plantas termosolares de colectores cilindro-parabólicos

Autor: Juan Antonio Romera Cabrerizo

Director: Alfonso Urquía Moraleda

Proyecto de tipo B: Proyecto específico propuesto por el alumno

Autorización

Se autoriza a la Universidad Complutense de Madrid y a la Universidad Nacional de Educación a Distancia (UNED) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: Juan Antonio Romera Cabrerizo

Firma del autor

Resumen del proyecto:

El presente Trabajo Fin de Master (TFM) es una librería escrita en código abierto *Modelica* y utilizando el interfaz gráfico *Dymola 6.1* para modelar y simular plantas termosolares de tecnología de colector cilindro-parabólico (CSP).

Los actuales software de modelado y simulación cada vez son más potentes debido a los avances en computación y programación, consiguiendo estimaciones precisas del comportamiento de plantas termosolares sin precedentes. No obstante, tienen la desventaja de tener poca flexibilidad si se quieren utilizar en aplicaciones alejadas del estándar y adaptarse a las necesidades de cada usuario, aspecto común en el caso de la industria de energía renovables. Las licencias de adquisición de estos sofware son caras para un producto en el que no se tiene la seguridad de que sea adecuado para la aplicación concreta que busca el usuario final.

La librería *ParaTrough* se ofrece como una herramienta pública gratuita, de código libre flexible, modular y fácilmente ampliable y modificable a las exigencias de cada planta y proceso en particular. En la versión inicial contemplada en este trabajo, esta librería se puede usar para el modelado y simulación del recurso solar y del sistema de fluido de transferencia calorífica. Los modelos han sido validados con datos reales de una planta en operación. En futuras versiones se añadirán los sistemas de almacenamiento térmico, bloque de potencia y sistemas auxiliares.

ParaTrough se estructura en las sub-librerías de los sistemas disponibles: recurso solar (*A_SolarResource*) y sistema de fluido de transferencia calorífica (*B_HeatTransferFluid*). Además se disponen de sub-librerías auxiliares con modelos de medios materiales (*media*) y modelos básicos genéricos (*basics*).

El objetivo es que *ParaTrough* pueda ser utilizado por analistas de procesos para uno o varios de los siguientes casos: la evaluación del rendimiento, la detección de fallos, la exploración de nuevos modos de operación y la optimización de la planta.

Palabras clave: Modelización, simulación, termosolar, cilindro-parabólico

Abstract:

The following Master Thesis is a library written in open code *Modelica* using the graphical interface *Dymola 6.1* to model and simulate parabolic trough thermosolar power plants (CSP).

The current modeling and simulating software is increasingly more powerful due to the present-day advances in computation and programming, achieving unprecedented accurate estimations of the thermosolar power plants behavior.

Nevertheless, they have the disadvantage of having little flexibility when they want to be used in no-standard applications and have to be adapted to the needs of a indivudual user. This is a common aspect in the case of the renewable energy industry. The acquiring licenses are expensive taking into account that the software validity is not assured for specific applications.

The *ParaTrough* library is shown as a public, free coded, modular, easily extended tool which is modifiable to the exigencies of each plant and particular process. In the initial version developed in this thesis, this library can be used to model and simulate the solar resource and the heat transfer fluid system. The models have been validated with real data of a plant in operation. The systems of energy storage, power block and balance of plant will be added in further versions.

ParaTrough is structured in the corresponding sub-libraries of the available systems: *A_SolarResource* and *B_HeatTransferFluid*. In addition, the auxiliary sub-libraries of material media (*media*) and generic basic models (*basics*) are available as well.

ParaTrough can be used by process analysts for one or more of the next aims: performance evaluation, failure detection, new operation modes exploration and plant optimization.

Keywords: Modeling, simulation, thermosolar, parabolic-trough

Tabla de contenido

Lista de figuras

Lista de tablas

1. Introducción, objetivos y estructura

1.1. Introducción

Una planta termosolar es una planta de producción de energía eléctrica mediante una turbina de vapor que opera en un ciclo Rankine clásico con economización, generación, sobrecalentamiento y recalentamiento [1]. El concepto es totalmente análogo a cualquier planta térmica como por ejemplo de carbón o nuclear. La gran diferencia estriba en que la fuente primaria de aporte energético es la radiación directa de origen solar en lugar de la quema de un combustible o una reacción de tipo nuclear.

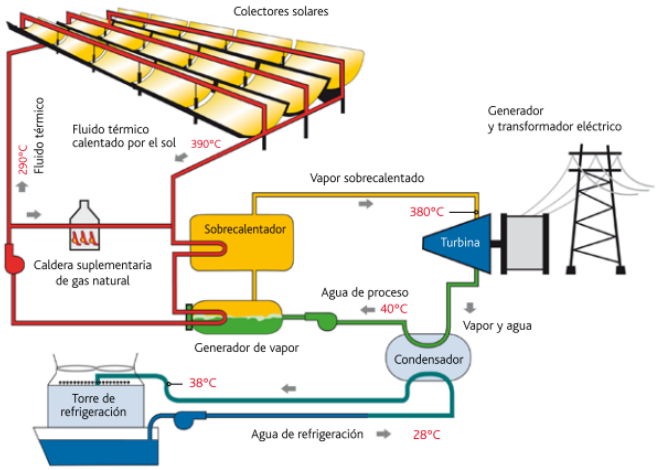


Figura 1.1: Esquema de una planta CSP sin almacenamiento térmico

Como se puede observar en la Figura 1.1, [2]; el campo solar concentra la radiación solar directa transmitiendo este calor a un fluido térmico que a su vez cede el calor a través del tren de generación (precalentador, generador de vapor, sobrecalentador y recalentador) a un circuito de vapor. Este vapor acciona la turbina para producir electricidad en un generador. El vapor es condensado a la salida de la turbina mediante un condensador refrigerado por agua. El condensado retorna al tren de generación para volver a repetir el ciclo.

Algunas plantas CSP poseen un sistema de almacenamiento térmico que permite a la central operar en tramos donde no hay radiación solar o durante varias horas nocturnas. La configuración de este tipo de plantas se muestra en la Figura 1.2.

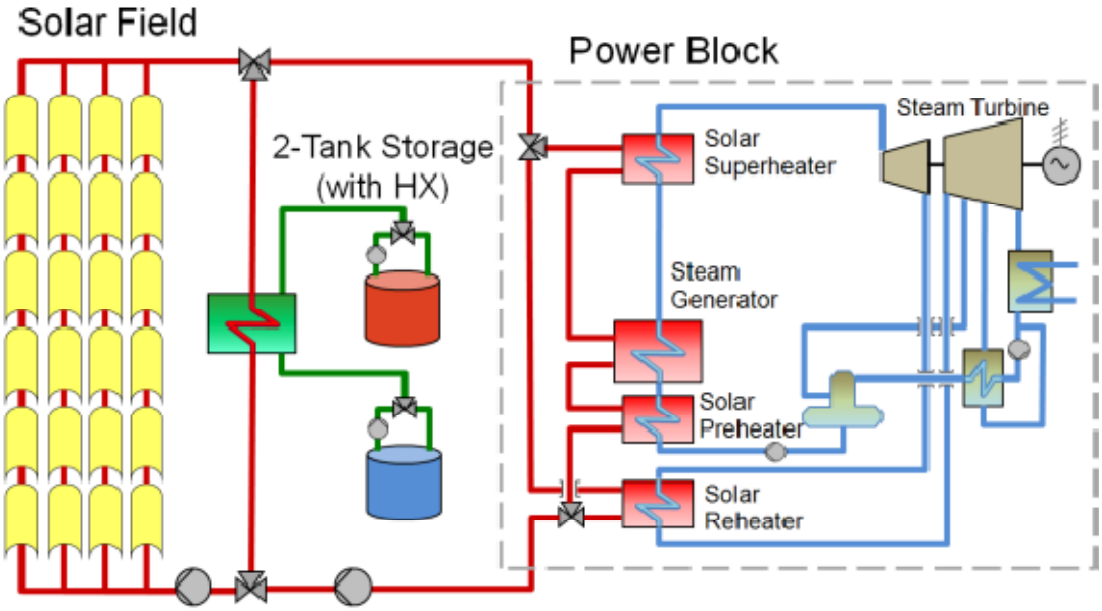


Figura 1.2: Esquema de una planta CSP con almacenamiento térmico

Tal como se muestra en la Figura 1.2, [3]; entre el campo solar y el ciclo de potencia se emplaza el sistema de almacenamiento térmico que suelen ser tanques de sales fundidas capaces de almacenar la energía sobrante del campo solar durante el día a través de un tren de intercambiadores. Durante periodos de nubes o en horas nocturnas, el sentido del intercambio de calor se revierte de manera que las sales fundidas con energía almacenada ceden su calor al circuito principal de fluido térmico para poder seguir operando con la turbina a plena carga. El sistema de almacenamiento térmico es esencialmente una batería con un tanque frío (tanque azul) y un tanque caliente (tanque rojo). Durante el día la particular batería se carga, habiendo flujo de sales desde el tanque frío al tanque caliente. Durante la noche o con nubes la batería se descarga, pasando las sales desde el tanque caliente (tanque rojo) al tanque frío (azul). Este sistema dota a las plantas CSP de gestionabilidad, capacidad de producción nocturna y cierta independencia ante cambios meteorológicos.

Se verifica en las figuras anteriores que una planta CSP típica con almacenamiento térmico tiene tres sistemas claramente diferenciados: sistema de fluido térmico (líneas rojas de la Figura 1.2), ciclo de potencia (líneas azules) y almacenamiento térmico (líneas verdes). La estructuración de la librería *ParaTrough* responde a esta diferenciación de macrosistemas básicos, correspondiendo una sub-librería a cada uno

de estos. De manera complementaria se ha establecido el recurso solar como otro macrosistema con sub-librería propia, debido a la relativa complejidad de la modelización de este recurso. En la versión inicial contemplada en este trabajo, esta librería cuenta con los sub-sistemas de recurso solar y fluido térmico.

1.2.Objetivos

El objetivo de la librería *ParaTrough* es que el ingeniero analista de procesos de una planta CSP durante su etapa de operación y mantenimiento pueda realizar con éxito y facilidad las siguientes labores:

- Evaluación del rendimiento: realizando la oportuna parametrización de la planta en particular, se puede comparar el rendimiento real con el rendimiento teórico que arroja el modelo de *ParaTrough* para identificar la necesidad de actuaciones al respecto.
- Detecciones de fallos: en aquellos días u operaciones en que se haya evaluado un rendimiento menor al teórico, se puede utilizar *ParaTrough* para comparar con los datos reales de la planta y así detectar fallos en equipos o procedimientos de operación. Esta comparación además sirve para afinar más el modelo teórico de la planta con la realidad particular, re-parametrizando *ParaTrough*.
- Exploración de nuevos modos de operación: para la mejora operacional de la planta se puede simular con *ParaTrough* nuevos escenarios y modos de operación de forma segura para equipos y personas antes de aplicar estos a la realidad.
- Optimización de la planta: el objetivo último de *ParaTrough* es ser una herramienta flexible para la optimización de la operación de una planta CSP tanto desde el punto de vista de producción como de beneficios económicos.

Para llevar a cabo la consecución de estos objetivos, se realizan las siguientes tareas-objetivo en la programación de *ParaTrough*:

Para evaluación de rendimiento y detección de fallos:

- Asegurar una conectividad a datos públicos de recurso solar de diferentes localizaciones para asegurar que la librería se pueda utilizar en cualquier lugar del mundo y utilizando formatos meteorológiocos estándar.
- Crear una base de datos típicos meteorológicos de diferentes localizaciones.
- Crear un modelo solar que calcule en cualquier momento y en cualquier localización terrestre la posición del sol.
- Crear un modelo detallado de lazo solar para modelar la conversión de energía solar en energía térmica.
- Crear un modelo simplificado de campo solar para simular plantas con numerosos lazos solares sin tener una carga computacional excesiva.

Para exploración de nuevos modos de operación y optimización de la planta

- Crear una librería de los medios materiales más usados en las plantas termosolares, permitiendo su cómoda reutilización y sustitución en los modelos de los diferentes componentes.
- Crear una librería de sistemas básicos de flujo de fluidos e intercambio térmico reutilizables en todos los macrosistemas de las plantas termosolares.
- Crear una base de datos de diferentes colectores solares y tubos absorbedores comerciales, para poder englobar las diferentes plantas existentes según la variabilidad de los fabricantes de los componentes.
- Crear modelos por instanciación gráfica que tengan en cuenta el sistema de control.

1.3.Estructura

El presente TFM comienza con una revisión bibliográfica del software existente de modelado y simulaicón termosolar, donde se argumenta la decisión de escoger el lenguaje *Modelica* para la librería (capítulo 2). Posteriormente se realiza la descripción de cada sub-librería de *ParaTrough* y sus modelos asociados.

La librería *ParaTrough* se estructura de la siguiente forma:

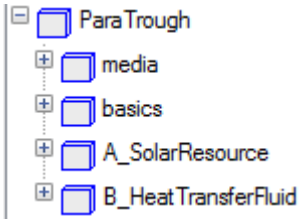


Figura 1.3: Estructura de ParaTrough

En primer lugar hay dos sub-librerías genéricas, *media* y *basics*. La primera contiene modelos y propiedades físicas de los medios materiales que se utilizan en los demás sistemas tales como agua, vapor, fluido térmico y sales fundidas (capítulo 3). La segunda contiene modelos físicos básicos de los diferentes dominios usados: dinámica de fluidos y transferencia de calor, junto con sus acoplamientos (capítulo 4). Estos modelos genéricos forman los elementos básicos para la construcción de los demás componentes más complejos.

Las siguientes sub-librerías corresponden a los macrosistemas básicos de una planta termosolar: recurso solar (*A_SolarResource*) en el capítulo 5 y sistema de fluido térmico (*B_HeatTransferFluid*) en el capítulo 6. Una vez descritas todas las sub-librerrías, se discuten las conclusiones, resultados y futuros trabajos de ampliación de *ParaTrough* en el capítulo 7.

Como anexo se incluye el código fuente de la librería en lenguaje *Modelica* y un CD-rom que acompaña a este TFM donde se incluye el presente documento en formato digital pdf, el archivo de extensión .mo de la librería *ParaTrough* ejecutable en *Dymola*

y las herramientas de conversión de datos meteorológicos *INTLConverter.xlsx* y *TMY2Converter.xlsx*.

2. Modelado y simulación de plantas termosolares

2.1. Introducción

En este capítulo se resume la revisión bibliográfica realizada para abordar el presente TFM, describiendo los fundamentos de las plantas termosolares y enumerando las herramientas de modelado y simulación para este tipo de instalaciones que actualmente están en el mercado. Se detallan los procedimientos habituales existentes para abordar este tipo de herramientas y finalmente se discuten los motivos que han llevado para optar por *Modelica* como lenguaje para soportar *ParaTrough*.

2.2. Herramientas software de modelado y simulación

La necesidad de evaluar el rendimiento de una planta CSP ha llevado a la introducción en el mercado de varias herramientas de modelado y simulación. A continuación se enumeran las principales:

2.2.1. System Advisor Model (SAM)

Es un software desarrollado por el laboratorio nacional de energías renovables de Estados Unidos (NREL), [4]. Es de acceso gratuito y presenta una amplia base de datos meteorológica en una gran variedad de localizaciones. Además del módulo de CSP cilindro parabólico posee otros módulos de otras tecnologías solares como fotovoltaica, térmica de torre, tecnología Fresnel [5] o discos Stirling [6]; y de otras energías renovables como la eólica, geotérmica e hidroeléctrica.

Está especialmente especificado para un estudio de viabilidad general de una planta, realizando predicciones de rendimiento y costes de energía asociados. Posee unos modelos financieros muy potentes englobando distintos escenarios (tarifa de venta de electricidad fija, precio negociado PPA y compra-venta desde el lado del cliente) diseñados para facilitar la toma de decisiones de directores de proyecto, analistas de políticas energéticas, desarroladores tecnológicos e investigadores. Sin embargo no posee la capacidad de un análisis detallado del rendimiento, limitándose éste a estudios anuales con poca resolución [7].

2.2.2. Transient Systems Simulation Program (TRNSYS)

TRNSYS [8] es un programa modular ampliamente usado en la simulación de sistemas solares y edificios [9], desarrollado por la Universidad de Wisconsin. Aunque especificado principalmente a sistemas solares pasivos como calentadores solares de agua en viviendas y de aire acondicionado, se han realizado intentos para adaptarlo a las centrales CSP tanto de tecnología cilindro-parabólica como de torre [10]. Tiene naturaleza modular, lo que le infiere de flexibilidad y potencial para incluir nuevos modelos. La librería estándadr de TRNSYS incluye muchos de los componentes usuales en los sistemas eléctricos y térmicos, así como rutinas programadas para añadir al programa datos meteorológicos o otras funciones temporalmente dependienets. No obstante el paradigma de modelos orientado a objetos está parcialmente desarrollado y muchas veces hay problemas de convergencia ante la incapacidad del programa de asignar la causalidad computacional.

2.2.3. Simulador termosolar Renovetec

Este simulador termosolar utiliza el programa SCADA de LabVIEW (de National Instrument) y está desarrollado principalmente para servir de plataforma de entrenamiento para operadores aunque también se puede verificar el estado de cada uno de los equipos principales que componen la planta comparando el resultado del simulador con los valores reales. Además de para las centrales termosolares de colector cilindro-parabólico, Renovetec [11] ha desarrollado hasta la fecha del presente trabajo (septiembre de 2015) cuatro simuladores más para las siguientes aplicaciones: plantas de biomasa, centrales de ciclo combinados, plantas de cogeneración y centrales termoeléctricas de carbón.

Aunque el simulador termosolare se ha comparado con una planta real situada en Ciudad Real y se consiguen pequeñas desviaciones, la poca flexibilidad al cambio de configuración para otros tipos de plantas y procedimientos de operación lo hace limitado para otros usos más complejos.

2.2.4. IPSEpro

Es un software modular desarrollado por SimTech, [12]. Pretende dar solución a las plantas a través de toda su vida útil, desde el diseño a la operación pasando por la verificación de las medidas reales en las pruebas de aceptación. Entre sus utilidades están las de calcular los balances energéticos y predecir el rendimiento preliminar por diseño, verificar y validar medidas instrumentales durante las pruebas de aceptación de las plantas, monitorizar y optmizar el rendimiento de la planta on-line y planear modificaciones y mejoras de las plantas existentes. Además de para las plantas termosolares de colector cilindro-parabólico, IPSEpro también ofrece soluciones de modelado y simulación para plantas de energía térmica fósil, energía geotérmica, desalinización, refrigeración, lavado de gases de emisión y plantas de gasificación de biomasa.

Se estructura en librerías y posee gran flexibilidad para desarrollar nuevos modelos mediante un paquete de desarrollo de nuevos modelos para el usuario. La librería termosolar tiene modelos bien desarrollados aunque no existe un pre-ensamblaje de estos en macrosistemas característicos de las plantas CSP, resultando tediosa y complicada su composición desde los elementos primarios. El problema de este software es también el alto coste de las licencias.

2.2.5. PCTrough

Este modelo en concreto fue desarrollado por Flabeg [13] y está especificado para que los clientes de las plantas CSP lo tomen como patrón contra el que baremar la operación anual de las empresas operadoras, que deben vencerle al cabo del año. El modelo es una caja negra en el que se calcula el resultado en energía eléctrica producida a partir de la introducción de los datos meteorológicos. El rango operacional en el que el modelo es preciso es limitado y la flexibilidad de modificación es nula debido a la protección a la que la empresa desarrolladora lo somete y a la intransigencia de clientes y operadores para tener un patrón fijo en el que comparar sus rendimientos anuales.

2.2.6. Matlab-Simulink

La famosa herramienta de Mathworks [14] para el modelado y simulación también ha sido utilizada para algunos estudios, desarrollando modelos para plantas CSP en régimen comercial [15] y plantas CSP de carácter innovador en centros de investigación [16]. Simulink es muy potente para modelar y simular con bloques por lo que es muy útil para los sistemas de control.

Sin embargo el paradigma de modelado no es ligado a objetos por lo que la flexibilidad para estudiar nuevos casos es baja, el sentido físico de las plantas se desfigura y la modificación de los modelos es tediosa e induce a grandes posibilidades de cometer error durante el modelado.

2.2.7. Modelica

Modelica es un lenguaje libre de modelado orientado a objetos especialmente concebido para el modelado de complejos sistemas multifísicos y cuya librería estándar y otras librerías accesibles contienen una gran cantidad de componentes mecánicos, eléctricos, electrónicos, térmicos, de control,... con código abierto y que presentan un gran potencial de modificación, reutilización y adaptabilidad.

El lenguaje puede ser soportado por múltiples entornos de modelado tanto libres como comerciales. Entre los libres se encuentran los siguientes:

- JModelica.org [19]
- Modelicac, Scicos [20]
- OpenModelica [21]

Dymola es un ambiente de simulación comercial desarrollado por Dassault Systèmes y posee un editor gráfico como principal característica, además de utilidades para la comunicación de los modelos con Simulink.

2.3.Elección de Modelica

La siguiente tabla resume y compara las herramientas presentadas:

Tabla 2.1: Comparativa entre diferentes herramientas de modelado y simulación existentes

Herramienta	Modular	Facilidad de utilización	Posible análisis de la operación diaria	Flexibilidad y adaptabilidad	Entornos de modelado gratuito
SAM	NO	SI	NO	NO	SI
TRNSYS	SI	NO	SI	SI	NO
Renovetec	NO	SI	SI	NO	NO
IPSEPro	SI	SI	SI	SI	NO
PCTrough	NO	SI	SI	NO	NO
Matlab-Simulik	NO	NO	SI	SI	NO
Modelica	SI	SI	SI	SI	SI

Se establecen los siguientes criterios de una buena herramienta de modelado y simulación:

- **Herramienta modular:** modulos con interfaces conexionables que aseguren que no se pierda el sentido físico del modelo, evitando un desarrollo completamente matemático mediante ecuaciones diferenciales.
- **Facilidad de utilización:** intuitividad y no necesidad de conocimientos avanzados en programación y computación.
- **Posibilidad de análisis de la operación diaria:** resolución alta para analizar periodos de tiempo relativamente cortos, en régimen diario.
- **Flexibilidad y adaptabilidad:** posibilidad de añadir nuevos modelos y de reutilización de los ya existentes en las librería estándar de las herramientas de modelado.
- **Entornos de modelado gratuito:** existencia de software libre gratuito.

Para conseguir los objetivos marcados en este TFM y que además se cumplan las especificaciones de calidad de la Tabla 2.1 se opta por utilizar el lenguaje *Modelica* [17] y el entorno de modelado *Dymola* [18]. A pesar de que este entorno es comercial, se utiliza al disponer de una licencia educativa para este trabajo. El código generado por *Dymola* y expuesto en el Anexo A: Código fuente en *Modelica*, es perfectamente utilizable en cualquiera de los entornos de modelado gratuito especificados en el apartado 2.2.7.

2.4. Trabajos similares en Modelica

Varios autores han sabido ya ver el enorme potencial de Modelica como lenguaje para el modelado y posterior simulación y optimización de plantas convencionales de producción energética, especialmente a problemas ligados al control de éstas [24].

La librería *Thermopower* [22] [23] para *Modelica* fue lanzada en 2002 por Francesco Casella y Alberto Leva. La última versión fue la 2.1, que vio la luz en 2009. Es una librería de código abierto para el modelado dinámico de plantas térmicas de gas desarrollada como herramienta para la investigación en el campo de los sistemas de

control por el departamento de electrónica, informática y bioingeniería del *Politecnico di Milano*. Posee grandes capacidades de flexibilidad y modularidad para acoplamientos entre el bloque de potencia de naturaleza térmica y los elementos y componentes propios de la ingeniería eléctrica (generador, transformadores, aparamente de mando y protección), así como facilidad de uso. Como principales limitaciones están la ausencia de garantía de soporte y de garantía de adecuación de la documentación.

Las características intrínsecas de no linealidad y condiciones cambiantes en los sistemas termosolares hacen que la gran flexibilidad, modularidad y existencia de entornos de modelado gratuitos de Modelica sean unas grandes ventajas a la hora de abordar modelos fiables, económicos y que aseguren su explotabilidad.

Los trabajos de Modelica para plantas termosolares presentan la problemática de los cambios bruscos en las condiciones atmosféricas y de las secuencias de operación.

Por ello los autores se centran en perfeccionar las dinámicas de los modelos, apoyándose en las favorables características de Modelica. Tal es el caso del modelo en *Modelica* de Robert Österholm en su artículo *Dynamic modelling of a parabolic trough solar power plant* [25] de 2014. Este contiene sub-modelos modulares de la entrada de radiación solar, del campo solar, almacenamiento térmico y un ciclo Rankine simplificado para el bloque de potencia. La validación del modelo con los datos de las plantas Andasol de Aldeire y La Calahorra mostraron buenos resultados para el campo solar. Sin embargo, la limitación queda patente en el resultado de generación eléctrica de la planta debido a la falta de algunas dinámicas presentes en las plantas reales como resultado de la utilización del ciclo Rankine simplificado.

La Plataforma Solar de Almería (PSA) [26], como referente mundial en la investigación de las tecnologías solares, es puntera en la utilización de Modelica para crear modelos fiables de tecnologías en desarrollo.

En el año 2005 el equipo cojunto de la PSA, la Universidad de Almería y la UNED de L.J Yebra, M. Berenguel, S. Dormido y M. Romero desarrollaron en su artículo *Modelling*

and Simualtion of Central Receiver Solar Thermal Power Plants [27] un modelo termohidráulico basado en la librería *ThermoFluid* de *Modelica* para el diseño del control avanzado y la optimización del rendimiento en plantas termosolares de torre, centrado en la planta piloto CESA-I de la PSA. La librería tiene como limitación la no utilización de valores reales o históricos de radiación solar ni el efecto óptico-geométrico de los heliostatos del campo solar sino que se introduce como señal de tipo escalón. Gracias a investigaciones como ésta se consiguió que hoy en día las plantas termosolares de torre sean una tecnología madura.

En el año 2006 el mismo equipo anterior junto con E.Zarza publicó en su artículo *Object Oriented Modelling of DISS Solar Thermal Power Plant* [28] un modelo en *Modelica* para la siumulación y control de la tecnología de generación de vapor directo mediante colectores cilindro parabólicos. Al igual que el artículo de 2005, en este también se reutiliza la librería *ThermoFluid* de *Modelica*. Los resultados se validaron con medidas experimentales la planta piloto de la PSA. El modelado del colector solar, la introducción de valores reales de radiación solar y de diferentes modos de operación son las principales cualidades presentadas en el trabajo. La librería está enfocada a una tecnología que actualmente aún no ha podido desarrollarse comercialmente debido a limitaciones técnicas de diversa índole.

Sin embargo, después de esas fechas (2001-2006) el grupo de investigación de la PSA dejó el estudio de modelos integrales de las plantas para continuar con los modelos de incipientes tecnologías como la foto-Fenton [29] o problemas de control específicos [30].

2.5.Conclusiones

El lenguaje *Modelica* y el ambiente de modelado *Dymola* son los sistemas escogidos para desarrollar la librería *ParaTrough* debido a que sus características son ideales para conseguir que *ParaTrough* cumpla con los objetivos marcados en este TFM (véase apartado 1.2) y con todas las especificaciones de calidad de la Tabla 2.1.

Partiendo de los trabajos de plantas térmicas convencionales del Politecnico de Milano y de los trabajos de investigación de la PSA se pretende ahondar en los modelos y en la y explotabilidad de estos para que sean una herramienta útil y accesible por las plantas termosolares que actualmente están en operación. Por ello *ParaTrough* será un vínculo entre lo conseguido por las entidades de investigación y lo conseguido por la experiencia de operación de las plantas existentes.

3. Librería de medios materiales (media)

3.1. Introducción

La librería de medios materiales (media) contiene los modelos de los fluidos que usualmente se utilizan en la industria termosolar.

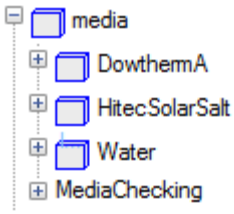


Figura 3.1: Estructura de la sub-librería media

Estos son el aceite térmico, del que se ha escogido la marca comercial Dowtherm A [31] por ser el más utilizado; la sal fundida solar, que por las mismas razones se ha escogido la marca comercial Hitec Solar Salt; y el agua.

Para realizar los modelos, se ha reutilizado el modelo *PartialMedium* de la librería estándar de Modelica (MSL) y se ha extendido redeclarando las funciones que establecen la dependencia de los parámetros físico-químicos con la temperatura.

Por último se programa un modelo para el chequeo de los medios materiales y analizar cómo dependen los parámetros físico-químicos de las variables de estado.

3.2. Aceite térmico (*DowthermA*)

El aceite térmico (HTF) usado en las plantas termosolares para recibir la energía concentrada del sol y cederla posteriormente al bloque de potencia es usualmente una mezcla eutéctica de difenilo y óxido de bifenilo. El rango de temperatura de este fluido es desde 12°C a 405°C. A 12°C presente problemas de congelación, por lo que el circuito del aceite térmico debe permanecer contantemente a temperaturas superiores a este valor [1], utilizando una caldera auxiliar de combustible fósil o traceado eléctrico.

A temperaturas superiores a 400ºC, empieza a degradarse en subproductos de naturaleza aromática con bajos puntos de ebullición como fenol y benceno, pudiendo ocasionar daños por cavitación en las bombas.

Se parte del modelo *PartialMedium* de la MSL. Se definen las funciones más importantes para el posterior modelado de los componentes de la planta: densidad, entalpía específica, viscosidad dinámica, conductividad térmica, capacidad calorífica específica, presión de vapor y calor latente de vaporización. Las funciones son sólo dependientes de la temperatura y esta dependencia viene dada por ecuaciones aprobadas por el Instituto de Diseño para Propiedades Físicas (DIPPR).

3.3.Sal solar (*HitecSolarSalt*)

La sal solar que más comúnmente se utiliza en la industria termosolar es la *Hitec Solar Salt*. Se compone de una mezcla eutéctica de sales fundidas de nitrato potásico y nitrato sódico. Es utilizada como fluido almacenador de energía debido a su alta capacidad calorífica y estabilidad a altas temperaturas. Aunque a empezando a utilizar directamente como aceite térmico, especialmente en las de tecnología de torre [32]. Las funciones que definen su comportamiento se refieren a las mismas propiedades que en el caso del *DowthermA* con la excepción de la presión de vapor y el calor latente de vaporización, debido a que el rango de operación de la sal solar está muy lejos de su punto de ebullición.

Su rango de operación va desde 238ºC a 593ºC. A 238ºC la sal congela, no siendo apta para el almacenamiento de energía o, en su caso, como fluido térmico. Por ello también se necesita que los sistemas que utilicen este medio estén permanentemente por encima de este valor de temperatura. Por encima de 593ºC la sal solar puede seguir operando. Sin embargo se ha escogido este punto como el máximo porque las fórmulas del DIPPR utilizadas para el modelado no son precisas por encima de este valor y porque esta temperatura ya se encuentra muy por encima de las temperaturas de operación de una planta termosolar, restringidas a 400ºC debido a la degradación del aceite térmico.

3.4.Agua

El agua es usada en una planta termosolar en el ciclo Rankine que mueve la turbina de vapor, el corazón de cualquier planta térmica, ya sea convencional o, como en este caso, termosolar. En este caso particular capta el tren de generación el calor portado por el aceite térmico como foco caliente del ciclo Rankine solar. También se utiliza como refrigeración externa para condensar el vapor en el condensador de la turbina, siendo esta agua de refrigeración el foco frío del ciclo.

Para modelar el comportamiento del agua tanto en estado líquido como en estado vapor se ha reutilizado el modelo *WaterIF97* explícito en presión y temperatura, que viene incluido en la MSL.

3.5.Ejemplos

Para chequear los modelos de las sustancias químicas o para analizar la dependencia de las propiedades físico-químicas con las variables de estado (presión y temperatura) se crea el modelo *MediaChecking*. En él se declara que el tiempo (*time*) es igual a la variable de estado de la que se quiere ver cómo dependen las otras propiedades y se deja fija la otra variable de estado como parámetro. Por lo tanto el modelo trabaja en dos modos seleccionables a partir de las variables booleanas *isobaric* o *isothermal*, para chequear a presión constante o a temperatura constante respectivamente. Se hace reemplazable el paquete (*package*) de los medios materiales para que el modelo *MediaChecking* pueda ser reutilizado para el análisis de todos los fluidos definidos en la sub-librería *media*.

En las siguientes figuras se muestran varias gráficas sobre la dependencia de las más destacadas propiedades físico-químicas con las variables de estado para las tres sustancias definidas.

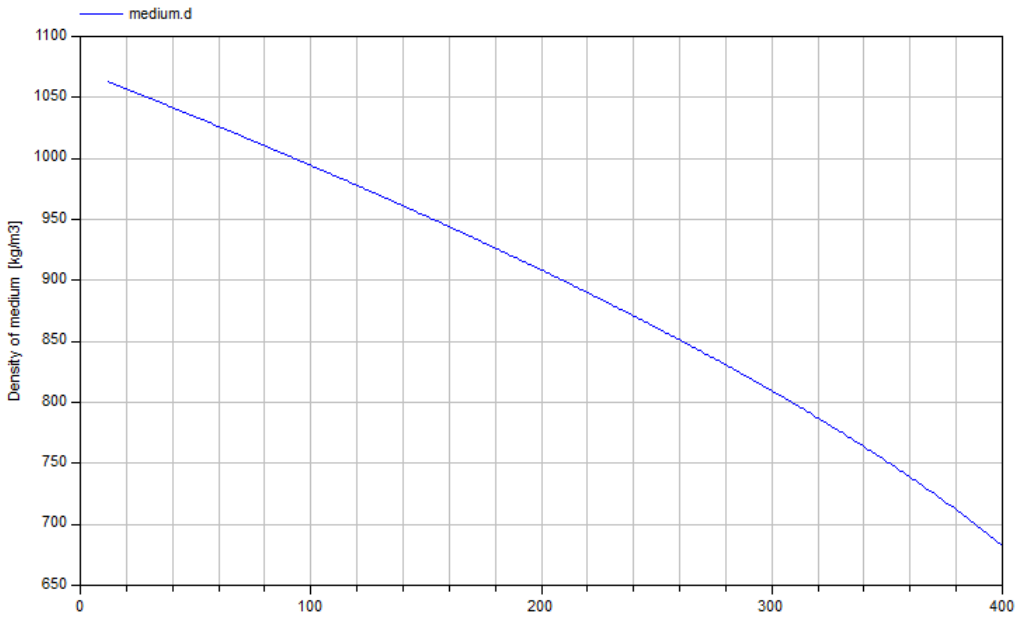


Figura 3.2: HTF. Dependencia de la densidad del HTF con la temperatura

Se observa en la Figura 3.2 la brusca dependencia de la densidad del HTF con la temperatura. Este cambio se manifiesta en el ciclo diario de una planta termosolar y tiene consecuencias operacionales, ya que por el día con el calentamiento solar la densidad baja considerablemente mientras que en la operación nocturna o con nubes, la densidad se incrementa.

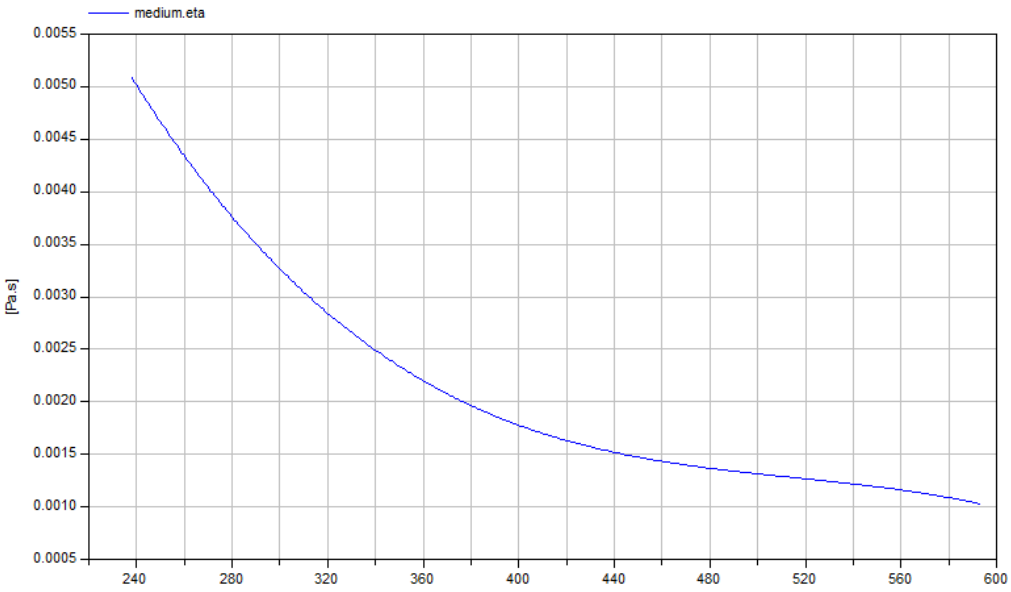


Figura 3.3: Dependencia de la viscosidad dinámica de las sales con la temperatura

La Figura 3.3 muestra cómo la viscosidad de las sales disminuye a altas temperaturas, siendo un rango óptimo a partir de 360°C y disminuye rápidamente cuando la temperatura cae por debajo de 320°C. Esto tiene consecuencias en el consumo de bombeo cuando el sistema se encuentra "frío".

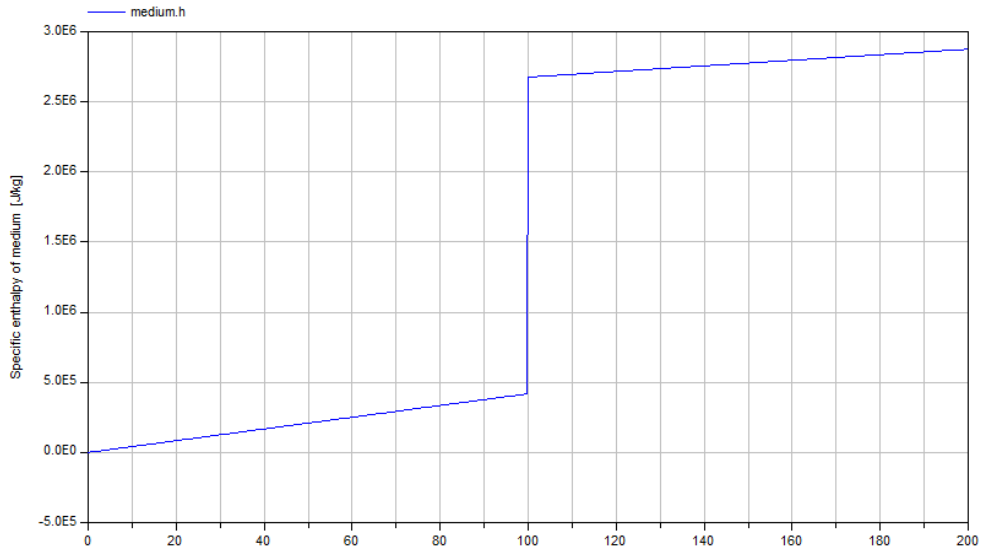


Figura 3.4: Dependencia de la entalpía específica del agua con la temperatura a presión atmosférica

En la Figura 3.4 se comprueba que el modelo del agua es válido para fase líquida como para fase vapor. El "salto" que se produce en la entalpía específica a los 100°C corresponde al calor latente de vaporización del agua.

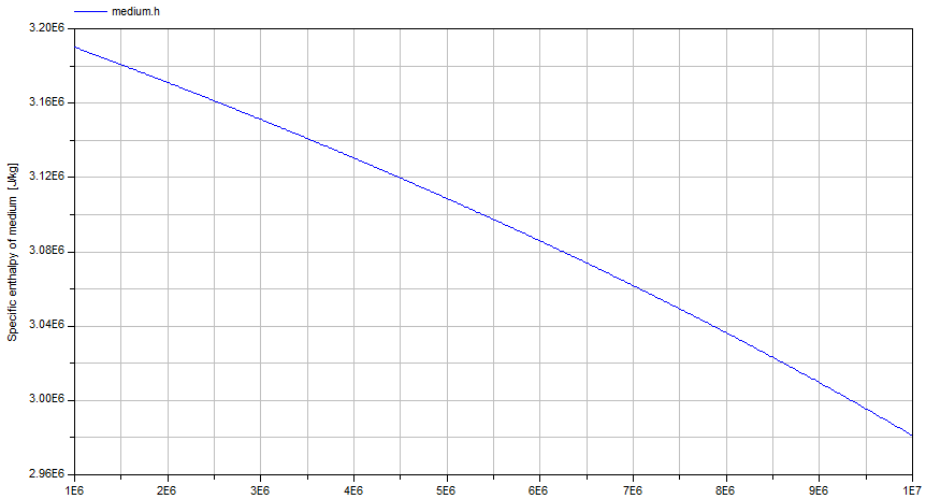


Figura 3.5: Dependencia de la entalpía del vapor frente a la presión a temperatura constante, 365°C

Se varía el modo del modelo a isotermo (*isothermal=true*). En la Figura 3.5 la presión es variable independiente del eje de abscisas, expresada en pascales. Se observa que la entalpía especifica a temperatura constante disminuye a medida que aumenta la presión desde la atmosférica (p=1E5 Pa) hasta la presión típica de una planta termosolar común de 50 MW (p=1E7, 100 bares).

3.6.Futuras ampliaciones

La sub-librería *media* es fácilmente ampliable mediante el uso de modelo MSL *PartialMedium* y editando las funciones de las propiedades físicas más importantes con respecto a la temperatura y la presión; o añadiendo directamente las múltiples sustancias que ya están definidas en MSL. Futuras ampliaciones de la librería serán la inclusión de otros medios materiales usados ya en las plantas termosolares como el nitrógeno, el gas natural, el diesel, productos químicos como ácido sulfúrico, hidróxido sódico, hipoclorito sódico, etc. Esto servirá para modelar y simular en detalle los sistemas auxiliares de la planta.

También hay campo de ampliación en el refinamiento del modelo de chequeo de sustancias *MediaChecking*. Se creará un nuevo modo para fluidos compresibles en el que mediante representación en 3D se podrá analizar la dependencia de las propiedades físicas de la presión y temperatura conjuntamente.

3.7.Conclusiones

En la sub-librería *media* contiene las sustancias que componen los sistemas principales de las plantas termosolares: el aceite térmico (*DowthermA*), la sal solar (*HitecSolarSalt*) y el agua (*Water*). En el caso de los dos primeros se ha adaptado el modelo de MSL *PartialMedium* y se han añadido las funciones que definen la dependencia de las propiedades físicas más importantes con la temperatura a partir de las ecuaciones publicadas por el DIPPR. El modelo del agua (*model*) se ha reutilizado del MSL, que utiliza las ecuaciones estándar IF97, que modelan conjuntamente el agua en estado líquido y vapor.

Finalmente se ha añadido a la sub-librería un modelo de chequeo de los medios materiales que permite analizar la dependencia de las propiedades físicas de las sustancias tanto en modo isobárico (se ve la dependencia con la temperatura, único modo con sentido para fluidos incompresibles como el *DowthermA* y la *HitecSolarSalt*) como en modo isotermo (se ve la dependencia con la presión, modo especificado para el análisis de fluidos compresibles como el vapor de agua).

4. Librería de modelos básicos (basics)

4.1. Introducción

La librería de modelos básicos (*basics*) contiene los bloques estructurales en los que se fundamentan las demás sub-librerías.

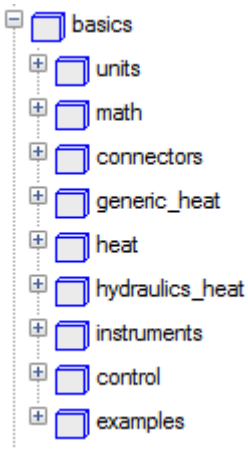


Figura 4.1: Estructura de la sub-librería *basics*.

Tal como se muestra en la Figura 4.1, la sub-librería se divide en el bloque de unidades (*units*), matemáticas (*math*), conectores (*connectors*), transmisión de calor genérica (*generic_heat*), transmisión de calor (*heat*), termohidráulica (*hydraulics_heat*), instrumentos (*instruments*), elementos de control (*control*) y ejemplos (*examples*).

Todos los modelos de esta sub-librería no son específicos para el modelado de plantas termosolares de colectores cilindro-parabólicos, sino que tienen un carácter genérico para poder ser aplicados en otras aplicaciones.

4.2. Unidades (*units*)

En este apartado se definen las unidades que no vienen por defecto en la sub-librería de MSL *Modelica.SIunits* y que se utilizan en los modelos de *ParaTrough*. Se definen los nuevos tipos (*type*) radiación acumulada (*AccRadiation*) cuyas unidades son MWh/m^2; energía en MWh, y flujo de calor por unidad de distancia, en W/m.

4.3.Matemáticas (*math*)

En este apartado se definen las operaciones matemáticas que no vienen por defecto en la sub-librería de MSL *Modelica.Math* y que serán utilizadas en las sub-librerías específicas de *ParaTrough*.

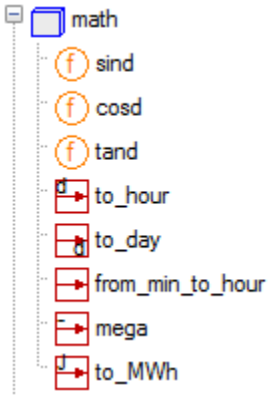


Figura 4.2: Nuevas definiciones de operaciones matemáticas

Tal como se muestra en la Figura 4.2, se definen las operaciones trigonométricas de seno, coseno y tangente como funciones del ángulo en grados en lugar de radianes. En estas funciones se incluye la operación de conversión de grados a radianes, ilustrada en la siguiente ecuación.

$$\alpha_{radianes} = \frac{2\pi}{360}\alpha_{grados} \qquad \text{Ec. (4.1)}$$

Dónde:

α_radianes: ángulo en radianes.

α_grados: ángulo en grados.

Además se definen conversiones de tiempo: de día a hora (*to_hour*), de hora a día (*to_day*), de minuto a hora (*from_min_to_hour*); la conversión al orden de magnitud "mega", que no es más que dividir la unidad base entre un millón; y la conversión de energía desde *joules* a MWh, dividiendo entre 3.6E9.

4.4.Conectores (*connectors*)

Los conectores son los elementos de interconexión entre diferentes modelos en el paradigma de modelado orientado a objetos [33] que utiliza Modelica. En estos conectores se establecen solo aquellas variables que interactúan con el exterior, quedando las demás en la estructura interna del modelo.

Existen dos tipos de variables que pueden ser definidas en los conectores: variables *through* y variables *across*. Las variables *trough* son las que tienen naturaleza de flujo o caudal, tal como puede ser el caudal volumétrico en el dominio hidráulico o la intensidad de corriente en el dominio eléctrico. Por el contrario, las variables *across* son las que tienen naturaleza de potencial como puede ser la temperatura en el dominio térmico o la velocidad en el dominio mecánico.

Tabla 4.1: Variables típicas en conectores en diferentes dominios de la ingeniería [34]

Dominio	Across	Through
Eléctrico	Voltaje (V)	Corriente (A)
Mecánica translación	Velocidad (m/s)	Fuerza (N)
Mecánica rotación	Veloc. angular (rad/s)	Torque (Nm)
Hidráulica	Presión (N/m^2)	Flujo volumen (m^3/s)
Térmico	Temperatura (K)	Flujo de entropía (W/K)
Químico	Potencial químico (J/mol)	Flujo molar (mol/s)

Las variables *across* y *through* típicas en los diferentes dominios de la ingeniería se ilustran en la Tabla 4.1. La diferencia entre ambos tipos de variables estriba en que en una unión entre conectores, la variable *across* se conserva y la variable *through* se hace de signo contrario para que la suma algebraica de esta variable en una unión de conectores siempre valga cero. Se establece en *ParaTrough* el criterio de signos de que cuando una variable *through* sale del modelo, ésta es negativa y cuando entra al modelo, ésta es positiva. El número de variables *through* y *across* por conector no están limitado para caso de que se quiera modelar acoplamientos entre varios dominios como puede ser el dominio termohidráulico (*through*: caudal, *across*: presión y temperatura) o electroquímico (*through:* flujo molar y corriente, *across*: potencial químico y voltaje).

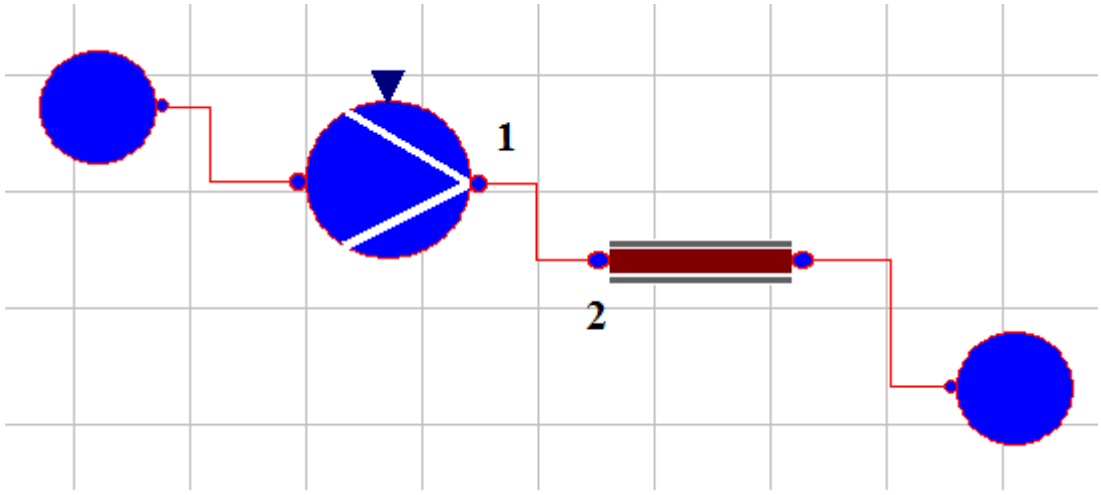


Figura 4.3: Unión entre dos conectores hidráulicos

La Figura 4.3 muestra una unión (línea roja) de dos modelos, una bomba (véase apartado 4.7.7) y un sumidero de presión (véase apartado 4.7.3) a través de sus conectores termohidráulicos, enumerados respectivamente como 1 y 2. Un conector termohidráulico se define en *ParaTrough* con el caudal másico como variable *through* y la temperatura y la presión como variables *across*. Tal como se ha explicado anteriormente, en esta unión se cumple lo siguiente:

$$t_2 = t_1$$

$$p_2 = p_1 \qquad \text{Ec. (4.2)}$$

$$m_2 = -m_1$$

Dónde:

t_i: temperatura en el conector i. Variable *across*.

p_i: presión en el conector i. Variable *across*.

m_i: caudal másico a través del conector i. Variable *through*.

Las uniones entre conectores en Modelica llevan empotradas las ecuaciones de conservación del flujo de potencia, por lo que se facilita la modelización. Es la generalización de las leyes sobre circuitos eléctricos de Kirchhoff [35] para ser utilizada

al resto de dominios de la ingeniería. La ley de corrientes es de aplicación para las variables *through* mientras que la ley de tensiones lo es para las variables *across*.

Los conectores definidos en *ParaTrough* se enumeran en la siguiente tabla:

Tabla 4.2: Conectores definidos en *ParaTrough*

Conector	Variables *through*	Variables *across*	Símbolo gráfico
Térmico (*heatPort*)	Flujo de calor (*Q*)	Temperatura (*t*)	
Termohidráulico (*hfPort*)	Caudal másico (*m*)	Presión (*p*) Temperatura (*t*)	
Conector meteorológico (*weather_connector*)		Radiación (*DNI*) Temperatura (*t*) Velocidad de viento (*ws*) Ángulos de posición solar	
Entrada analógica		Señal de entrada (*u*)	
Salida analógica		Señal de salida (*y*)	

Los conectores de la Tabla 4.2 serán utilizados para unir a través de conexiones los modelos atómicos creados en *ParaTrough* para formar modelos de sistemas complejos.

4.5.Transmisión de calor genérica (*generic_heat*)

En esta sub-librería se encuentran modelos de los diferentes mecanismos de transmisión de calor (conducción, convección y radiación) así como modelos del ambiente. Los mecanismos de transmisión de calor son genéricos, de manera que no dependen de la geometría del sistema.

4.5.1. Calor por conducción (*conduction*)

En este modelo se establece el fenómeno de conducción térmica unidimensional a través de un medio material sólido cuyos extremos se encuentran a temperaturas diferentes.

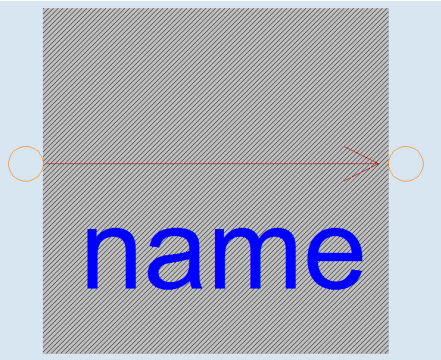


Figura 4.4: Icono del modelo *conduction*

$$Q = K \cdot \Delta T \qquad \text{Ec. (4.3)}$$

Dónde:

Q: flujo de calor a través del medio material, en W.

ΔT: incremento de temperatura entre los extremos del medio material, en K.

K: resistencia equivalente a la conducción de calor del medio material, en $W \cdot K^{-1}$.

4.5.2. Calor por convección (*convection*)

En este modelo se establece el fenómeno de convección térmica unidimensional a través de un medio material fluido en movimiento donde hay un gradiente de temperatura entre dos puntos.

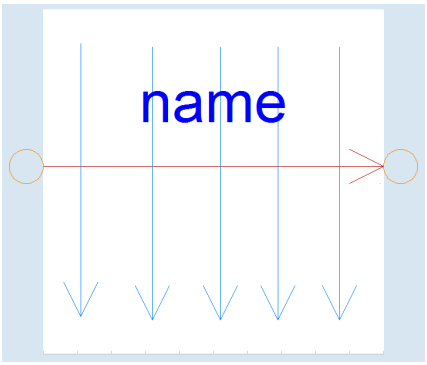


Figura 4.1: Icono del modelo *convection*

$$Q = H \cdot \Delta T \qquad \text{Ec. (4.4)}$$

Dónde:

Q: flujo de calor a través del medio fluido, en W.

ΔT: incremento de temperatura entre dos puntos del medio material, en K.

H: resistencia equivalente a la convección de calor del medio fluido, en $W \cdot K^{-1}$.

4.5.3. Calor por radiación (*radiation*)

En este modelo se establece el fenómeno de radiación térmica unidimensional a través entre dos cuerpos radiantes que se encuentran a diferente temperatura o entre un cuerpo radiante y el ambiente.

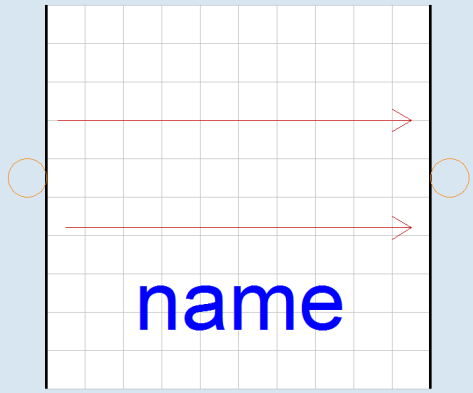


Figura 4.2: Icono del modelo *radiation*

$$Q = \sigma \cdot E \cdot ({T_1}^4 - {T_2}^4) \qquad \text{Ec. (4.5)}$$

Dónde:

Q: flujo de calor por radiación, en W.

σ: constante de Stefan-Boltzmann. 5.67E-8 $W \cdot m^{-2} \cdot K^{-4}$.

E: resistencia equivalente a la radiación, en m^2.

T_i: temperatura del cuerpo o ambiente i, en K.

4.5.4. Ambiente simplificado (*Ambient_simple*)

Este modelo actúa de sumidero térmico para los mecanismos de convección y de radiación, simulando el ambiente a una temperatura ambiente y velocidad de viento constante.

Figura 4.3: Icono del modelo *Ambient_simple*

Se hace una primera estimación de la temperatura del cielo (*T_sky*) como 8 grados inferior a la temperatura ambiente. Esta temperatura es la variable T_2 de la Ec. (4.5) cuando hay radiación térmica entre un cuerpo y el ambiente.

4.5.5. Ambiente (*Ambient*)

Este modelo tiene la misma función que *Ambient_simple* pero permite la entrada de datos meteorológicos variables de temperatura ambiente y velocidad de viento a través de un conector *weather_connector*. Tiene además dos conectores térmicos *heatPort*, uno para simular el ambiente la conexión con el ambiente (temperatura ambiente y velocidad de viento) cuando hay mecanismo de convección térmica y otro para simular el cielo (temperatura del cielo) cuando hay mecanismo de radiación.

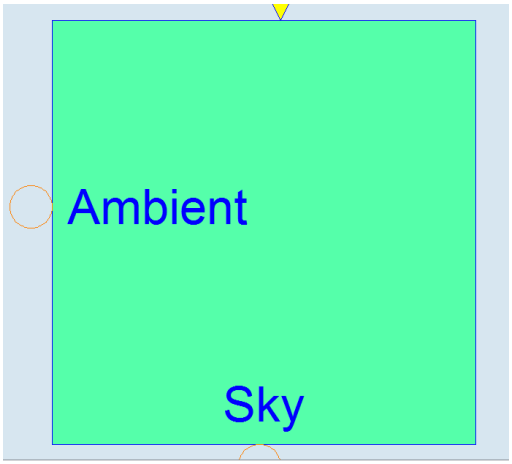


Figura 4.4: Icono del modelo *Ambient*

4.5.6. Eficiencia radiante (*IrradiationEfficiency*)

Este modelo simula una pérdida de eficiencia en la transmisión de la radiación solar, que puede ser causada por errores geométricos o de enfoque de las superficies a las que se radian.

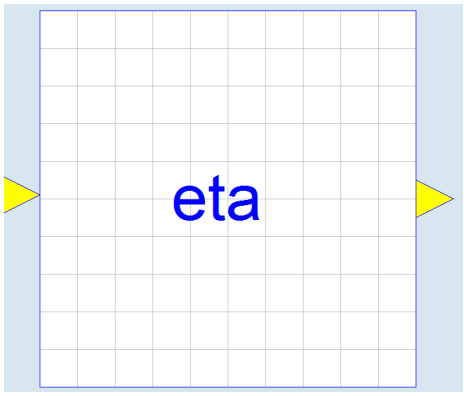


Figura 4.5: Icono del modelo *IrradiationEfficiency*

Su ecuación constitutiva es la siguiente:

$$DNI_2 = \eta \cdot DNI_1 \qquad \text{Ec. (4.6)}$$

Dónde:

DNI_1: radiación inicial, en $W \cdot m^{-2}$.

DNI_2: radiación final, en $W \cdot m^{-2}$.

η: factor de eficiencia.

Las demás variables del conector *weather_connector* (temperatura, velocidad de viento y ángulos de posición solar no cambian entre el punto inicial y el punto final.

4.6.Transmisión de calor (heat)

En esta sub-librería se particularizan los modelos de transmisión de calor (conducción, convección y radiación) para unas geometrías específicas. Las geometrías necesarias en *ParaTrough* son la corona anular y el cilindro.

4.6.1. Conducción a través de una corona anular (*k_annulus*)

Este modelo es una extensión de *conduction* (véase apartado 4.5.1) particularizando la definición de la resistencia equivalente a la conducción (*K*) a la geometría de corona anular:

$$K = \frac{2\pi k}{\ln\left(\frac{D_o}{D_i}\right)} L \qquad \text{Ec. (4.7)}$$

Dónde

K: resistencia equivalente a la conducción de calor del medio material, en $W \cdot K^{-1}$.
k: conductividad térmica del material, en $W \cdot m^{-1} \cdot K^{-1}$.
Do: diámetro exterior de la corona anular, en m.
Di: diámetro interior de la corona anular, en m.
L: longitud de la corona anular, en m.

Todas las variables de la Ec. (4.7) son definidas como parámetros a excepción de la conductividad térmica del material (*k*), que en la mayoría de los casos es una función de la temperatura intrínseca a cada material.

4.6.2. Convección en un cilindro (*h_cylinder*)

Este modelo es una extensión de *convection* (véase apartado 4.5.2) particularizando la definición de la resistencia equivalente a la convección (*H*) a la geometría del cilindro:

$$H = \pi h D L \qquad \text{Ec. (4.8)}$$

Dónde

H: resistencia equivalente a la convección de calor del medio fluido, en $W \cdot K^{-1}$.

h: coeficiente de convección térmica del medio fluido, en $W \cdot m^{-2} \cdot K^{-1}$.

D: diámetro del cilindro, en m.

L: longitud del cilindro, en m.

Todas las variables de la Ec. (4.8) son definidas como parámetros a excepción del coeficiente de convección del medio fluido (*h*), que en la mayoría de los casos es una función del caudal o velocidad del medio.

4.6.3. Radiación a través de una corona anular (*r_annulus*)

Este modelo es una extensión de *radiation* (véase apartado 4.5.3) particularizando la definición de la resistencia equivalente a la radiación (*E*) a la geometría de la corona anular:

$$E = \frac{\pi D_i}{\frac{1}{\varepsilon_i} + \frac{1 - \varepsilon_o}{\varepsilon_o}\frac{D_i}{D_o}} L \qquad \text{Ec. (4.9)}$$

Dónde

E: resistencia equivalente a la radiación, en m^2.

ε_i: emisividad del cuerpo interno de la corona anular, adimensional.

ε_o: emisividad del cuerpo externo de la corona anular, adimensional.

D_i: diámetro interno de la corona anular, en m.

D_o: diámetro externo de la corona anular, en m.

L: longitud de la corona anular, en m.

Todas las variables de la Ec. (4.9) son definidas como parámetros a excepción de la emisividad interna (ε_i) y la emisividad externa (ε_o) que en la mayoría de los casos es una función de la temperatura de los cuerpos emisores.

4.6.4. Radiación en un cilindro (*r_cylinder*)

Este modelo también es una extensión de *radiation* (véase apartado 4.5.3) particularizando la definición de la resistencia equivalente a la radiación (*E*) a la geometría del cilindro:

$$E = \pi \varepsilon D L \qquad \text{Ec. (4.10)}$$

Dónde

E: resistencia equivalente a la radiación, en m^2.

ε: emisividad del cuerpo del cilindro, adimensional.

D: diámetro del cilindro, en m.

L: longitud del cilindro, en m.

Todas las variables de la Ec. (4.10) son definidas como parámetros a excepción de la emisividad (ε) que en la mayoría de los casos es una función de la temperatura del cuerpo emisor.

4.7.Termohidráulica (*hydraulics_heat*)

Esta sub-librería consta de modelos del dominio termohidráulico.

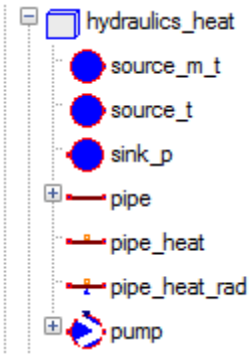


Figura 4.6: Modelos de la sub-librería *hydraulics_heat*

A continuación se detalla cada uno de los modelos que aparecen en la Figura 4.6.

4.7.1. Fuente de caudal y temperatura (*source_m_t*)

Este modelo simple solamente suministra un caudal másico (*m*) y una temperatura (*t*) determinados como parámetros a través de su único conector de tipo *hfPort* (véase apartado 4.4).

4.7.2. Fuente de temperatura (*source_t*)

Este modelo es similar al anterior pero suministrando solamente temperatura (*t*), que también es definida como parámetro ajustable.

4.7.3. Sumidero de presión (*sink_p*)

Este modelo suministra una presión al sistema (*p*), definida como parámetro ajustable.

4.7.4. Tubería genérica (*pipe*)

Este modelo es más complejo que los anteriores y representa una tubería con una sustancia específica fluyendo en su interior.

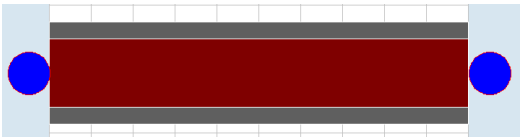


Figura 4.7: Icono del modelo *pipe*

Tal como se muestra en la Figura 4.7, su interfaz está formada por dos conectores de tipo *hfPort*. Además se define el material que fluye en su interior llamando a una de las sustancias definidas en la sub-librería *media* (véase apartado 3), haciendo que esta sustancia sea reemplazable en el modelo para poder así simular la tubería con diferentes medios materiales utilizando el mismo componente.

Las siguientes ecuaciones son las que constituyen el modelo:

- ✓ Balance de materia

$$m = hfPort1.m \qquad \text{Ec. (4.11)}$$

$$m = -hfPort2.m$$

Dónde:

m: caudal másico a través de la tubería, en $kg \cdot s^{-1}$.
hfPort1.m: caudal másico entrando o saliendo del conector 1, en $kg \cdot s^{-1}$.
hfPort2.m: caudal másico entrando o saliendo del conector 2, en $kg \cdot s^{-1}$.

La Ec. (4.11) define el balance de materia en el interior de la tubería, en el que no hay ni acumulación ni generación por lo que lo que entra por un extremo es igual a lo que sale por el otro.

- ✓ Pérdida de presión por fricción

Para la determinación de la pérdida de presión por fricción, se utiliza el coeficiente de fricción de Darcy-Weisbach.

$$\Delta P = \frac{1}{2}\frac{f L v^2}{\rho D}$$ Ec. (4.12)

Dónde:

ΔP: diferencia de presión entre el conector *hfPort1* y el *hfPort2*, en Pa.

f: factor de fricción de Darcy-Weisbach [36], adimensional.

v: velocidad lineal del fluido a través de la tubería, en $m \cdot s^{-1}$.

ρ: densidad del fluido, en $kg \cdot m^{-3}$. Se evalúa a través del medio material de la sub-librería *medium*.

D: diámetro interno de la tubería, en m.

La velocidad lineal del fluido se determina a través de la ecuación de continuidad:

$$m = vA\rho$$ Ec. (4.13)

Dónde:

m: caudal másico a través de la tubería, en $kg \cdot s^{-1}$.

v: velocidad lineal del fluido a través de la tubería, en $m \cdot s^{-1}$.

ρ: densidad del fluido, en $kg \cdot m^{-3}$. Se evalúa a través del medio material de la sub-librería *medium*.

A: sección transversal de la tubería, en m^{-2}.

La sección transversal de la tubería es la correspondiente a una geometría circular:

$$A = \frac{\pi}{4}D^2$$ Ec. (4.14)

Dónde:

A: sección transversal de la tubería, en m^2.

D: diámetro interno de la tubería, en m.

El coeficiente de fricción depende del número adimensional de Reynolds. Éste se define como:

$$Re = \frac{vD\rho}{\eta} \qquad \text{Ec. (4.15)}$$

Dónde:

Re: número de Reynolds [37], adimensional.
v: velocidad lineal del fluido a través de la tubería, en $m \cdot s^{-1}$.
D: diámetro interno de la tubería, en m.
ρ: densidad del fluido, en $kg \cdot m^{-3}$. Se evalúa a través del medio material de la sub-librería *medium*.
η: viscosidad dinámica del fluido, en Pa·s. Se evalúa a través del medio material de la sub-librería *médium*.

Finalmente, el factor de fricción se evalúa mediante la siguiente conjunto de ecuaciones:

Si el régimen de flujo es laminar ($Re \leq 2300$):

$$f = \frac{64}{Re} \qquad \text{Ec. (4.16)}$$

Dónde:

f: factor de fricción de Darcy-Weisbach [36], adimensional.
Re: número de Reynolds [37], adimensional.

Si el régimen de flujo es de transición o turbulento (Re > 2300), se utiliza la ecuación de Colebrook-White [38]:

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\frac{k}{D}}{3.7} + \frac{2.51}{Re\sqrt{f}}\right) \qquad \text{Ec. (4.17)}$$

Dónde:

f: factor de fricción de Darcy-Weisbach [36], adimensional.

Re: número de Reynolds [37], adimensional.

D: diámetro interno de la tubería, en m.

k: rugosidad de la pared de la tubería, en m. Es característica del material de construcción de la misma.

El modelo *pipe* está constituido por tanto por las ecuaciones desde la Ec. (4.11) hasta la Ec. (4.17). Los parámetros son la longitud (*L*), el diámetro (*D*) y la rugosidad (*k*) de la tubería. Las propiedades físicas de la densidad (ρ) y viscosidad dinámica (η) se calculan con las funciones específicas de la sustancia que fluye a través de la tubería, empotradas en la sub-librería reemplazable de *media* (véase apartado 3).

4.7.5. Tubería con intercambio de calor (*pipe_heat*)

Este modelo es una extensión de *pipe* por lo que hereda todas las ecuaciones descritas en el apartado 4.7.4. Se añade un conector térmico (*heatPort*) para tener en cuenta una transmisión de calor a través de las paredes de la tubería cilíndrica.

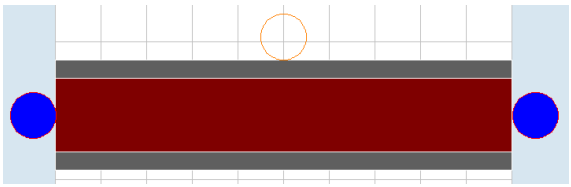


Figura 4.8: Icono del modelo *pipe_heat*

Se añade la ecuación del balance energético:

$$Q = H_2 - H_1 \qquad \text{Ec. (4.18)}$$

Dónde:

Q: flujo de calor a o desde la tubería, en W.

H_i: flujo de entalpía que entra o sale del conector i, en W.

Los flujos de entalpía que aparecen en la Ec. (4.18) se evalúan con la función de entalpía específica dependiendo de la temperatura para el medio material concreto que fluye a través de la tubería:

$$H_i = m \cdot h_i \qquad \text{Ec. (4.19)}$$

Dónde

H_i: flujo de entalpía que entra o sale del conector i, en W.

m: caudal másico a través de la tubería, en $kg \cdot s^{-1}$.

h_i: entalpía específica del medio material a la temperatura del conector i, en W/kg.

4.7.6. Tubería irradiada con intercambio de calor (*pipe_heat_rad*)

Este modelo es una extensión del anterior (*pipe_heat*) pero que incluye un conector del tipo *weather_connector*.

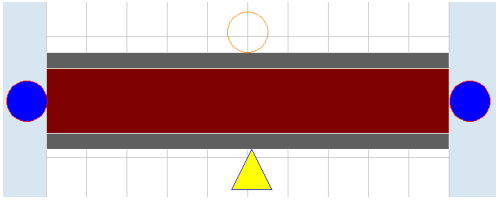


Figura 4.9: icono del modelo *pipe_heat_rad*

Se redefine el flujo de calor (*Q*) de la Ec. (4.18) como:

$$Q = weather\_connector.DNI + heatPort.Q \qquad \text{Ec. (4.20)}$$

Dónde:

Q: flujo neto de calor a o desde la tubería, en W.

weather_connector.DNI: radiación que incide sobre la tubería, en W.

heaPort.Q: flujo de calor a o desde la tubería, en W.

Este modelo es la base para la creación del modelo *HCE* (véase apartado 6.4.2).

4.7.7. Bomba (*pump*)

Este modelo simula una bomba que dota al fluido de un aumento de presión a un caudal másico determinado.

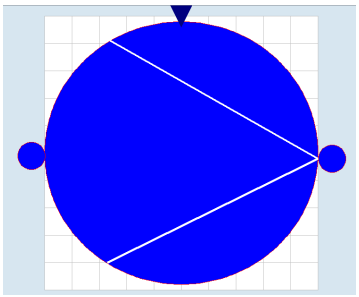


Figura 4.10: Icono del modelo *pump*

Una bomba centrífuga responde a su curva característica, que relaciona la altura manométrica vencida frente al caudal [39]:

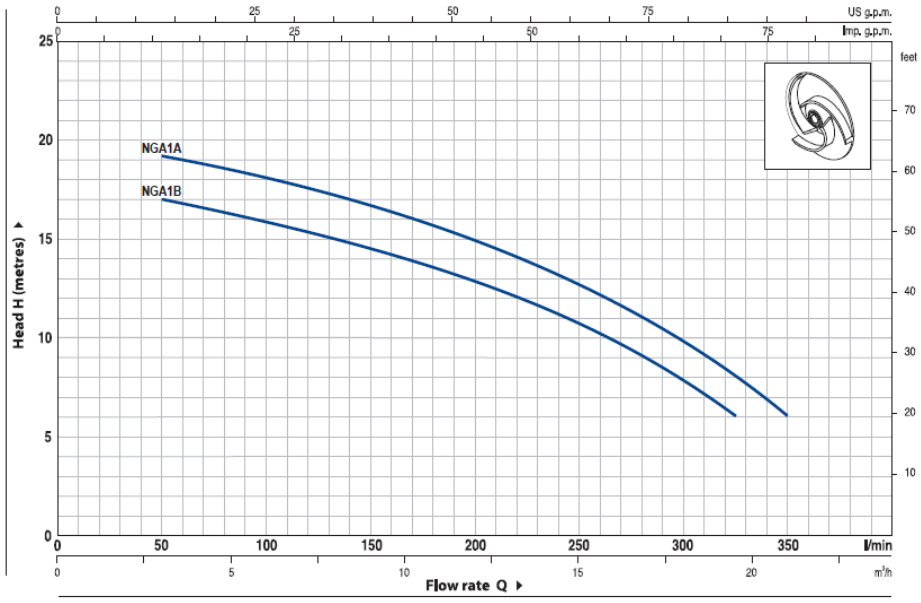


Figura 4.11: Curvas características de dos bombas del mismo fabricante

En el modelo de *ParaTrough* para *pump* se establecen cuatro parámetros (dos de altura manométrica y dos de caudal volumétrico) para establecer dos puntos conocidos de la curva característica de la bomba. A partir de estos dos, se aproxima la curva característica a una línea recta. Esta suposición es tanto más cierta cuando el intervalo de operación de la bomba es estrecho pero cuando este intervalo es más ancho, hay desviaciones sustanciales.

$$\Delta H = \Delta H_0 + \frac{\Delta H_0 - \Delta H_1}{q_o - q_1} q \qquad \text{Ec. (4.21)}$$

Dónde:

ΔH: altura manométrica que vence la bomba, en m.

ΔH_0: altura manométrica que vence la bomba en el punto 0 de la curva característica, en m.

ΔH_1: altura manométrica que vence la bomba en el punto 1 de la curva característica, en m.

q: caudal volumétrico que suministra la bomba, en m^3/s.

q_0: caudal volumétrico que suministra la bomba en el punto 0 de la curva característica, en m^3/s.

q_1: caudal volumétrico que suministra la bomba en el punto 0 de la curva característica, en m^3/s.

El punto 0 y el punto 1 a los que se refiere la Ec. (4.21) son dos puntos arbitrarios de la curva característica de la bomba (que tendrá un aspecto similar a la de la Figura 4.11) y tendrán que ser elegidos como parámetros adecuadamente para que la hipótesis de linealización de la curva sea lo más acertada posible.

4.8. Instrumentos (*instruments*)

En esta sub-librería se modelan los instrumentos. Estos componentes solamente convierten una medida física (de un conector tipo *hfPort*) en una señal analógica de salida (de un conector tipo *analog_output*) para posteriormente ser utilizada en los elementos de control del apartado 4.9.

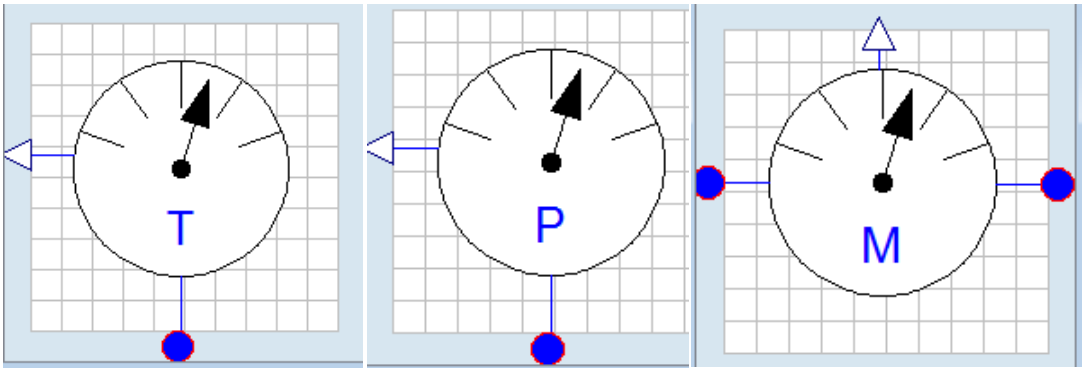


Figura 4.12: Iconos de los modelos *temperature_sensor*, *pressure_sensor*, *massflow_sensor*

Los modelos ilustrados en la Figura 4.12 constituyen los elementos primarios de medida en un lazo de control.

4.9.Elementos de control (*control*)

Aquí se establecen los modelos que actúan de parte central en un lazo de control, dotándolo de la ley de control adecuada para el correcto funcionamiento. Se ha reutilizado el modelo de MSL *Blocks.Continuos.LimPID*.

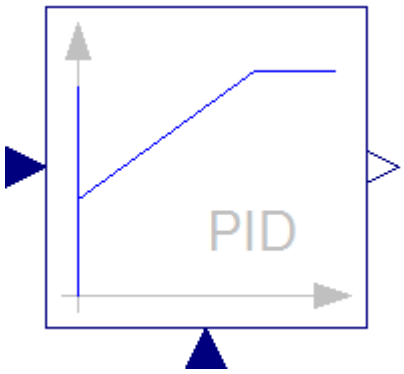


Figura 4.13 Icono de los modelos *LimPID_direct* y *LimPID_indirect*

Se ha duplicado el modelo para diferenciar entre una acción de control directa (*LimPID_direct*) y una acción de control indirecta (*LimPID_indirect*). En la primera, al producirse un aumento en la señal del instrumento, se produce un aumento en el mismo sentido de la acción de control. En la segunda, al producirse un cambio en la señal del instrumento (por ejemplo, ascendente), se produce un cambio en el sentido contrario de la acción de control (en el caso del ejemplo, descendente).

4.10. Futuras ampliaciones

La sub-librería *basics* se ampliará añadiendo los modelos genéricos necesarios para las otros macrosistemas de la planta, una vez que estos se vayan añadiendo. Así, por ejemplo, se programarán intercambiadores de calor, tanques, válvulas, bifurcaciones en "T",...

4.11. Conclusiones

Los modelos de la sub-librería *basics* son reutilizados en las sub-librerías de los macrosistemas de las plantas termosolares. Los modelos son genéricos no sólo para estos sistemas solares sino también para otras librerías de *Modelica* donde se puedan reutilizar estos componentes básicos.

5. Librería de recurso solar (A_SolarResource)

5.1. Introducción

La librería de recurso solar (A_SolarResource) contiene tablas de datos, funciones astronómicas, datos meteorológicos de diferentes localizaciones de plantas termosolares, herramientas para su interpolación dentro de los modelos y el modelo solar (*Sun*).

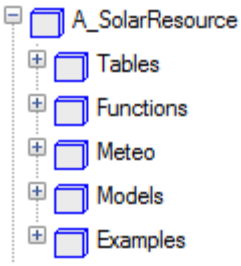


Figura 5.1: Estructura de la sub-librería *A_SolarResource*

Por último está la sub-librería de Ejemplos (*Examples*) donde se localizan dos modelos para comparar los datos meteorológicos entre diferentes días y diferentes localizaciones, respectivamente.

5.2. Tablas (*Tables*)

Aquí se introducen varias tablas que serán útiles para la definición de modelo solar (*Sun)* en el apartado 5.5.2.

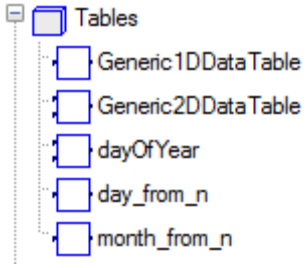


Figura 5.2: Tablas de datos útiles para el modelo solar

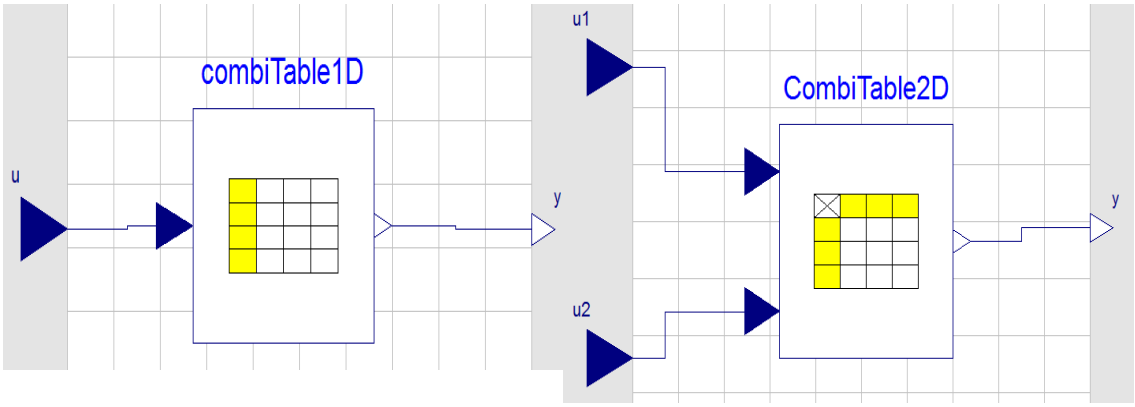


Figura 5.3: Modelos MSL *CombiTable1D* y *CombiTable2D*

Los modelos básicos de tablas son *Generic1DDataTable* y *Generic2DDataTable*. Estos utilizan los modelos de la MSL *CombiTable1D* y *CombiTable2D* respectivamente *(*emplazados en *Modelica.Blocks.Tables)*.

Se trata de unos bloques que contienen incrustada una tabla bidimensional. La entrada u1 indica la búsqueda por fila y la entrada u2 indica la búsqueda por columna (en el caso de la tabla unidimensional, sólo hay una columna de búsqueda). La salida y se obtiene de cruzar la fila y la columna buscadas.

La tabla *dayOfYear* es una extensión de *Generic2DDataTable* y relaciona el día del año (de 1 a 365) con el día (en filas) y mes (en columnas).

Tabla 5.1: Conversión desde día y mes a día del año. Tabla incrustada en *DayOfYear*

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	32	60	91	121	152	182	213	244	274	305	335
2	2	33	61	92	122	153	183	214	245	275	306	336
3	3	34	62	93	123	154	184	215	246	276	307	337
4	4	35	63	94	124	155	185	216	247	277	308	338
5	5	36	64	95	125	156	186	217	248	278	309	339
6	6	37	65	96	126	157	187	218	249	279	310	340
7	7	38	66	97	127	158	188	219	250	280	311	341
8	8	39	67	98	128	159	189	220	251	281	312	342
9	9	40	68	99	129	160	190	221	252	282	313	343
10	10	41	69	100	130	161	191	222	253	283	314	344
11	11	42	70	101	131	162	192	223	254	284	315	345
12	12	43	71	102	132	163	193	224	255	285	316	346
13	13	44	72	103	133	164	194	225	256	286	317	347
14	14	45	73	104	134	165	195	226	257	287	318	348
15	15	46	74	105	135	166	196	227	258	288	319	349
16	16	47	75	106	136	167	197	228	259	289	320	350
17	17	48	76	107	137	168	198	229	260	290	321	351
18	18	49	77	108	138	169	199	230	261	291	322	352
19	19	50	78	109	139	170	200	231	262	292	323	353
20	20	51	79	110	140	171	201	232	263	293	324	354
21	21	52	80	111	141	172	202	233	264	294	325	355
22	22	53	81	112	142	173	203	234	265	295	326	356
23	23	54	82	113	143	174	204	235	266	296	327	357
24	24	55	83	114	144	175	205	236	267	297	328	358
25	25	56	84	115	145	176	206	237	268	298	329	359
26	26	57	85	116	146	177	207	238	269	299	330	360
27	27	58	86	117	147	178	208	239	270	300	331	361
28	28	59	87	118	148	179	209	240	271	301	332	362
29	29	0	88	119	149	180	210	241	272	302	333	363
30	30	0	89	120	150	181	211	242	273	303	334	364
31	31	0	90	0	151	0	212	243	0	304	0	365

A partir de la Tabla 5.1 se correlaciona que el 1 de enero (fila=1, columna=1) corresponde al día del año 1 y que por ejemplo el 23 de marzo (fila=23, columna=3) corresponde al día del año 82.

Al el modelo se le añade la siguiente ecuación para simular el paso de los días con respecto al tiempo de simulación (*time*).

$$n = n_0 + \frac{time}{24} \qquad \text{Ec. (5.1)}$$

Dónde:

n= día del año (de 1 a 365).
n_0=día del año inicial cuando empieza la simulación.
Time=tiempo de simulación, en horas.

Las tablas *day_from_n* y *month_from_n* realizan la operación opuesta a la tabla *dayOfYear*. La primera calcula el día (*day*) a partir del día del año (*n*) y la segunda calcula el mes a partir del día del año (*n*). Éstas son independientes del tiempo de simulación.

5.3.Funciones solares (*Functions*)

A continuación se detallan cada una de las funciones:

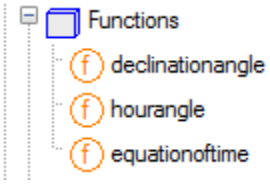


Figura 5.4: Estructura de la sub-librería Functions

5.3.1. Ángulo de declinación (*declinationangle*)

El ángulo de declinación de la Tierra es la posición angular del sol a mediodía solar con respecto al plano del ecuador. Este ángulo es debido al ángulo entre el eje de rotación y el eje de traslación de la tierra, siendo 23.45º.

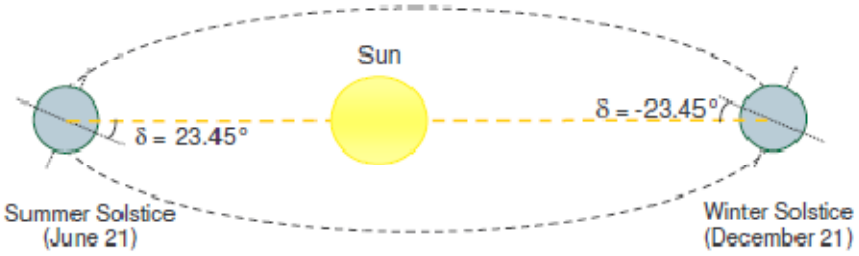


Figura 5.5: Ángulo de declinación en función de la traslación terrestre

Tal como se puede observar en la Figura 5.5, en el movimiento de traslación de la Tierra alrededor del sol, el ángulo de declinación va cambiando entre -23.45º (solsticio de invierno boreal) y +23.45º (solsticio de verano boreal) [9].

La expresión del ángulo de declinación es la siguiente [40]:

$$\delta = 23.45 \sin(360 \frac{284 + n}{365}) \qquad \text{Ec. (5.2)}$$

Donde,

δ: ángulo de declinación de la Tierra, en grados.
n: día del año, de 1 (primero de enero) hasta 365 (último de diciembre).

La variación del ángulo de declinación de la Tierra a lo largo del año se puede observar en la Figura 5.6.

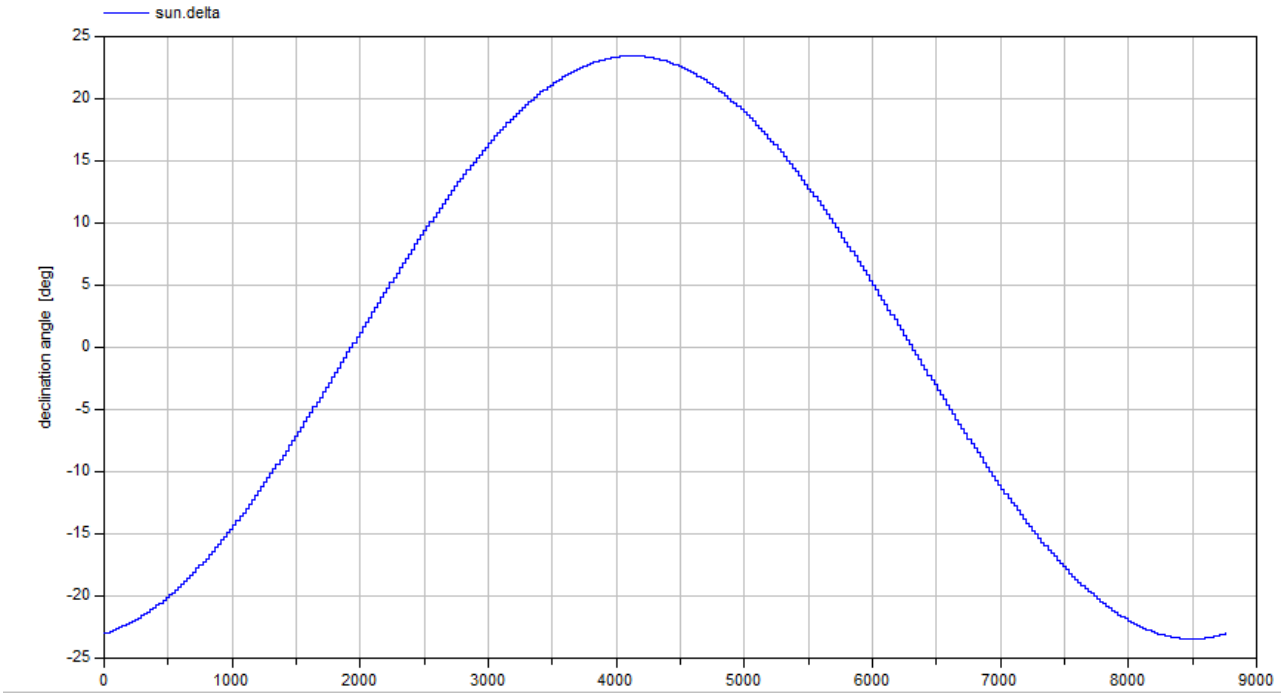


Figura 5.6: Variación del ángulo de declinación en el año

En Modelica se ha programado como una función cuya entrada es el día del año (n) y su salida el ángulo de declinación (δ).

5.3.2. Ángulo horario (*hourangle*)

El ángulo horario es el desplazamiento angular del sol hacia el este o el oeste del meridiano local. En el hemisferio norte éste es negativo por la mañana, se hace cero cuando el sol cruza el meridiano local (mediodía solar) y es positivo por la tarde [9].

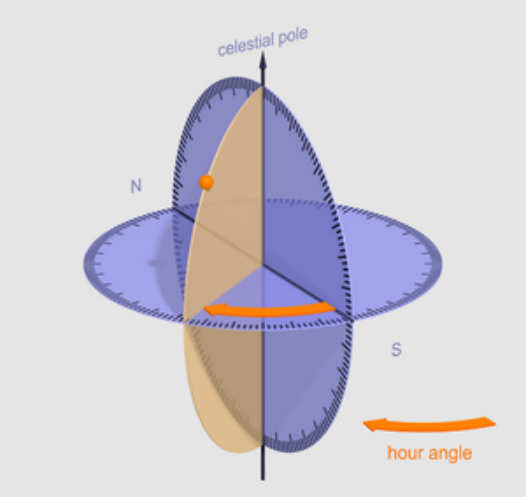


Figura 5.7: Ilustración del ángulo horario

La expresión del ángulo horario es la siguiente:

$$\omega = (SoTime - 12) \cdot 15^{o}/h \qquad \text{Ec. (5.3)}$$

Donde,

ω: ángulo horario, en grados.
SoTime: hora solar.

La Ec. (5.3) evalúa el ángulo horario en función de la hora solar y estableciendo la velocidad de rotación de la Tierra en 15^{o}/h.

La hora solar difiere en la hora estándar. La hora solar es la corregida para que coincida el mediodía solar exactamente a las 12:00. La corrección depende del meridiano local, del meridiano estándar de la hora local y de la ecuación del tiempo. La expresión de la hora solar se introduce en el apartado 5.5.2, en el modelo solar (*Sun*).

En Modelica se ha programado el ángulo horario como una función cuya entrada es la hora solar (SoTime) y su salida el ángulo horario (ω).

5.3.3. Ecuación del tiempo (*equationoftime*)

La ecuación del tiempo aparecerá en la expresión de la hora solar y tiene en cuenta las pequeñas irregularidades en la duración del día debidas a la órbita elíptica de traslación de la Tierra con respecto al Sol.

La expresión de la ecuación del tiempo es la siguiente [41]:

$$E = 222.18\,(0.000075 + 0.001868\cos\left(\frac{360}{362}\cdot(n-1)\right) - 0.032077\sin\left(\frac{360}{362}\cdot(n-1)\right) - 0.014615\cos\left(2\cdot\frac{360}{362}\cdot(n-1)\right) - 0.04089\sin\left(2\cdot\frac{360}{362}\cdot(n-1)\right)$$

Ec. (5.4)

Donde:

E: corrección del tiempo, en minutos.
n: día del año, de 1 (primero de enero) hasta 365 (último de diciembre).

La variación de la corrección del tiempo (E) a lo largo del año se muestra en Figura 5.8:

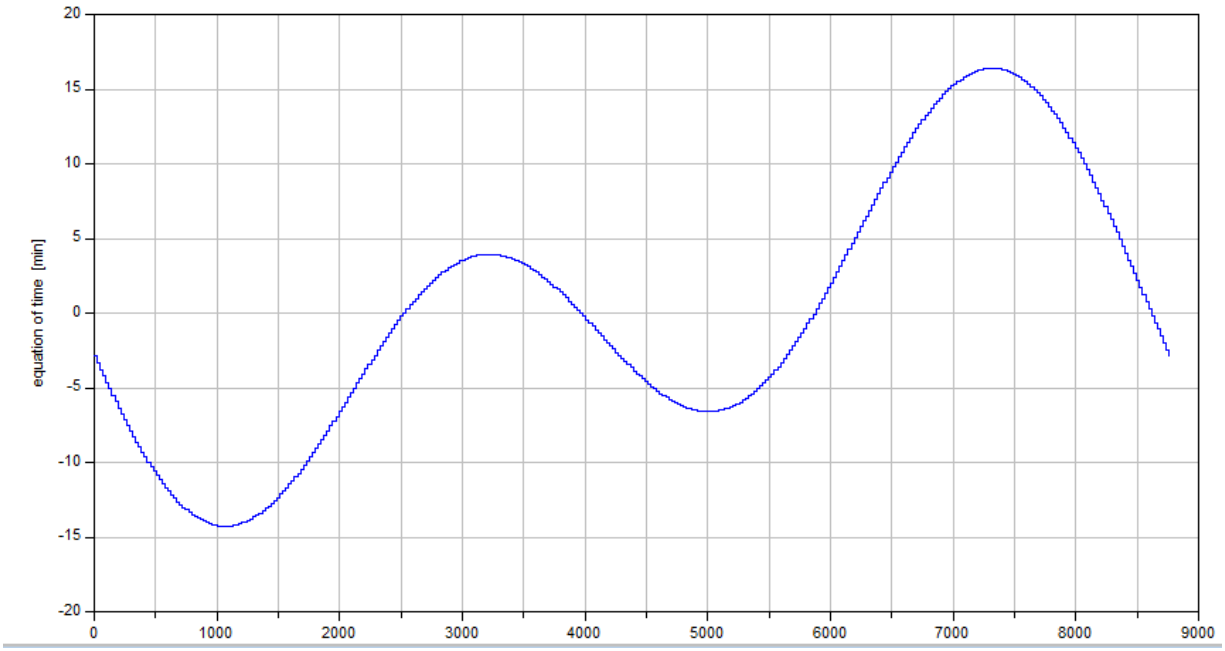


Figura 5.8: Variación anual de la correción del tiempo

5.4.Datos meteorológicos (Meteo)

Esta sub-librería está compuesta por datos de localización (latitud, longitud y meridiano local) y datos de condiciones ambientales (DNI, temperatura ambiente y velocidad del viento).

5.4.1. Datos de localización

Los datos de latitud y longitud están fácilmente accesibles a través de internet, en la aplicación de mapas de Google [42]. El meridiano local es aquel meridiano en el cuál se basa la hora local de la localización en cuestión.

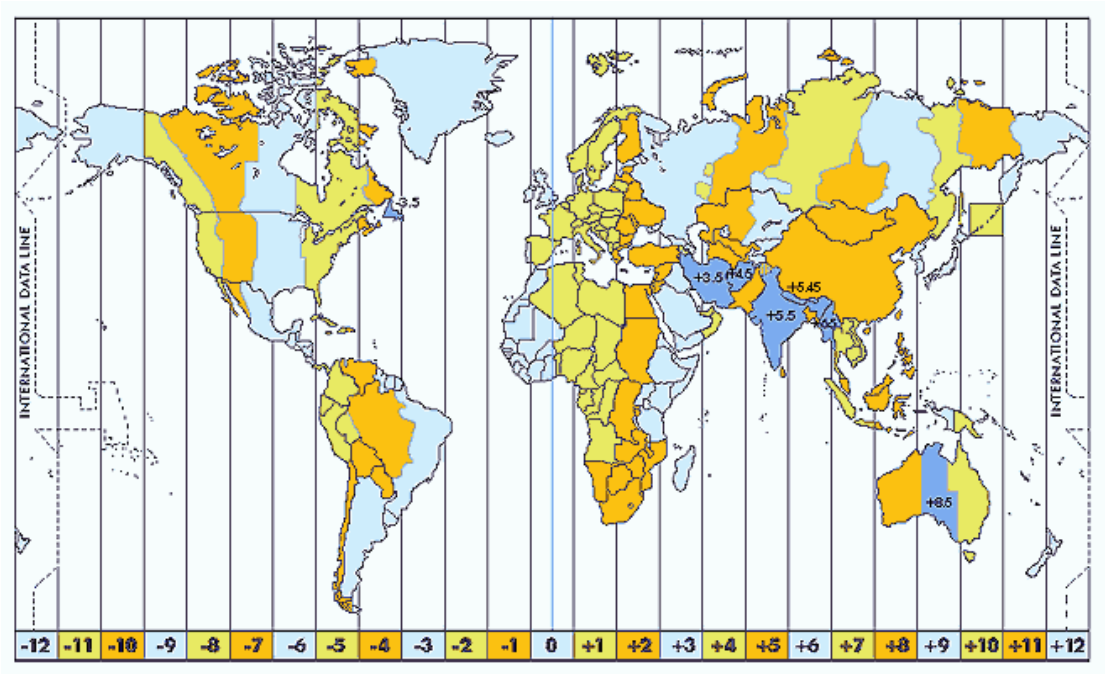


Figura 5.9: Mapa de husos horarios

Este meridiano se correlaciona con el uso horario del lugar, ya que cada huso horario tiene 15 grados de arco:

$$meridiano = X \cdot 15^{o}/hora \qquad \text{Ec. (5.5)}$$

Donde:

meridiano: meridiano estándar en el que se basa la hora local, en grados.
X: desfase horario con respecto a la hora del meridiano de Greenwich (GMT).

Por ejemplo, España está situada en la franja horaria GMT+1, hora central europea. Por tanto el valor de X es igual a 1 y su meridiano estándar por el que se rige la hora española y la de Europa central es 15^{o}.

5.4.2. Datos de condiciones ambientales

Los datos de condiciones ambientales que inciden directamente en el modelado de las plantas termosolares son básicamente tres: DNI, temperatura ambiente y velocidad de viento.

El DNI es el principal dato, considerándose el combustible de la planta termosolar.

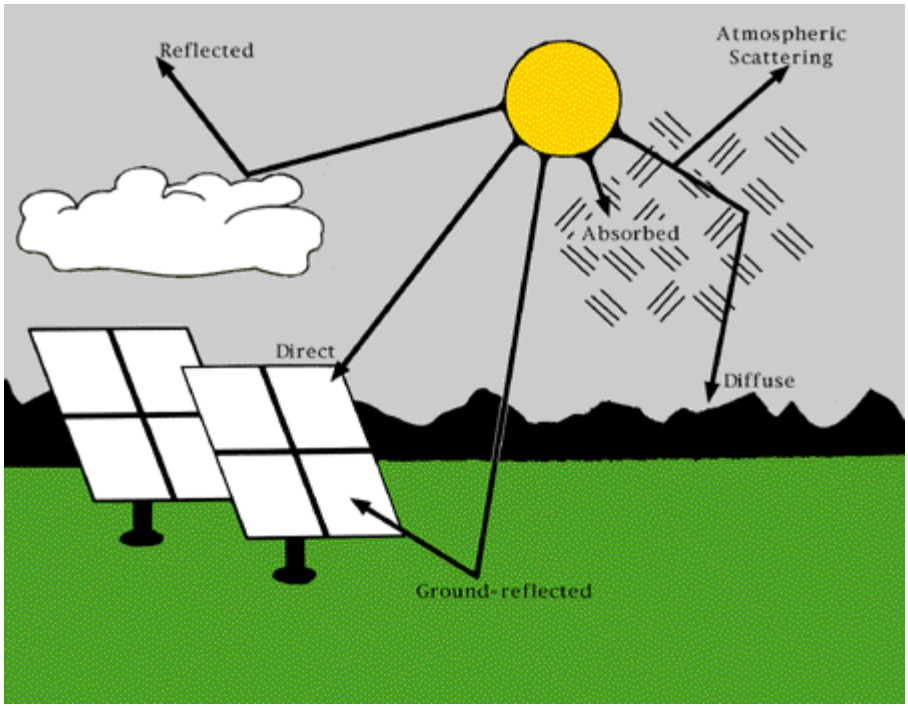


Figura 5.10: Descomposición de la radiación solar

La radiación solar que llega a un colector solar (después de que parte sea absorbida, reflejada o dispersada por la atmósfera) se divide en radiación directa, radiación difusa y radiación reflejada, tal como se muestra en la Figura 5.10, [43].

De todas ellas, la radiación útil para la tecnología solar de concentración sólo es la radiación directa. El valor de referencia es la radiación normal directa (DNI) puesto que se mide el rayo solar con un ángulo de incidencia normal (90^{o}) al instrumento de medida, que suele ser un pirheliómetro.

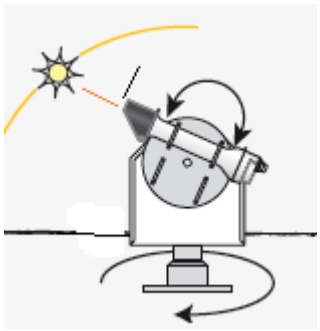


Figura 5.11: Ilustración de un pirheliómetro, medidor de DNI

La temperatura ambiente y la velocidad del viento también son importantes para el modelado de las plantas termosolares, aunque mucho menos en comparación con el DNI. La temperatura ambiente y la velocidad del viento gobiernan las pérdidas térmicas de los receptores solares por convección térmica y por radiación.

5.4.3. Introducción de datos

La sub-librería tiene un componente patrón de la clase *record* llamado *data* y los datos de las diferentes localizaciones son extensiones de este modelo.

```
record data

constant Angle_deg latitude "latitude of the plant location";
constant Angle_deg longitude "local meridian of the collector site";
constant Angle_deg meridian "standard meridian for the local time hour";

constant Real DNI[:, :]=fill(0,0,0)
    "2D table that correlates time,day of year and DNI";
constant Real AmbTemp[:,:]=fill(0,0,0)
    "2D table tahat correlates time, day of year and ambient temperature";
constant Real WindSpeed[:,:]=fill(0,0,0)
    "2D table that correlates time, day of year and ambient temperature";

end data;
```

Figura 5.12: Modelo *data* de la clase *record*.

A partir del modelo de la Figura 5.12 se construyen los datos meteorológicos de las diferentes localizaciones.

El DNI, la temperatura ambiente (*AmbTemp*) y la velocidad del viento (*WindSpeed*) se añaden a los modelos clase *record* como constantes encuadradas en tablas bidimensionales de M filas y N columnas con la siguiente estructura:

Tabla 5.2: Tabla genérica de DNI en los datos meteorológicos.

0	n[j]	n[j+1]	[...]	n[N]
StTime[i]	DNI(i,j)	DNI[i,j+1]	DNI[i,...]	DNI[i,N]
[...]	DNI[...,j]	DNI[...,j+1]	DNI[...,...]	DNI[...,N]
StTime[M]	DNI[M,j]	DNI[M,j+1]	DNI[M,...]	DNI[M,N]

Donde:

n: del año, de 1 (primero de enero) hasta 365 (último de diciembre).
StTime: hora estándar, de 0:00 (12 en punto de la noche) a 23:59 (11:59 de la noche).
DNI[i,j]: radiación normal directa a las StTime[i] horas del día del año n[j].

Las tablas para la temperatura ambiente (*AmbTemp*) y para la velocidad el viento (*WindSpeed*) son equivalentes a la Tabla 5.2.

En *ParaTrough* se han seguido dos conceptos para introducir datos meteorológicos de localizaciones: introducción del año típico meteorológico (TMY) e introducción de datos diarios.

5.4.3.1. Introducción del año típico meteorológico (TMY)

Los datos del año típico meteorológico (TMY) para una localización específica son estudios de registros meteorológicos en administraciones, de datos en estaciones meteorológicas cercanas y medidas en campo para obtener finalmente un perfil meteorológico anual patrón que defina a la localización. Los TMYs son ampliamente utilizados para realizar estudios de viabilidad de plantas termosolares. Están compuestos de promedios horarios de numerosas variables solares y ambientales.

Existen formatos específicos en extensión *.csv* (texto plano dividido por comas) para almacenar los datos de forma estandarizada. Los formatos más utilizados son el TMY2, TMY3 e INTL y difieren entre sí en las variables que contemplan y el orden en que aparecen en el archivo. Por ejemplo, el formato TMY2 contempla las siguientes variables y en este orden:

Year,Month,Day,Hour,GHI,DNI,DHI,Tdry,Twet,RH,Pres,Wspd,Wdir,Albedo

Dónde:

Year: año.

Month: mes.

Day: día.

Hour: hora.

GHI: *Global Horizontal Irradiation*. Radiación horizontal global.

DNI: *Direct Normal Irradiation*. Radiación normal directa.

Tdry: temperatura seca, equivalente a la temperatura ambiente.

Twet: temperatura de bulbo húmedo.

RH: humedad relativa.

Press: Presión atmosférica.

Wspd: velocidad del viento.

Wdir: dirección del viento.

Albedo: factor de reflexión por efecto albedo.

El software libre SAM [4] posee una amplia librería TMYs de más de 1500 localizaciones diferentes. En *ParaTrough* se han extraído los valores de DNI, temperatura ambiente y velocidad de viento de los TMYs de varias localizaciones y se han introducido en los datos meteorológicos de la sub-librería *Meteo*.

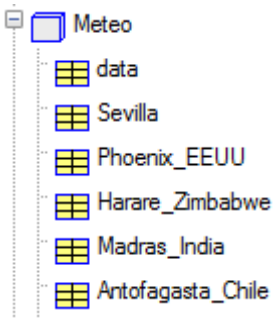


Figura 5.13: TMYs en formato ParaTrough de diferentes localizaciones

La conversión desde los formatos TMY2 y INTL al formato de disposición de datos *ParaTrough* de la Tabla 5.2 se ha realizado usando las herramientas externas *INTLconverter.xlsx* y *TMY2converter.xlsx*. Para más información sobre estas herramientas, véase el Anexo B: Conversión de TMY a ParaTrough.

Las localizaciones cargadas en *ParaTrough* aparecen en la Figura 5.13 y se detallan a continuación:

Tabla 5.3: Detalles de las localizaciones con TMY cargado en ParaTrough

Datos Meteorológicos	Planta Termosolar cercana	Latitud	Longitud	Huso Horario	Meridiano Hora Local
Sevilla	Los Arenales [44]	37.12	-5.56	GMT+1	15
Phoenix_EEUU	Solana [45]	32.93	-112.98	GMT-7	-105
Harare_Zimbabwe	KaXu Solar One [46]	-17.92	31.13	GMT+2	30
Madras_India	N/A	13.00	80.18	GMT+5.5	82.5
Antofagasta_Chile	N/A	-23.43	-70.43	GMT-4	-60

Las localizaciones de la Tabla 5.3 han sido escogidas para tener una localización por cada continente y que tres de ellas tengan una planta termosolar ya construida cerca. La sub-librería *Meteo* es fácilmente ampliable aprovechando los TMYs libres de SAM y siguiendo el procedimiento del Anexo B: Conversión de TMY a ParaTrough.

5.4.3.2. Introducción de datos diarios

La estructura de datos a introducir en modelos de la clase *record* de la sub-librería *Meteo*, reflejada en la Tabla 5.2, es flexible en lo que se refiere al número de filas (M) y al número de columnas (N). Los datos de los TMYs son promedios horarios a lo largo

de todo el año y por tanto no tienen suficiente resolución para analizar en detalle la operación diaria de una planta solar. Para ello se introducen dos modelos de datos meteorológicos de clase *record* con datos minutales de DNI, temperatura ambiente y velocidad de viento durante 24 horas.

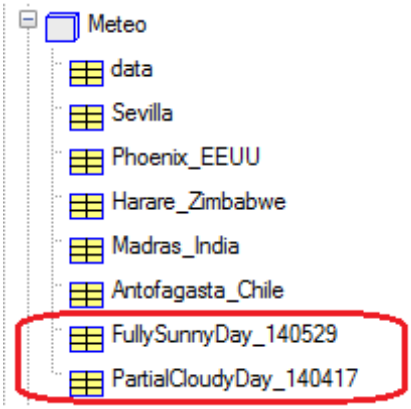


Figura 5.14: Datos diarios con resolución minutal en la sub-librería Meteo

Las características de los dos conjuntos de datos meteorológicos se detallan en la siguiente tabla:

Datos Meteorológicos	Día de cogida de datos	Características del día	Planta Termosolar cercana	Latitud	Longitud	Huso Horario	Meridiano Hora Local
FullySunnyDay_140529	29/05/2014	Día totalmente soleado	Andasol	37.20	-3.10	GMT+1	15
PartialCloudyDay_140417	17/04/2014	Día parcialmente nublado					

Estos datos con más resolución también son fácilmente introducibles en *ParaTrough* a partir de archivos de registro de las estaciones meteorológicas de las plantas termosolares en operación.

5.5.Modelos principales (*Models*)

Los modelos que componen esta sub-librería son las herramientas de interpolación de datos (*DNI, AmbTemp, WindSpeed, WeatherEvaluation*) y el modelo solar (*Sun*).

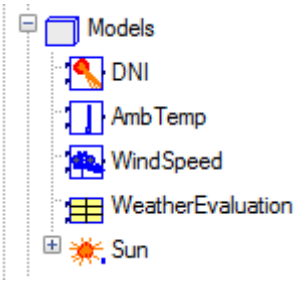


Figura 5.15: Modelos en *Models*

5.5.1. Herramientas de interpolación de datos

Los modelos *DNI, AmbTemp* y *WindSpeed* son extensiones del modelo genérico *Generic2DDataTable* (ver apartado 5.2) pero particularizados para que la tabla bidimensional incrustada sea de datos de DNI, temperatura ambiente y velocidad de viento, respectivamente. Las filas corresponden a la hora estándar y las columnas al día del año.

El modelo *WeatherEvaluation* une los tres modelos anteriores mediante composición gráfica (véase Figura 5.16) para evaluar todos los datos meteorológicos de interés de manera conjunta.

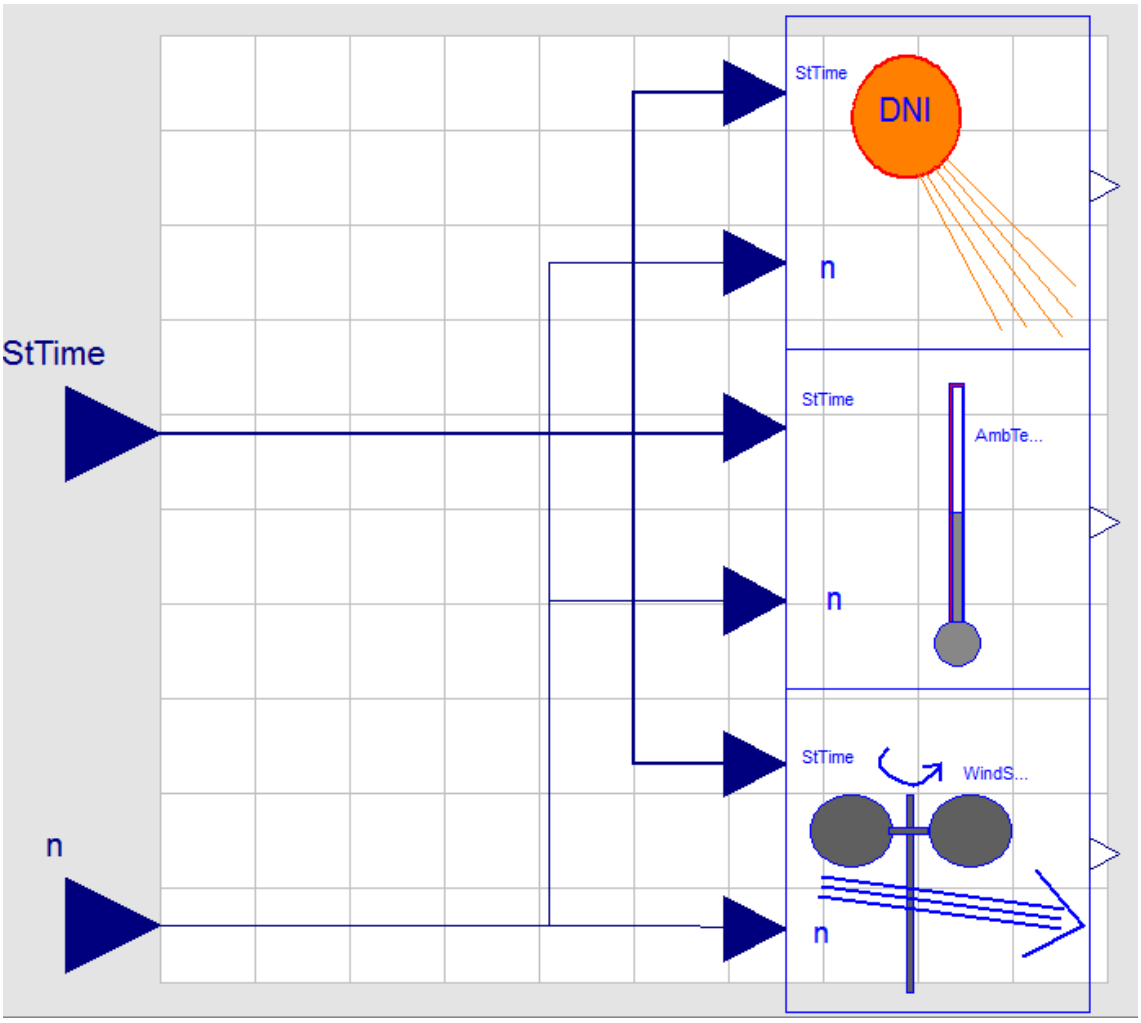


Figura 5.16: Composición del modelo *WeatherEvaluation*

El modelo *WeatherEvaluation* y todos los sub-modelos integrados en éste tienen como entradas la hora estándar (*StTime*) y el día del año (*n*).

5.5.2. Modelo solar (*Sun*)

El modelo *Sun* condensa todos los componentes de la sub-librería *A_SolarResource* en un modelo físico que simula la posición del sol en cualquier momento del año en cualquier localización del planeta y que para esta localización evalúa con la resolución de los datos de entrada los datos meteorológicos de DNI, temperatura ambiente y velocidad del viento en función del tiempo.

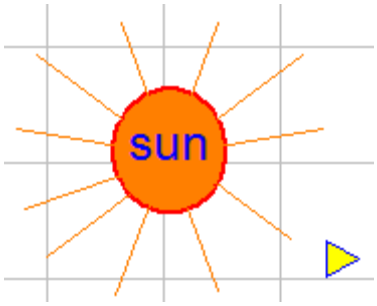


Figura 5.17: Icono del modelo *Sun*

El modelo cuyo icono se puede observar en la Figura 5.17 tiene en su interfaz un conector de la clase *weather_connector* (véase apartado 4.4).

Como datos reemplazables tiene la variable de clase *record* llamada *data*, que llama a los datos meteorológicos (*records* del apartado 5.4).

Los parámetros a introducir son el día inicial (*day_0*) y el mes inicial (*month_0*) en los que se quiere que empiece la evaluación solar.

El modelo *Sun* se compone del sub-modelo llamado *WeatherEvaluator*, de la clase *WeatherEvaluation* (ver 5.5.1) para la interpolación de los datos meteorológicos (*DNI, AmbTemp y WindSpeed*) de *data* a partir de los valores de los parámetros de día (*day)*, mes (*month*) y la hora estándar (*StTime*). Se define la variable de DNI acumulado como la integral del DNI con respecto al tiempo de simulación:

$$DNI_{acc} = \int_0^{time} DNI\ dt \qquad \text{Ec. (5.6)}$$

Esta variable resulta de interés para analizar el recurso solar total a lo largo de un periodo concreto de tiempo.

Para la simulación del paso del tiempo (horas, días y meses), se añade a *Sun* las tablas *DayOfYear*, *Day_from_n* y *Month_from_n* (véase apartado 5.2) como sub-modelos. La primera tabla calcula el día del año (*n*) a partir del día y mes inicial de simulación (*day_0* y *month_0*) y hace que *n* aumente en una unidad cada 24 horas de tiempo de

simulación. La segunda y tercera tabla calculan el día y mes corriente (*day* y *month*) a partir del valor creciente de *n.*

Para definir la posición del sol, aparte de los ángulos ya definidos de ángulo de declinación (ver apartado 5.3.1) y ángulo horario (ver apartado 5.3.2), se define el ángulo cenital como el ángulo formado entre el cenit de la localización y la posición del sol.

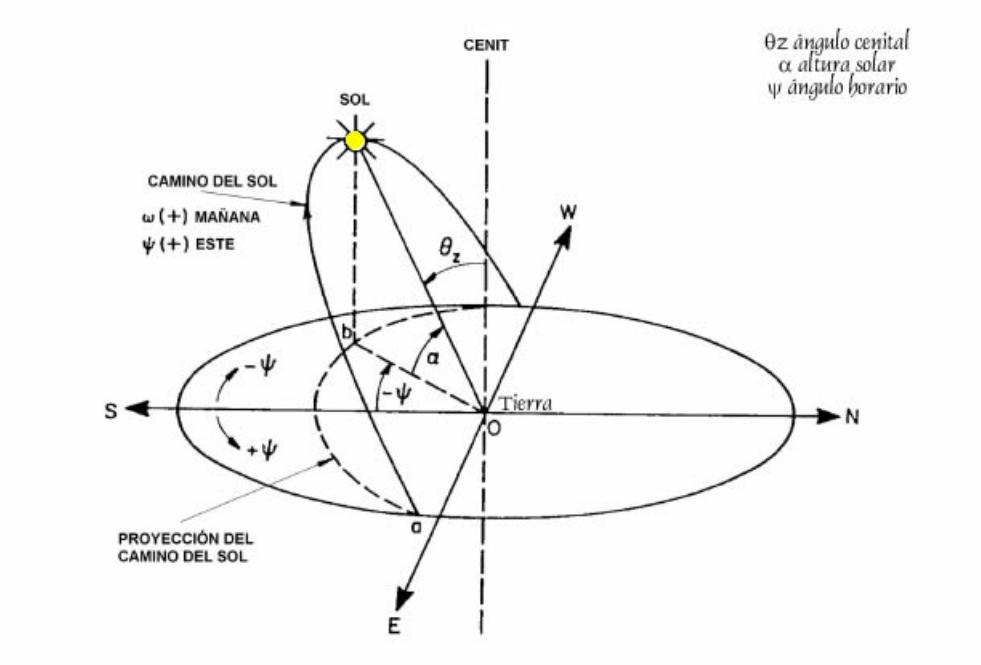


Figura 5.18: Ángulos que definen la posición del sol

En la Figura 5.18 se ilustran gráficamente los tres ángulos que definen perfectamente la posición del sol: ángulo centital, ángulo horario y altura solar.

- ✓ Ángulo cenital (θ_z): se relaciona con los demás ángulos y con la latitud del lugar mediante la siguiente Ec. [47]:

$$\cos(\theta_z) = \cos(\delta)\cos(latitude)\cos(\omega) + \sin(\delta)\sin(latitude) \qquad \text{Ec. (5.7)}$$

Dónde:

θ_z: ángulo cenital, en grados.
δ: ángulo de declinación, en grados.
latitude: latitud de la localización, en grados.
ω: ángulo horario, en grados.

- ✓ Altura solar (α): es el ángulo complementario al ángulo cenital:

$$\alpha = \begin{cases} 90 - \theta_z, & \theta_z < 90 \\ 0, & \theta_z \geq 90 \end{cases} \qquad \text{Ec. (5.8)}$$

Dónde:

α: altura solar, en grados.
θ_z: ángulo cenital, en grados.

Se ha definido la ecuacíon Ec. (5.8) por partes para tener en cuenta la altura solar sólo cuando el sol está visible (por el día) y no tener alturas solares negativas (por la noche, cuando el ángulo cenital es mayor que 90^o), ya que no tiene sentido físico.

El ángulo horario (*omega*) depende de la hora solar (*SoTime*), tal como se indica en el apartado 5.3.2. Ésta es ligeramente diferente a la hora estándar (*StTime*) y se relacionan mediante la siguiente ecuación:

$$SoTime = StTime - DST + \frac{(meridian - longitude)}{15} + E \cdot \frac{1h}{60min} \quad \text{Ec. (5.9)}$$

Dónde:

SoTime: hora solar.
StTime: hora estándar.

DST: *Daylight Savings Time*. Ajuste de hora: 1 en horario de verano y 0 en horario de invierno. En *Sun* se ha definido DST como 1 desde el 27 de marzo (*n*=86) hasta el 27 de octubre (*n*=300*)* y 0 para el resto. Dependiendo de qué año se evalúe, se deberá cambiar por las fechas reales.

meridian: meridiano estándar para la hora local, en grados. Se toma de los datos de localización de *data*.

longitude: longitud de la localización. Se toma de los datos de *data*.

E: corrección del tiempo, en minutos. Se utiliza la función del apartado 5.3.3.

Por último se establece la relación entre la variable tiempo de simulación (*time*) con la hora estándar (*StTime*).

El tiempo de simulación empieza (*time*=0) en el día inicial (*day_0*) y mes inicial (*month_0*) escogidos como parámetros para la evaluación y tiene dimensión de horas en *ParaTrough* y tal como está predefinido en *Modelica*, avanza positivamente hasta el tiempo de parada de la simulación.

La hora estándar (*StTime*) responde a la siguiente fórmula, relacionada con el tiempo de simulación (*time*):

$$StTime = time - (n - n_0) \cdot 24 \frac{h}{día}$$ Ec. (5.10)

Dónde:

StTime: hora estándar.

time: tiempo de simulación, en horas.

n: día del año (de 1 a 365).

n_0: día del año inicial, calculado a partir del día inicial (*day_0*) y mes inicial *(month_0*) en el sub-modelo *DayOfYear*.

La Ec. (5.10) hace que la hora estándar (*StTime*) se reinicie cada 24 horas del tiempo de simulación (*time*). Así se simula el comportamiento cíclico de la hora estándar del día con independencia del tiempo de simulación. Con esta ecuación, el modelo solar (*sun*) también aporta el tiempo de simulación y el tiempo real cuando compone otros modelos más complejos.

5.6.Ejemplos

Se han diseñado dos ejemplos para la comparación del recurso solar.

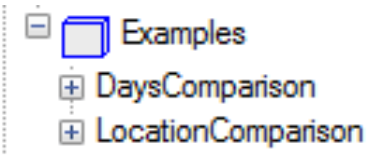


Figura 5.19: Ejemplos de la sub-librería A_SolarResource

5.6.1. Comparación de tipos de día (*DaysComparison*)

El primer modelo está especificado para comparar diferentes tipos de días en una misma localización.

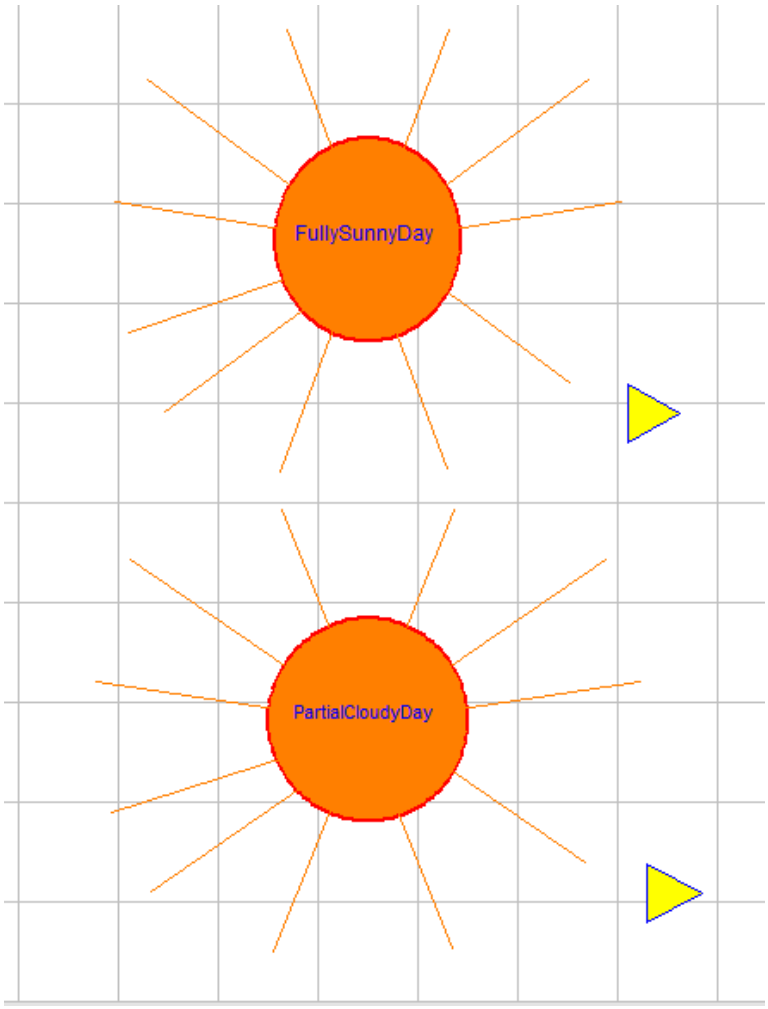


Figura 5.20: Modelo para comparación entre un día completamente soleado y un día parcialmente nublado en Aldeire

Se compone el ejemplo de dos modelos solares (*Sun*), uno para un día totalmente soleado (que utiliza los datos meteorológicos de la clase record *FullySunnyDay_140529*) y otro para un día parcialmente nublado (que utiliza los datos meteorológicos de la clase record *PartialCloudyDay_140417*). Se debe tener la precaución de insertar adecuadamente en día y mes inicial (*day_0* y *month_0*) en cada uno de los modelos.

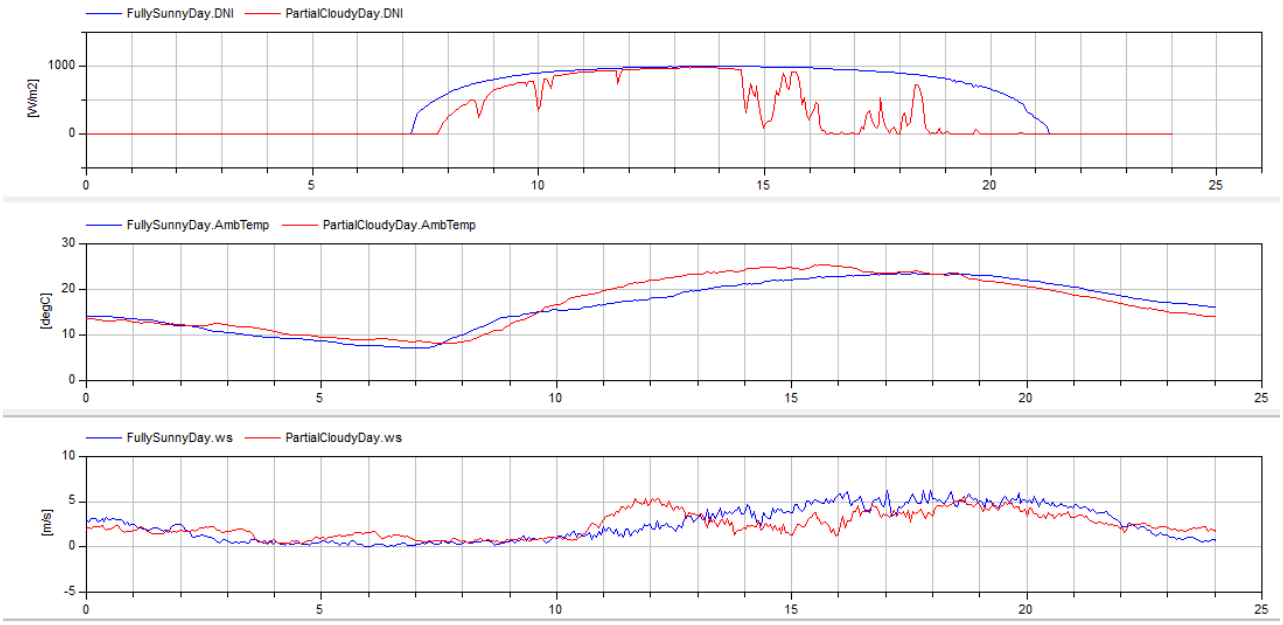


Figura 5.21: Simulación de DNI, temperatura ambiente y velocidad de viento para dos tipos de día

La Figura 5.21 muestra la comparación entre el DNI (gráfica superior), temperatura ambiente (gráfica intermedia) y velocidad de viento (gráfica inferior) para dos días diferentes en la misma localización.

5.6.2. Comparación entre localizaciones (*LocationComparison)*

El segundo ejemplo está especificado para comparar datos meteorológicos de diferentes localizaciones.

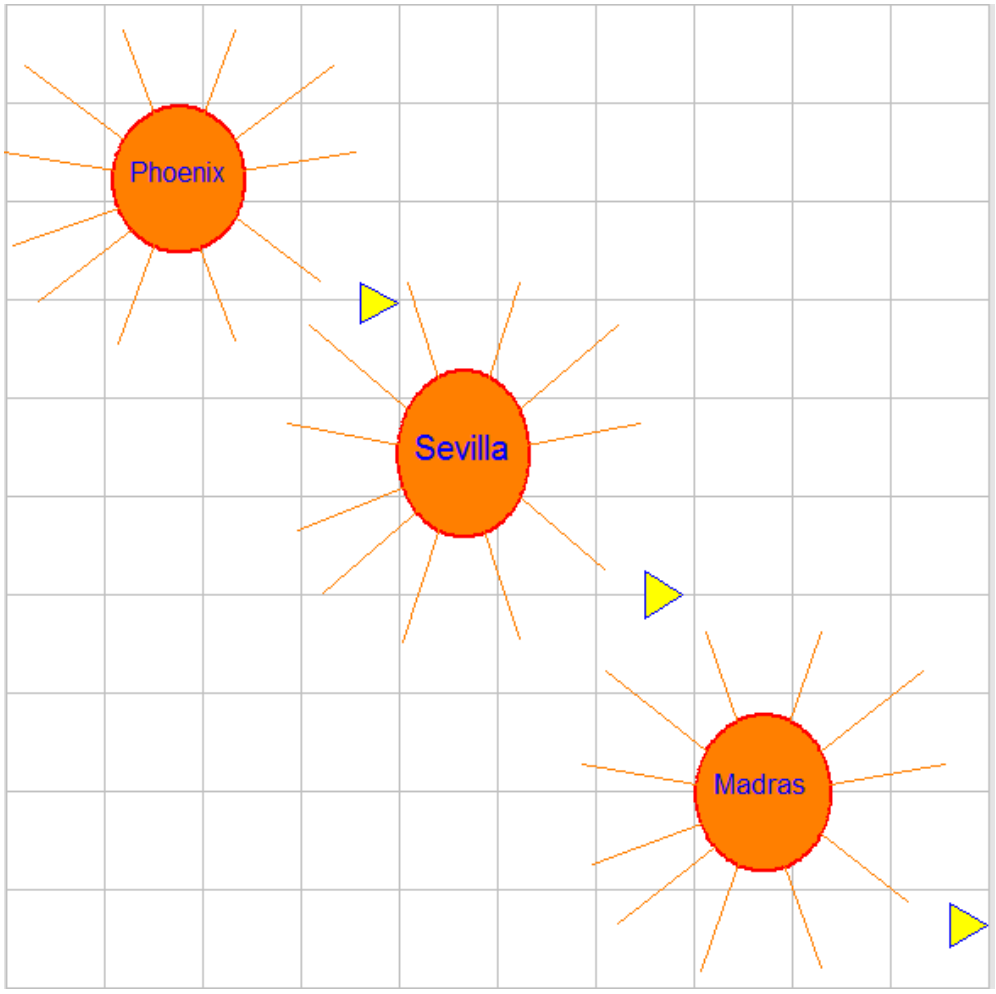


Figura 5.22: Modelo para comparación meteorológica entre Phoenix, Sevilla y Madras

Se compone el ejemplo de tres modelos solares (*Sun*), cada datos meteorológicos de Phoenix, Sevilla y Madras. Estos datos se llaman desde la sub-librería *Meteo*, mediante la redeclaración de la clase *record* llamada *data* para cada uno de los sub-modelos que componen el ejemplo.

En el ejemplo se modela además unos parámetros de día inicial y mes inicial para que alimenten simultáneamente los valores de *day_0* y *month_0* a los tres sub-modelos, asegurando que los tres modelos empiezan a simularse en el mismo día del año.

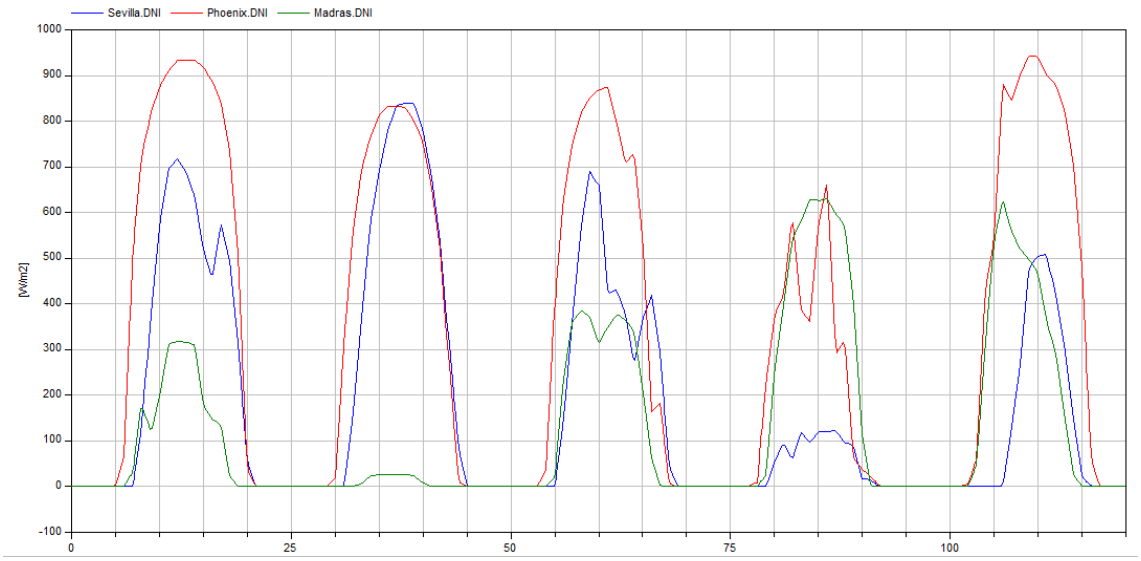


Figura 5.23: Comparación de DNI del TMY entre Phoenix, Sevilla y Madras desde el 3 al 7 de junio

En la Figura 5.23 se observa en el periodo de tiempo escogido Phoenix (línea roja) tiene mejores valores de DNI que Sevilla (línea azul) y que Madras (línea verde).

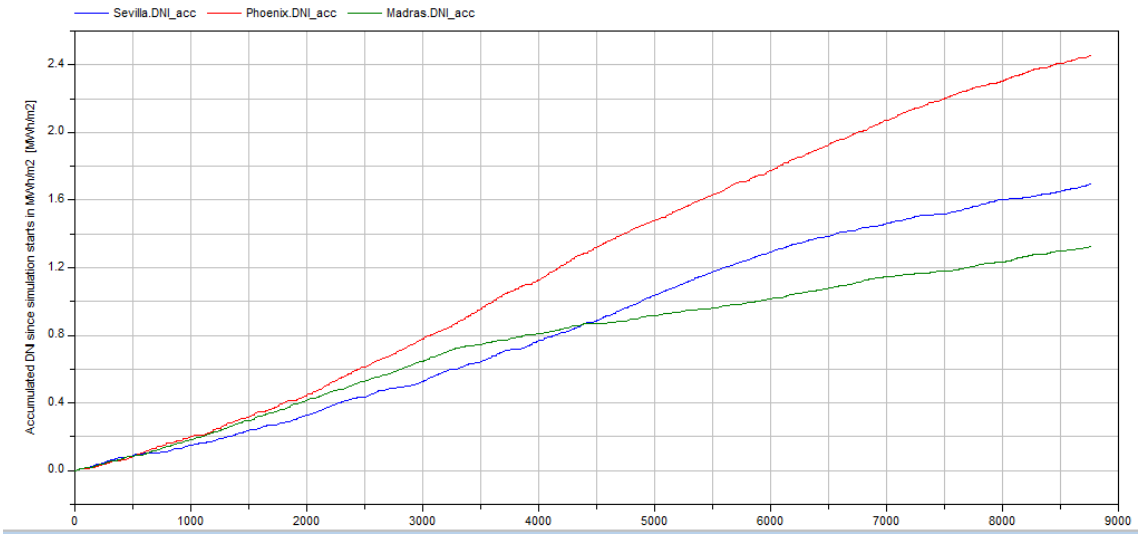


Figura 5.24: DNI acumulado en un año para Phoenix, Sevilla y Madras

En la Figura 5.24 se determina el valor anual de DNI acumulado para las tres localizaciones estudiadas. Se observa que Phoenix (línea roja) es el lugar donde se tiene un mayor recurso solar, alcanzando 2.45 $MWh \cdot m^{-2} \cdot año^{-1}$. Madras tiene más energía irradiada que Sevilla en el primer tercio del año pero finalmente la capital andaluza sobrepasa a mitad del año a la ciudad india y obtiene un DNI acumulado anual de 1.69 $MWh \cdot m^{-2} \cdot año^{-1}$. Madras acumula una radiación de 1.32 $MWh \cdot m^{-2} \cdot año^{-1}$.

Con este mismo ejemplo se pueden sacar numerosas comparaciones meteorológicas entre las diferentes localizaciones.

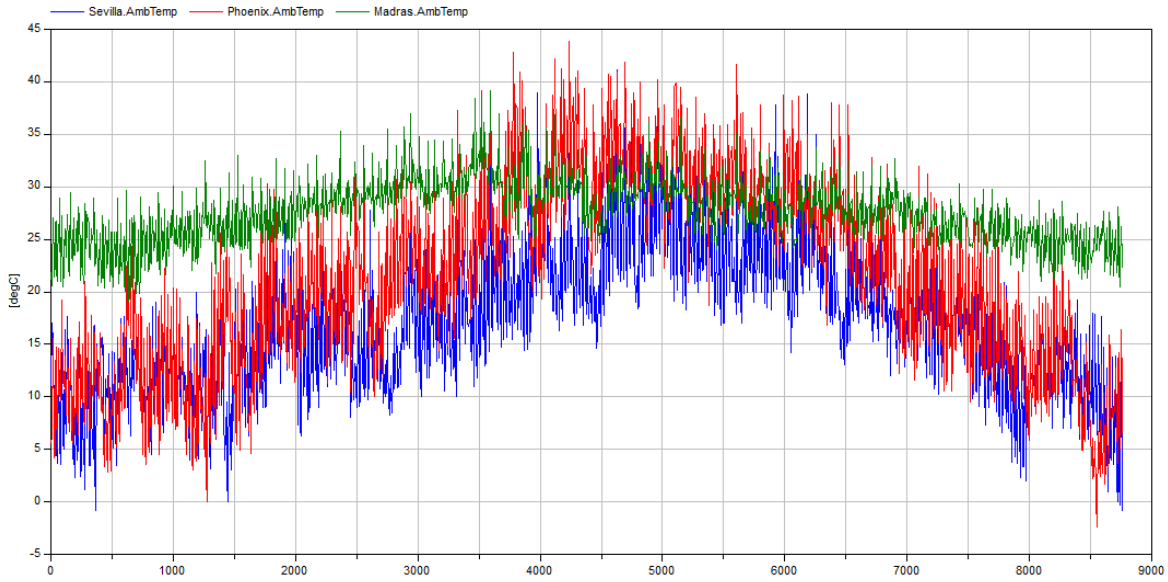


Figura 5.25: Comparativa de temperatura ambiente entre Phoenix, Sevilla y Madras para todo el TMY

En la Figura 5.25 se observa que Madras tiene una variación de temperatura anual pequeña, no bajando nunca de 20^{o}C. La temperatura ambiente de Phoenix es generalmente mayor a la de Sevilla.

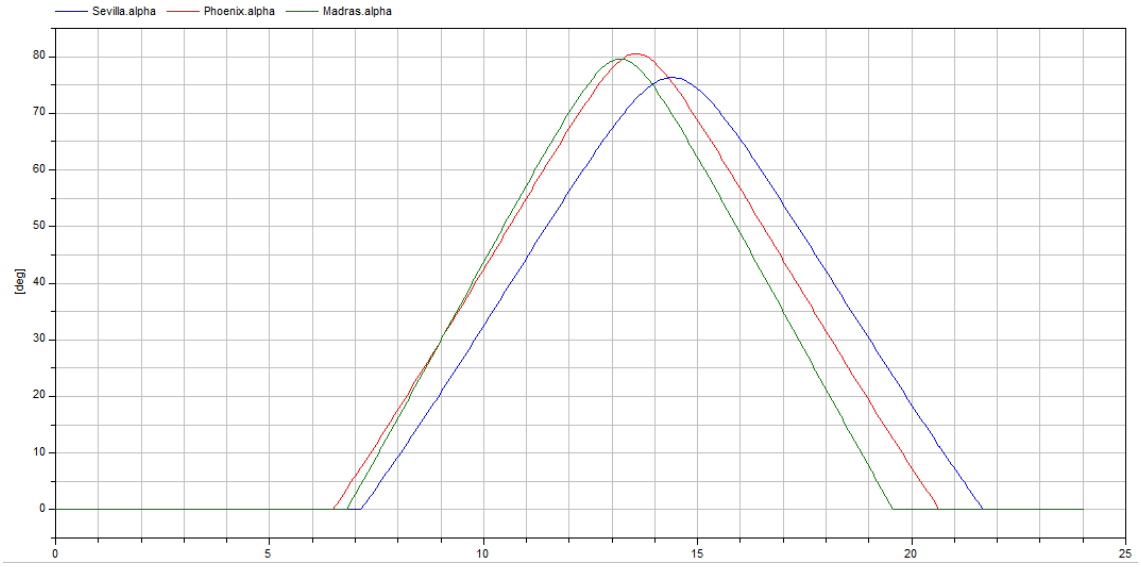


Figura 5.26: Comparativa de altura solar entre Phoenix, Sevilla y Madras para el 24 de junio

En la Figura 5.26 se observa que la máxima altura solar para este día se produce en Phoenix, alcanzando casi 80.5^{o} aproximadamente a las 13:30 hora local. Un dato característico es la diferencia entre la puesta de sol entre las tres localizaciones, identificable cuando la curva de la altura solar llega a cero por la tarde:

Tabla 5.4: Hora de la puesta de sol entre Phoenix, Sevilla y Madras el día 24 de junio

Localización	Hora de la puesta de sol
Phoenix	20:36
Sevilla	21:39
Madras	19:34

Nótese que el eje de tiempo en la abscisa se refiere a la hora local, por lo que no tienen relación de simultaneidad entre sí las curvas. Se simula de esta manera para facilitar la interpretación de los resultados.

5.7.Futuras ampliaciones

En el futuro se pretende mejorar el sistema de adquisición de nuevos datos ya sea desde bases de datos abiertas o desde sensores de estaciones meteorológicas.

Para ello se pretende realizar una herramienta empotrada en Modelica para importar los datos meteorológicos del SAM de una manera más amigable e integrada. Otro futuro foco de ampliación es comunicar datos en tiempo real desde estaciones meteorológicas de plantas en operación con Modelica a partir de interfaces existentes entre Dymola y Matlab, con su toolbox de procesamiento de señales en tiempo real.

5.8.Conclusiones

Con la sub-librería A_SolarResource se posee una aceptable cantidad de datos meteorológicos de diferentes localizaciones y lo que es más importante, la fácil posibilidad de ampliación de estos datos a través de los TMYs de aplicaciones libres como SAM o de otras fuentes. La flexibilidad en la resolución de los datos es otro punto importante a destacar.

El modelo solar (*Sun*) es el combustible de los demás modelos de *ParaTrough*, de igual manera que el Sol es el combustible de toda una planta termosolar. Provee la posición exacta del sol y los datos de DNI, temperatura ambiente y velocidad de viento en función del tiempo para todo el año y para todas las localizaciones posibles. La única limitación reside en cargar los datos meteorológicos adecuados.

6. Librería del sistema de fluido térmico (B_HeatTransferFluid)

6.1. Introducción

Esta sub-librería contiene los modelos que forman el sistema de fluido térmico, que es el sistema diferencial en las plantas termosolares ya que a partir de éste se capta la energía procedente solar y se concentra mediante el calentamiento de un fluido que circula a través del colector solar. Este colector solar es el corazón de la tecnología solar cilindro-parabólica.

6.2. Colectores solares (*Collectors*)

En el mercado actual son varios los fabricantes de colectores solares y estos difieren ligeramente entre sí. Para que la librería *ParaTrough* sea una herramienta potente de comparación entre diferentes tipos de colectores, se ha definido un modelo clase *record* para recoger las constantes características de cada tipo de colector, [4].

Tabla 6.1: Características principales de los diferentes tipos de colectores solares del mercado

Tipo de colector	Área reflectora, m^2	Anchura reflectora, m	Longitud del colector, m	Número de SCEs por colector	Número de HCEs por SCE	Distancia entre filas de colectores, m	Distancia focal media, m
EuroTrough ET150 [48]	817.5	5.77	148.6	12	3	15	2.11
Luz LS2 [49]	235	5	49	6	3	15	1.8
Luz LS3	545	5.75	100	12	3	15	2.11
Solargenix SGX1 [50]	470.3	5	100	12	3	15	1.8
Albiasa Trough AT150	817.5	5.774	150	12	3	15	2.11
Siemens SunField 6	545	5	95.2	8	3	15	2.17
Sky Trough	656	6	115	8	3	15	2.15

A continuación se exponen brevemente los elementos de los que está formado un colector solar:

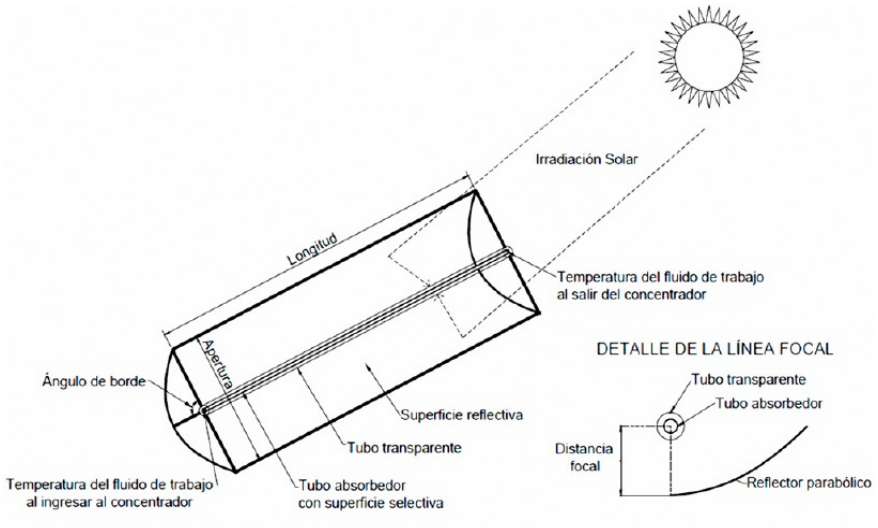


Figura 1. Diagrama esquemático de la geometría del concentrador tipo PTC.

Figura 6.1: Esquema de un colector solar cilindro-parabólico

La parábola está formada por espejos que reflejan la radiación directa y la concentran en el foco de la estructura, por donde corre una serie de tuberías con fluido térmico (HTF) en su interior. Estos tubos de absorción se denominan HCEs (*Heat Collector Elements*). Estructuralmente, el colector está formado por la unión en hilera de elementos más pequeños llamados SCEs (*Solar Collector Elements*). Cada SCE suele tener una tubería en su foco de 3 HCEs. La unión de entre 6 y 12 SCEs forma finalmente el colector solar, también llamado SCA (*Solar Collector Assembly*). Los SCAs tienen movimientos independientes unos de otros ya que poseen una unidad hidráulica de pistones para permitir su movimiento y enfoque al sol.

La unión también en hilera de 4 ó más SCAs forma un lazo solar (*loop*). Éste consigue el salto de temperatura requerido por la planta, que suele ser de 100 K entre entrada al lazo y salida.

Finalmente, la unión en paralelo de numerosos lazos consigue la energía térmica necesaria para el funcionamiento de la planta termosolar.

Además de las constantes principales definidas en la Tabla 6.1, se definen también en las mismas clases *record* de los colectores otras constantes secundarias pero también de importancia para el modelado de los diferentes componentes, [4]:

Tabla 6.2: Características secundarias de los diferentes tipos de colectores solares del mercado

Tipo de colector	Eficiencia de enfoque, %	Efectos geométricos, %	Reflectividad de los espejos, %	Eficiencia óptica general, %	IAM_0	IAM_1	IAM_2
EuroTrough ET150	99	98	93.5	12	1	0.506	-0.1763
Luz LS2	99	98	93.5	6	1	0.506	-0.1763
Luz LS3	99	98	93.5	12	1	0.506	-0.1763
Solargenix SGX1	99.4	98	93.5	12	1	0.506	-0.1763
Albiasa Trough AT150	99	98	93.5	12	1	0.506	-0.1763
Siemens SunField 6	99	96.8	92.5	8	1	-0.0753	-0.03698
Sky Trough	98.8	95.2	93	8	1	0.0327	-0.1351

Con los conjuntos de constantes dispuestos en la Tabla 6.1 y Tabla 6.2, es posible modelar en *ParaTrough* los diferentes elementos de los colectores presentes en el mercado utilizando un solo modelo para cada uno de estos (véase apartado 6.4).

6.3.Receptores solares (*Receivers*)

Análogamente a lo que ocurre con los colectores, en el mercado también existen varios modelos diferentes de receptores solares, también llamados HCEs. Estas constantes se almacenan en *ParaTrough* en tipos *record*. Las características principales de los diferentes HCEs del mercado se resumen en la siguiente tabla, [4]:

Tabla 6.3: Características principales de los diferentes tipos de HCEs del mercado

Tipo de receptor HCE	Longitud del HCE, m	Diámetro interno del tubo, m	Diámetro externo del tubo, m	Diámetro interno de la camisa de vidrio, m	Diámetro externo de la camisa de vidrio, m	Emisividad del vidrio, %	Conductividad térmica del vidrio, $W \cdot m^{-1} \cdot K^{-1}$
Schott PTR70	4	0.066	0.07	0.115	0.120	89	1.1
Schott PTR70 2008	4	0.066	0.07	0.115	0.120	89	1.1
Solel UVAC3	4	0.066	0.07	0.115	0.121	89	1.1
Siemens UVAC 2010	4	0.066	0.07	0.109	0.115	89	1.1
Schott PTR80	4	0.076	0.08	0.115	0.12	89	1.1

Los diferentes componentes del HCE se muestran en la siguiente figura:

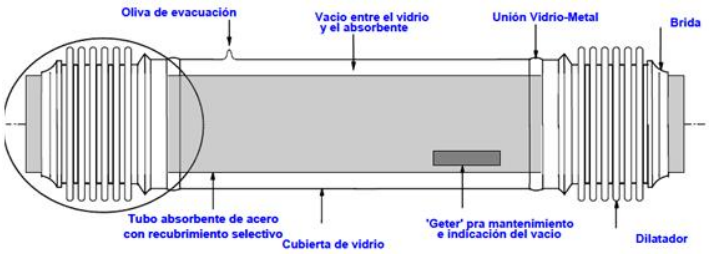


Figura 6.2: Esquema de un HCE

Tal como se muestra en la Figura 6.2, el HCE consta de un tubo metálico recubierto con una pintura especial que maximiza la absorbancia de la radiación solar y de una cubierta o camisa de vidrio a vacío que maximiza la transmitancia de la radiación solar y minimiza las pérdidas por convección térmica hacia el exterior. Entre HCEs se instala un fuelle para absorber las diferencias de dilatación térmica entre el tubo de acero y el vidrio.

Adicionalmente, los HCEs tienen otras constantes secundarias que también son almacenadas en los tipos *record* de los receptores (*receivers*) y utilizados para el modelado de los componentes principales de la sub-librería (véase apartado 6.4). Estas constantes secundarias se resumen en la siguiente tabla, [4]:

Tabla 6.4: Características secundarias de los diferentes tipos de HCEs del mercado

Tipo de receptor HCE	Factor de polvo en HCE, %	Factor de sombra por fuelle, %	Transmitancia de la camisa de vidrio, %	Absorbancia de la capa de pintura del tubo de acero, %	Factor misceláneo para contar otras pérdidas, %	Rugosidad de las paredes internas del tubo de acero, m	Pérdida media de calor estimada, $W \cdot m^{-1}$
Schott PTR70	98	96.0	96.3	96	96	2.45E-5	190
Schott PTR70 2008	98	96.0	96.3	95	96	2.45E-5	150
Solel UVAC3	98	97.1	96.0	96	96	2.45E-5	175
Siemens UVAC 2010	100	96.3	96.5	96	96	2.45E-5	192
Schott PTR80	98	93.5	96.4	96.3	96	2.45E-5	190

Con los conjuntos de constantes dispuestos en la Tabla 6.3 y Tabla 6.4, es posible modelar en *ParaTrough* los diferentes HCEs presentes en el mercado utilizando un solo modelo para cada uno de estos (véase apartado 6.4.2).

6.4.Modelos principales (*Models*)

Aquí se modelan los elementos principales del sistema de fluido térmico, tal como se ilustra en la Figura 6.3: espejos (*Mirrors*), elemento colector de energía térmica (*HCE*), elemento de colector solar (*SCE*), colector solar (*SCA)*, lazo (*Loop*) y campo solar (*SolarField*).

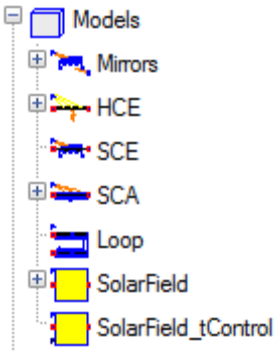


Figura 6.3: Estructura de la sub-librería *Models*

6.4.1. Espejos (*Mirrors*)

Este modelo simula la concentración de la energía procedente del sol a través de la estructura de espejos con geometría cilindro-parabólica. Se llama a la clase *record collector*, que es reemplazable y define el tipo de colector comercial en el que están montados los espejos del modelo.

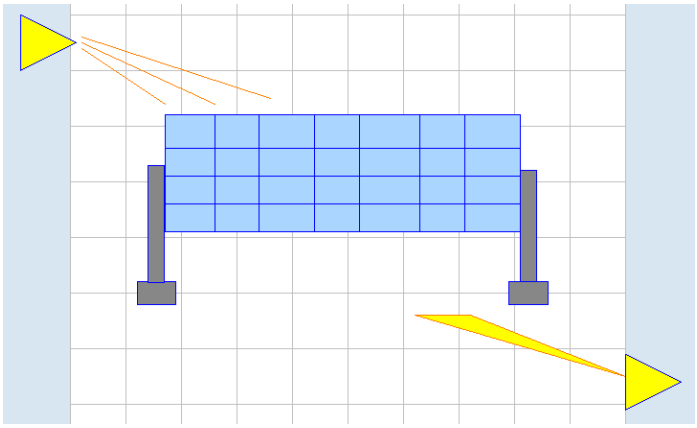


Figura 6.4: Icono del modelo *Mirrors*

Como se muestra en la Figura 6.4, la interfaz tiene dos conectores de tipo *weather_connector*. El modelo determina la radiación realmente concentrada por los espejos mediante la evaluación del ángulo de incidencia y diferentes factores de pérdidas, unidos en un único factor de eficiencia. Las siguientes ecuaciones constituyen el modelo:

$$weather\_connector2.DNI = \eta \cdot weather\_connector1.DNI \cdot \frac{Ap}{N_{SCE} \cdot N_{HCE}} \quad \text{Ec. (6.1)}$$

Dónde:

weather_connector2.DNI: radiación normal directa concentrada por los espejos, en W.
weather_connector1.DNI: radiación normal directa incidente en los espejos, en $W \cdot m^{-2}$.
A_p: Área reflectora del colector, en m^2. Se toma del *record collector*, véase Tabla 6.1.
N_{SCE}: número de SCEs por colector. Se toma del record *collector*, véase Tabla 6.1.
N_{HCE}: número de HCEs por SCE. Se toma del record *collector*, véase Tabla 6.1.
η: factor de eficiencia de los espejos, adimensional.

La Ec. (5.10) se define para que el DNI en el conector meteorológico 2 sea el flujo de energía concentrado que incide contra un único HCE. Por ello las unidades entre el DNI del conector meteorológico 1 y el conector meteorológico 2 difieren.

Se define el factor de eficiencia de los espejos como:

$$\eta = cos\theta \cdot IAM \cdot RowShadow \cdot EndLoss \cdot TrackEff \cdot GeomEffects \cdot MirrorReflectance \cdot OptiEff \quad \text{Ec. (6.2)}$$

Dónde:

η: factor de eficiencia de los espejos, adimensional.
θ: ángulo de incidencia, en grados. Es el ángulo que forma la normal de la parábola de espejos con el rayo de incidencia solar.

IAM: Modificador del ángulo de incidencia, adimensional. Es característico a cada tipo de colector.

RowShadow: factor de pérdidas por sombras entre filas, adimensional.

EndLoss: factor de pérdidas por pérdida de radiación concentrada en los extremos de los tubos.

TrackEff: eficiencia de enfoque, adimensional. Se toma del record *collector*, véase Tabla 6.2.

GeomEffects: efectos geométricos, adimensional. Se toma del record *collector*, véase Tabla 6.2.

MirrorReflectance: reflectividad de los espejos, adimensional. Se toma del record *collector*, véase Tabla 6.2.

OptiEff: eficiencia óptica general, adimensional. Se toma del record *collector*, véase Tabla 6.2.

A continuación las ecuaciones constituyentes de las variables de la Ec. (6.2):

- ✓ Ángulo de incidencia

$$\cos\theta = \sqrt{\cos^2\theta_z + \cos^2\delta \sin^2\omega} \qquad \text{Ec. (6.3)}$$

Dónde:

θ: ángulo de incidencia, en grados.

θ_z: ángulo cenital, en grados. Se toma del conector *weather_connector1*.

δ: ángulo de declinación, en grados. Se toma del conector *weather_connector2*.

Para ver la definición de los demás ángulos solares, véase Figura 5.18.

- ✓ Modificador el ángulo de incidencia (IAM)

Este IAM cuantifica empíricamente para cada colector otras pérdidas en los espejos que pueden ser correlacionadas con el ángulo de incidencia (θ). Estas pérdidas ocurren

debido a la reflexión y absorción por la camisa de vidrio del HCE cuando el ángulo de incidencia crece [9].

$$IAM = IAM_0 + IAM_1 \cdot \theta + IAM_2 \cdot \theta^2 \qquad \text{Ec. (6.4)}$$

Dónde:

IAM: modificador del ángulo de incidencia, adimensional.

IAM_0: factor de orden cero, adimensional. Se toma del record *collector*, véase Tabla 6.2.

IAM_1: factor de orden uno, en grados^{-1}. Se toma del record *collector*, véase Tabla 6.2.

IAM_2: factor de orden dos, en grados^{-2}. Se toma del record *collector*, véase Tabla 6.2.

θ: ángulo de incidencia, en grados.

- ✓ Factor de pérdidas por sombras entre filas (*RowShadow*)

Este factor de pérdidas cuantifica el efecto de sombreado entre filas cuando la altura solar (α) es lo suficientemente pequeño.

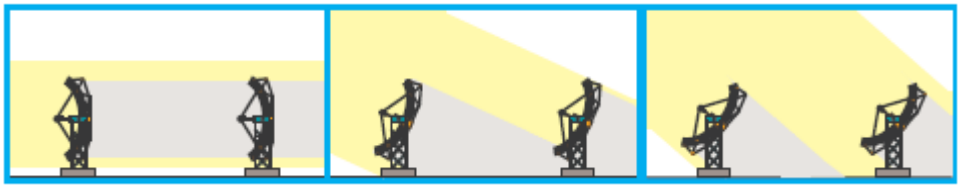


Figura 6.5: Ilustración del efecto de sombra entre filas

$$RowShadow = \begin{cases} 0, & W_{eff} \leq 0 \\ \dfrac{W_{eff}}{W}, & W_{eff} \geq W \\ 1, & W_{eff} < W \end{cases} \qquad \text{Ec. (6.5)}$$

Dónde:

RowShadow: factor de pérdidas por sombras entre filas, adimensional.

W_{eff}: Anchura reflectora eficaz no sombreada, en m.

W: Anchura reflectora total, en m. Se toma del record *collector*, véase Tabla 6.1.

- ✓ Anchura reflectora eficaz no sombreada (W_{eff}), [51]

$$W_{eff} = L_{row} \frac{\cos \theta_z}{\cos \theta} \qquad \text{Ec. (6.6)}$$

Dónde:

W_{eff}: Anchura reflectora eficaz no sombreada, en m.
L_{row}: Distancia entre filas de colectores. Se toma del record *collector*, véase Tabla 6.1.
θ_z: ángulo cenital, en grados. Se toma del conector *weather_connector1*.
θ: ángulo de incidencia, en grados.

- ✓ Factor de pérdidas por pérdida de radiación concentrada en los extremos de los tubos (*EndLoss*).

Estas pérdidas ocurren siempre que haya ángulos de incidencia distintos de cero. Cierta longitud de HCE no es iluminada por la radiación solar reflejada por los espejos [52].

$$EndLoss = 1 - f \cdot \frac{\tan \theta}{L_{SCA}} N_{SCE} \cdot N_{HCE} \qquad \text{Ec. (6.7)}$$

Dónde:

EndLoss: Factor de pérdidas por pérdida de radiación concentrada en los extremos de los tubos,
f: distancia focal media, en m. Se toma del record *collector*, véase Tabla 6.1.
θ: ángulo de incidencia, en grados.
L_{SCA}: longitud del colenctor, en m. Se toma del record *collector*, véase Tabla 6.1.
N_{SCE}: número de SCEs por colector. Se toma del record *collector*, véase Tabla 6.1.
N_{HCE}: número de HCEs por SCE. Se toma del record *collector*, véase Tabla 6.1.

6.4.2. Elemento colector de energía térmica (*HCE*)

El elemento colector de energía térmica (*HCE*) se modela mediante composición de modelos más simples de la sub-librería *basics* (véase apartado 4).

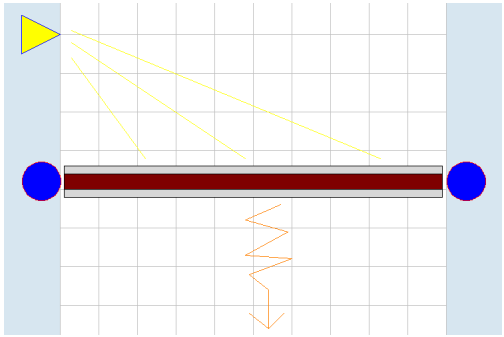


Figura 6.6: Icono del modelo *HCE*

Tal como se muestra en la Figura 6.6, el modelo tiene como interfaz a un conector meteorológico (*weather_connector*) y a dos conectores termohidráulicos (*hfPort*). Además se llama a la clase *record receiver*, que es reemplazable y define el tipo de HCE comercial utilizado.

La composición interna del modelo se muestra en la siguiente figura:

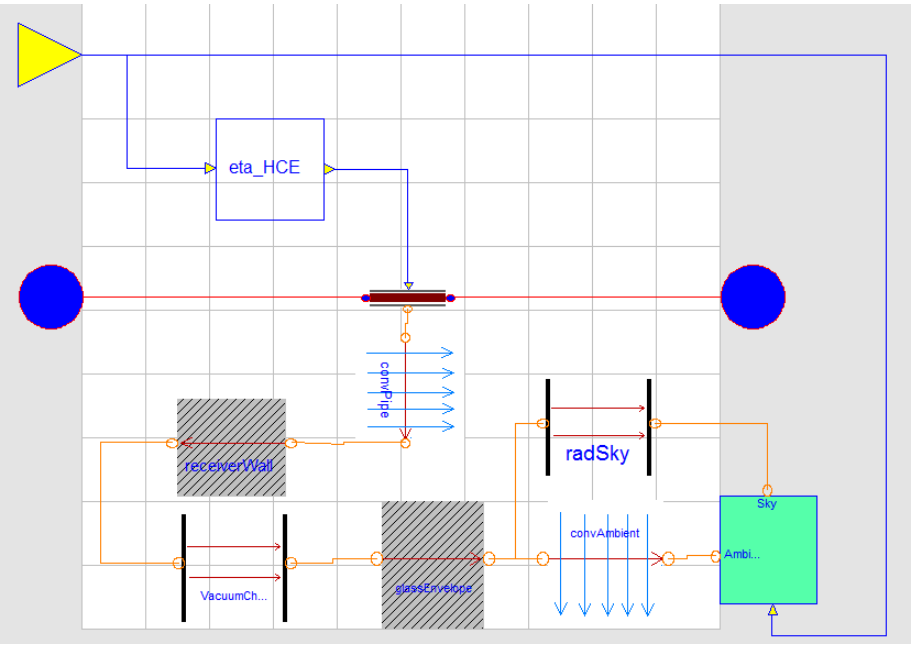


Figura 6.7: Modelado por composición del modelo *HCE*

La Figura 6.7 muestra que el HCE está formado por tubo absorbedor (redeclaración del modelo *pipe_heat_rad*, véase apartado 4.7.6), un factor de eficiencia de la radiación (redeclaración del modelo *IrradiationEfficiency*, véase apartado 4.5.6) y una sucesión de mecanismos de transmisión de calor: convección en la tubería (redeclaración del modelo *h_cylinder*, véase apartado 4.6.2), conducción por las paredes del tubo HCE (redeclaración del modelo *k_annulus*, véase apartado 4.6.1), radiación por la cámara de vacío del HCE (redeclaración del modelo *r_annulus*, véase apartado 4.6.3), conducción por las paredes de la camisa de vidrio (redeclaración del modelo *k_annulus*, véase apartado 4.6.1), convección hacia el ambiente (redeclaración del modelo *h_cylinder*, véase apartado 4.6.2) y radiación hacia el cielo (redeclaración del modelo *r_cylinder*, véase apartado 4.6.4). Esta transmisión de calor se realiza entre el tubo absorbedor y el ambiente estimando una transmisión unidimensional [53]:

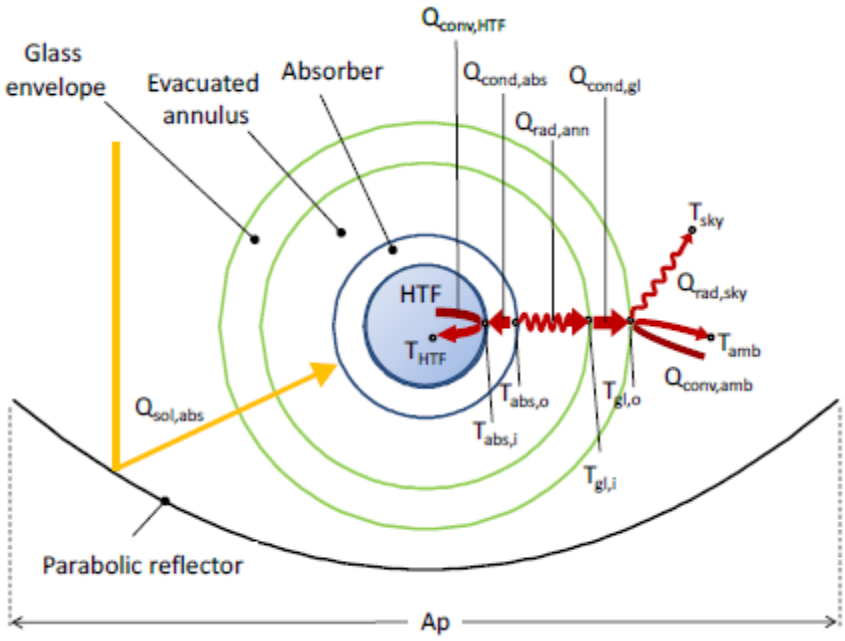


Figura 6.8: Mecanismo de transmisión de calor unidimensional del HCE

El ambiente que actúa como sumidero térmico en la Figura 6.8 y que se muestra en la Figura 6.7 se modela mediante redeclaración del modelo *Ambient* (véase apartado 4.5.5) y se conecta mediante su conector meteorológico al conector meteorológico de jerarquía superior del modelo *HCE*, ambos del tipo *weather_connector*, por lo que las se puede introducir las variables temperatura ambiente (*t*) y velocidad de viento (*ws*) directamente de los datos meteorológicos (véase apartado 5.4).

Todos los modelos redeclarados y que se utilizan en la composición utilizan las propiedades del HCE reflejadas en la Tabla 6.3 y Tabla 6.4, llamadas por la clase record *receiver* y las se definen los coeficientes de transporte de los fenómenos de transmisión de calor para particularizarlos al sistema [53].

Se define el factor de eficiencia del HCE mediante la siguiente ecuación, para completar el modelo de composición mostrado en la Figura 6.7:

✓ Factor de eficiencia del HCE

$$\eta_{HCE} = HCEdust \cdot BelShad \cdot EnvTrans \cdot HCEabs \cdot HCEmisc \qquad \text{Ec. (6.8)}$$

Dónde:

η_{HCE}: factor de eficiencia del HCE, adimensional. Corresponde al factor de eficiencia del modelo *HCEEfficiency* (redeclaración del modelo *IrradiationEfficiency* del apartado 4.5.6).

HCEdust: factor de polvo en HCE, adimensional. Se toma del record *receiver*, véase Tabla 6.4.

BelShad: factor de sombra por fuelle, adimensional. Se toma del record *receiver*, véase Tabla 6.4.

EnvTrans: transmitancia de la camisa de vidrio, adimensional. Se toma del record *receiver*, véase Tabla 6.4.

HCEabs: absorbancia de la capa de pintura del tubo de acero, adimensional. Se toma del record *receiver*, véase Tabla 6.4.

HCEmisc: factor misceláneo para contar otras pérdidas, adimensional. Se toma del record *receiver*, véase Tabla 6.4.

6.4.3. Elemento de colector solar (*SCE*)

Este modelo simula los elementos estructurales en los que se divide un colector solar o SCA.

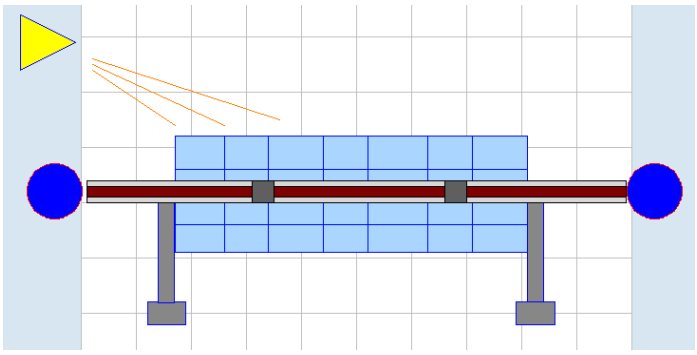


Figura 6.9: Icono del modelo *SCE*

Es enteramente definido mediante composición gráfica:

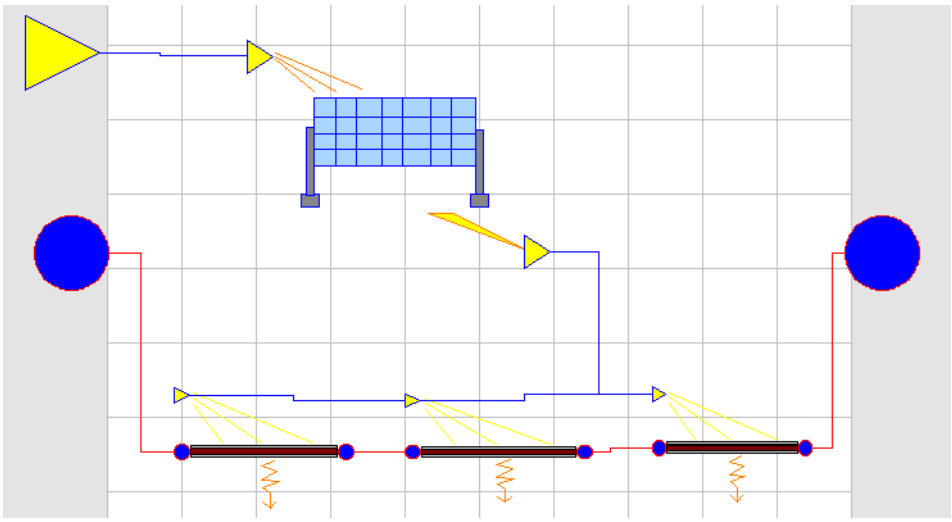


Figura 6.10: Modelado por composición del modelo *SCE*

Como se muestra en la Figura 6.10, el sistema está compuesto por las conexiones de un modelo *Mirrors* (véase apartado 6.4.1) y tres modelos *HCE* (véase apartado 6.4.2) conectados en serie.

6.4.4. Colector solar (*SCA*)

El modelo *SCA* llama a la clase record *collector* y dependiendo de la constante del número de SCEs por colector (N_{SCE}), se compone el modelo conectando en serie modelos *SCE*s mediante recurrencia con una cláusula *for*.

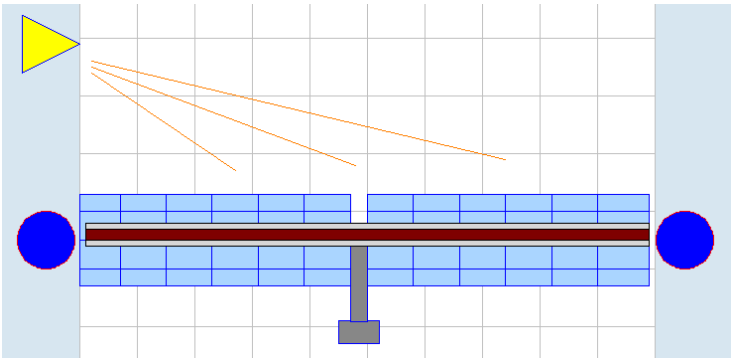


Figura 6.11: Icono del modelo *SCA*

Se define la temperatura central del SCA, que es donde el objeto físico real tiene un transmisor de temperatura.

Como se ha comentado anteriormente, los SCAs son elementos que tienen movimientos independientes unos de otros ya que poseen una unidad hidráulica de pistones para permitir su movimiento y enfoque al sol.

6.4.5. Lazo solar (*Loop*)

El lazo solar consigue el salto de temperatura requerido por la planta, que suele ser de aproximadamente 100 K entre entrada al lazo y salida, en régimen nominal.

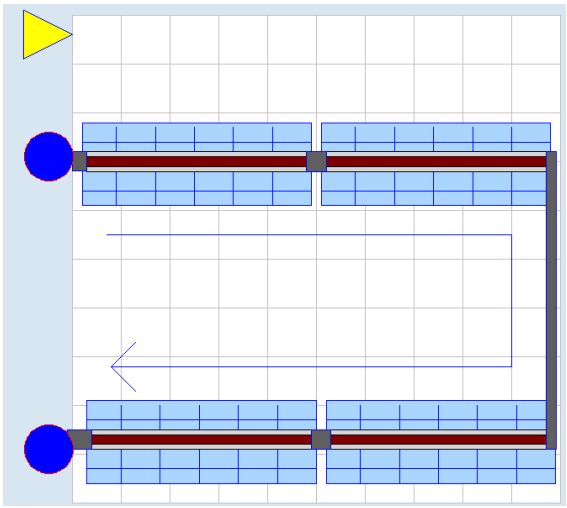


Figura 6.12: Icono del modelo *Loop*

La composición interna del modelo de la Figura 6.12 se muestra a continuación:

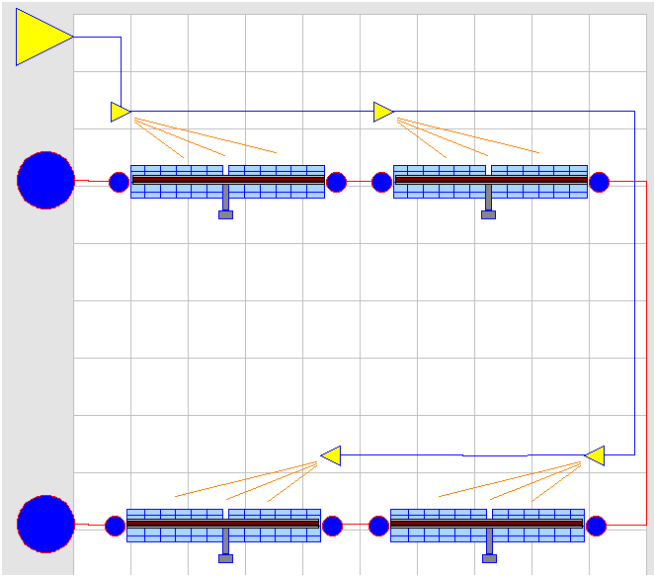


Figura 6.13: Modelado por composición del modelo *Loop*

Tal como se muestra en la Figura 6.13, el modelo del lazo solar (*Loop*) se obtiene mediante composición gráfica de cuatros SCAs en serie y conectando entre sí todos los conectores de tipo *weather_connector*.

Adicionalmente se definen las siguientes variables en el modelo del lazo:

- ✓ Temperatura de entrada al lazo (t_{in}): es igual a la temperatura del conector *hfPort1*.
- ✓ Temperatura de salida del lazo (t_{out}): es igual a la temperatura del conector *hfPort2*.
- ✓ Potencia térmica (*ThermalPower*):

$$ThermalPower = m \cdot (h_{out} - h_{in}) \qquad \text{Ec. (6.9)}$$

Dónde:

ThermalPower: potencia térmica conseguida por el lazo, en W.

m: caudal másico a través del lazo, en kg/s.

h_{out}: entalpía específica del fluido térmico a la salida del lazo, en W/kg.

h_{in}: entalpía específica del fluido térmico a la entrada del lazo, en W/kg.

Las entalpías específicas son determinadas dependiendo de la sustancia que fluye a través del lazo, definida en última instancia en el modelo *pipe_heat_rad* dentro del modelo *HCE*.

- ✓ Energía acumulada (*E*):

$$E = \int_0^{time} ThermalPower\ dt \qquad \text{Ec. (6.10)}$$

Dónde:

E: energía acumulada.
ThermalPower: potencia térmica conseguida por el lazo, en W.

6.4.6. Campo solar (*SolarField*)

El campo solar es la unión de los lazos necesarios para conseguir la potencia térmica nominal necesaria para el funcionamiento de la planta solar. Este modelo llama a tres clases reemplazables: *Medium*, *collector y receiver*, que permite conjugar distintos tipos de fluido térmico, colectores y HCEs, respectivamente.

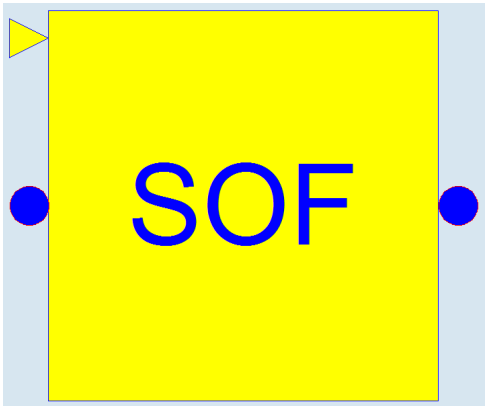


Figura 6.14: Icono del modelo *SolarField*

El número de lazos en un campo solar típico suele ser de más de 100. No se modela por composición de lazos (*Loop*) debido a que el modelo sería muy pesado computacionalmente para la información verdaderamente útil que se suele buscar en la simulación. Por ello se modela un campo solar simplificado al crearlo como un modelo atómico con el número de lazos (*N*) como parámetro y que comprende todas las ecuaciones correspondientes a la consecución de un lazo (*loop*) de los apartados anteriores con la salvedad de la definición del factor de rendimiento del campo solar:

$$\eta_{SOF} = \eta' \cdot CF - \frac{Q_{loss}}{DNI \cdot f} \qquad \text{Ec. (6.11)}$$

Dónde:

η': factor de eficiencia conjunto de espejos y HCEs, adimensional. Es el producto de ambos factores de eficiencia (η y η_{HCE}).

CF: factor de limpieza (*Cleanliness Factor*). Indica en tanto por uno el grado de limpieza medio de los espejos del campo solar, que perjudica a la reflectividad del espejo (*MirrReflectance*).

Q_{loss}: pérdidas de calor en los tubos por convección y radiación, en $W \cdot m^{-1}$. Se toma del record *receiver*, véase Tabla 6.4.

f: anchura reflectora del colector. Se toma del record *collector*, véase la Tabla 6.1.

DNI: radiación normal directa, en $W \cdot m^{-2}$. Se toma del conector *weather_connector*.

6.4.7. Campo solar con control de temperatura (*SolarField_tControl*)

Este modelo se crea totalmente por composición gráfica a partir de otros modelos.

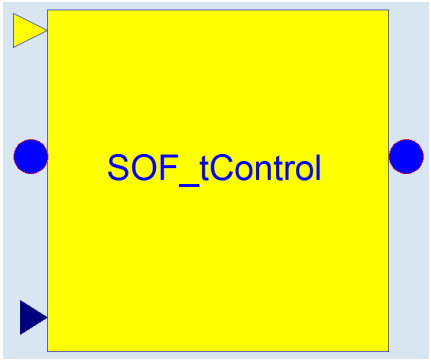


Figura 6.15: Icono del modelo *SolarField_tControl*

Tal como se muestra en la Figura 6.15, el icono es básicamente el mismo que el del modelo *SolarField* (Figura 6.14) pero con la adición de un conector de tipo *analog_input*, por el que se introduce el punto de consigna de temperatura de salida del campo solar. La estructura interna de composición del modelo se ilustra en la siguiente figura:

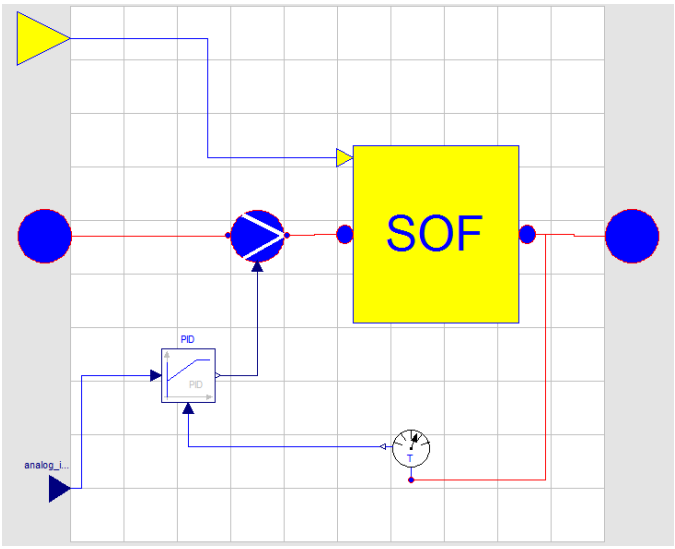


Figura 6.16: Modelado por composición del modelo *SolarField_tControl*

En la Figura 6.15 se muestran los modelos individuales de los que está formado *SolarField_tControl*, junto con las conexiones de sus interfaces. La disposición es de un sistema de control típico por retroalimentación. Se utiliza el modelo *SolarField* (véase apartado 6.4.6) como el sistema a controlar, el modelo *temperatura_sensor* (véase apartado 4.8) como elemento primario del lazo de control, el modelo *LimPID_indirect* (véase apartado 4.9) como controlador y el modelo *pump* (véase apartado 4.7.7) como elemento final de control [54].

El control de temperatura del campo solar es un asunto clave en las plantas termosolares ya que el fluido térmico (usualmente de tipo *DowthermA*) está limitado operacionalmente ya que a cierta temperatura la degradación del fluido aumenta exponencialmente causando el empeoramiento de sus propiedades térmicas y la formación de productos volátiles que puede poner en peligro las bombas del sistema por problemas de cavitación.

6.5.Particularización de los modelos (*HCEs, SCEs, SCAs, Loops*)

Las sub-librerías de *Mirrors, HCEs*, *SCEs, SCAs* y *Loops* continen la particularización de los modelos para crear modelos de los diferentes equipos comerciales que actualmente existen en el mercado.

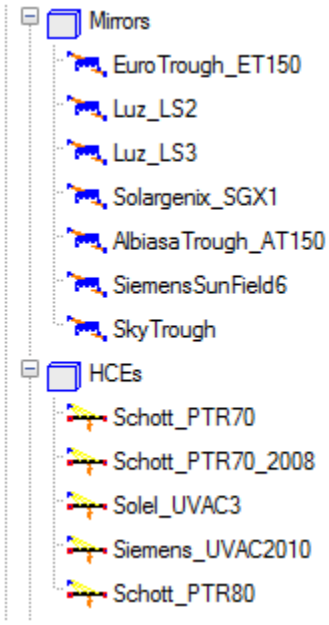


Figura 6.17: Estructura de las sub-librerías *Mirrors* y *HCEs*

En la Figura 6.17 se muestran los diferentes modelos particularizados de espejos y HCEs. Todos los modelos son extensiones de los modelos básicos *Mirrors* (véase apartado 6.4.1) y *HCE* (véase apartado 6.4.2), respectivamente. Se llaman a las clases records que almacenan las propiedades de los colectores (Tabla 6.1 y Tabla 6.2) y las propiedades de los HCEs (Tabla 6.3 y Tabla 6.4).

Las posibles combinaciones entre tipo de estructura de espejos y tipo de HCE resultan en 40 diferentes tipos de SCEs y SCAs, modelos almacenados en las sub-librerías homónimas. Los modelos se nombran por el código de la estructura de espejos, seguido del código del HCE que monta. Así, por ejemplo, el modelo de SCA de nombre *ET150_UVAC3* indica que el colector monta una estructura de espejos de tipo *EuroTrough_ET150* y HCEs de tipo *Solel_UVAC3*.

Finalmente en la sub-librería *Loops* se particularizan los lazos solares (extensión del modelo *loop*, apartado 6.4.5). Para simplificar la librería, se ha optado por componer los diferentes lazos para que monten todos los SCAs del mismo tipo y que el HCE sea siempre el tipo *Schott_PTR70*. Se deja al usuario libertad para componer todas las demás posibilidades. Estas son grandes puesto que un lazo puede montar diferentes SCAs y estos también combinaciones de diferentes tipos de HCE.

6.6.Ejemplos (*Examples*)

Se plantean en esta sub-librería cuatro tipos de ejemplos: uno que permite comparar el rendimiento de diferentes tipos de lazos (*Loop_comparison*), otro que ilustra la operación normal de un lazo individual (*Loop_operation*), otro que permite ajustar el control de temperatura del campo solar (*SOF_tempControl*) y finalmente otro que permite comparar la operación del campo solar en diferentes localizaciones del mundo (*SOF_location_comparison*).

6.6.1. Comparación de lazos (*Loop_comparison*)

El ejemplo se compone gráficamente utilizando el modelo *Sun* con datos de un día completamente soleado (véase apartado 5.5.2) y siete diferentes modelos de lazo, correspondientes a los siete modelos de colectores comerciales existentes en *ParaTrough*. A todos los lazos se les somete a las mismas condiciones atmosféricas (todos se conectan al mismo modelo solar), de caudal (8 kg/s) y temperatura de entrada (293^{o}C) mediante la utilización de modelos *source_m_t* (véase apartado 4.7.1).

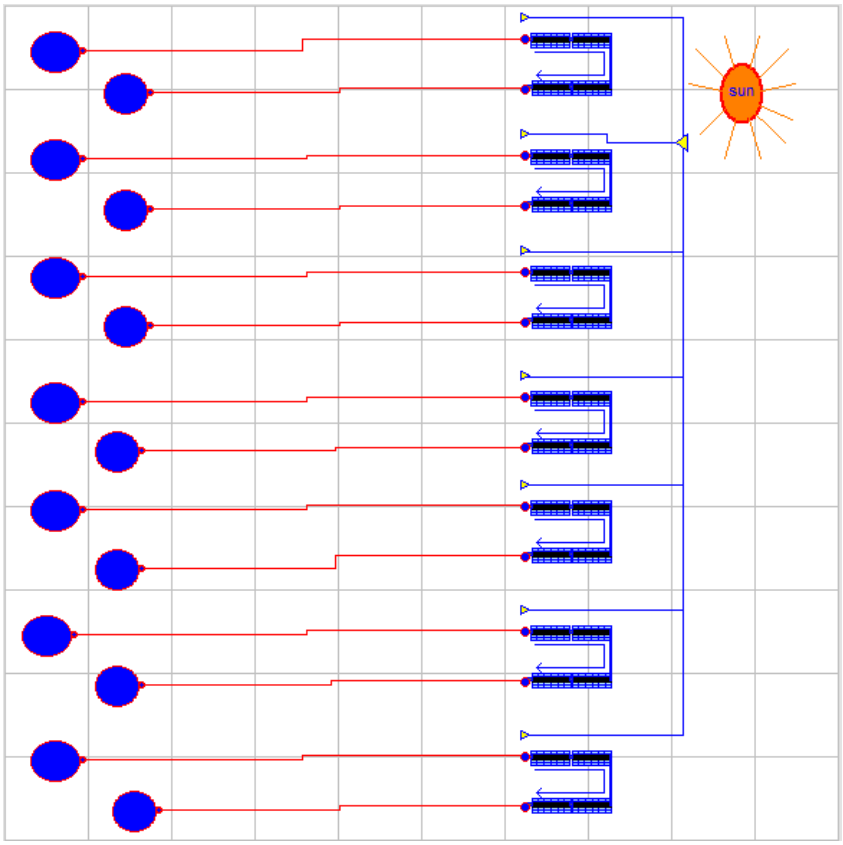


Figura 6.18: Composición gráfica del ejemplo *Loop_comparison*

La simulación del ejemplo arroja los siguientes resultados:

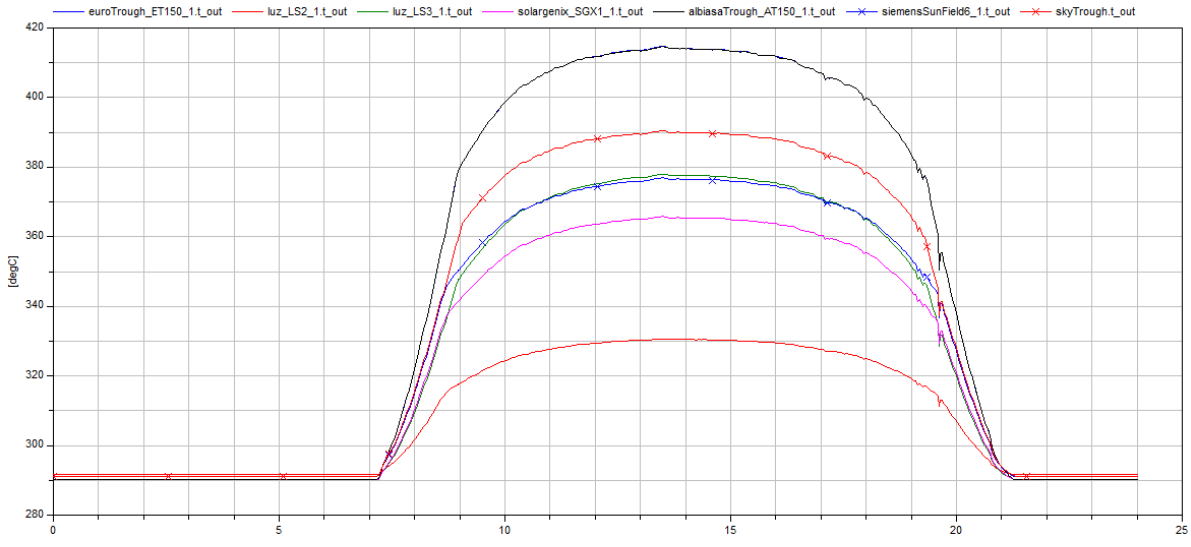


Figura 6.19: Comparativa de la temperatura de salida de diferentes lazos

La Figura 6.19 muestra la temperatura de salida de los diferentes lazos en comparación, que puesto que se aplican idénticas condiciones de caudal y recurso

solar a cada uno de ellos, se relaciona directamente con su rendimiento térmico. De esta manera, el lazo más eficiente es el compuesto por el tipo de colector *EuroTrough_ET150*, seguido desde muy cerca y empatando prácticamente con el tipo *AlbiasaTrough_AT150*. Por el contrario, se observa que el tipo *Luz_LS2* es el que peor rendimiento tiene aunque hay que decir que es el primero que fue desarrollado y todos los demás avanzaron sobre lo conseguido por este primer colector.

Este ejemplo se puede reutilizar para comparar diferentes configuraciones de lazos como por ejemplo uno formado por los dos primeros SCAs del tipo *EuroTrough* y los dos últimos del tipo *Albiasa* pero montando HCEs de tipo *Solel* en lugar de de tipo *Schott*. E incluso cambiando el fluido térmico *DowthermA* que circula a través del lazo por la sal solar *HitecSolarSalt*. Todas estas combinaciones se las deja al usuario para que interactúe con *ParaTrough*.

6.6.2. Operación de un lazo (*Loop_operation*)

Este ejemplo está destinado a estudiar las variables que componen todos los componentes de un lazo en una operación de un día soleado.

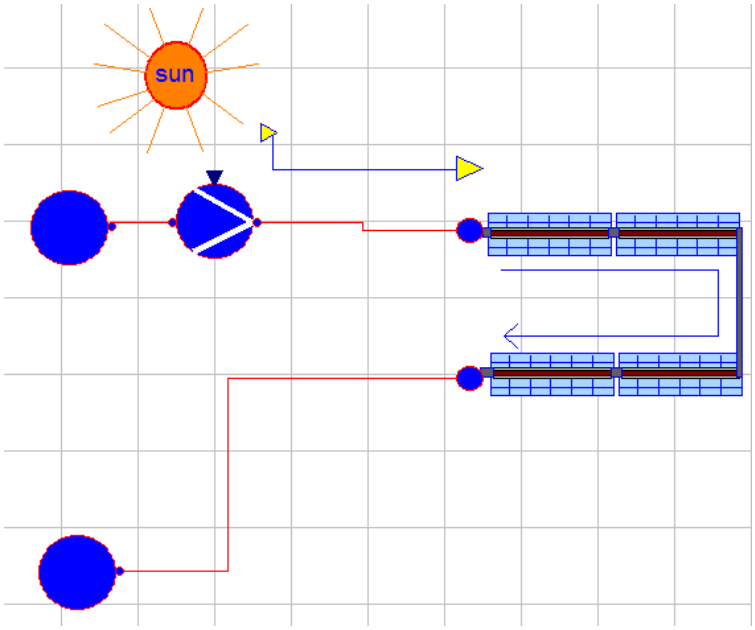


Figura 6.20: Composición del ejemplo *Loop_operation*

Tal como se muestra en la Figura 6.20, el ejemplo se compone gráficamente de un modelo *loop* (véase apartado 6.4.5), un modelo *pump* (véase apartado 4.7.7), un modelo *sun* (véase apartado 5.5.2), un modelo *source_t* (véase apartado 4.7.2) y un modelo *sink_p* (véase apartado 4.7.3). El objetivo de la composición es bombear un caudal constante a un lazo (8 kg/s) en un día plenamente soleado.

La simulación arroja los siguientes resultados:

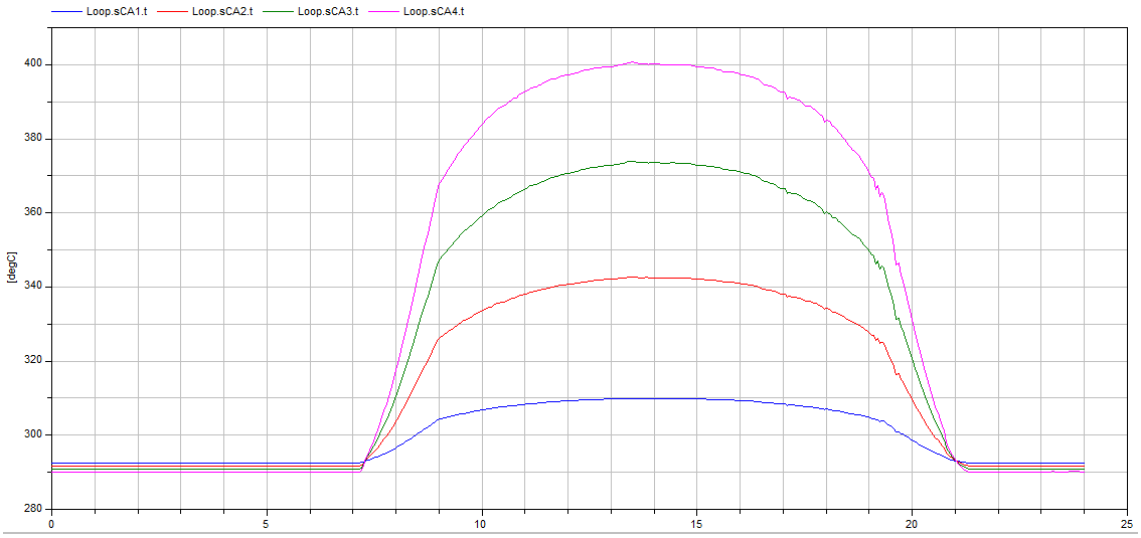


Figura 6.21: Temperaturas centrales de los SCAs que forman el lazo

La Figura 6.21 muestra cómo varían las temperaturas centrales de los SCAs que forman el lazo en el día soleado, pudiéndose observar su aumento en las horas de salida de la radiación y su disminución por la tarde, cuando la radiación decae. También es interesante observar que por la noche, al seguir recibiendo el lazo el mismo caudal que por el día, las temperaturas de los SCAs se invierten siendo el primero (SCA1) el más caliente y el último (SCA4) el más frío. Este efecto es debido al enfriamiento del fluido térmico por la noche debido a la ausencia de radiación y a las pérdidas térmicas a través de los HCEs.

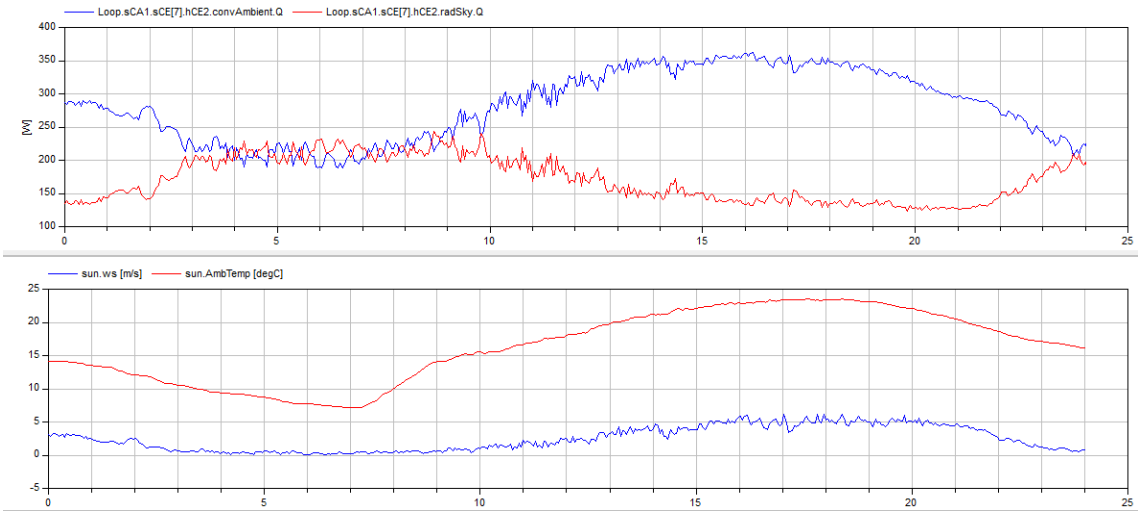


Figura 6.22: Pérdidas térmicas en un HCE y su correlación con las condiciones meteorológicas

La Figura 6.22 muestra las pérdidas térmicas al ambiente en un HCE aleatorio del lazo tanto por convección (línea azul en la gráfica superior) como por radiación (línea roja en la gráfica superior). Se observa que estos fenómenos están directamente correlacionados a los cambios de temperatura ambiente (su aumento durante el día hace que las pérdidas por radiación disminuyan) y de velocidad de viento (su aumento durante el día hace que las pérdidas por convección aumenten).

Se deja al usuario para que reutilice el ejemplo un tipo de día nublado y analice las variables involucradas en los distintos componentes del lazo.

6.6.3. Control de temperatura en el campo solar (*SOF_tempControl*)

Si se observan con más detenimiento la Figura 6.19 y la Figura 6.20, se observará que la temperatura crece sin control hasta lo que el lazo es capaz de alcanzar al caudal suministrado. En algunos puntos de estas figuras se alcanzan valores superiores a 400^{o}C, que es el punto en el que el fluido térmico *DowthermA* empieza a degradarse exponencialmente. Para evitar esta degradación, es necesario dotar al campo solar y a los lazos de mecanismos de control para la regulación del caudal en concordancia. Si se alcanza una temperatura próxima a 400^{o}C, el sistema de control aumentará el caudal y si la temperatura baja demasiado como para comprometer el rendimiento de la planta termosolar, el sistema de control disminuirá el caudal.

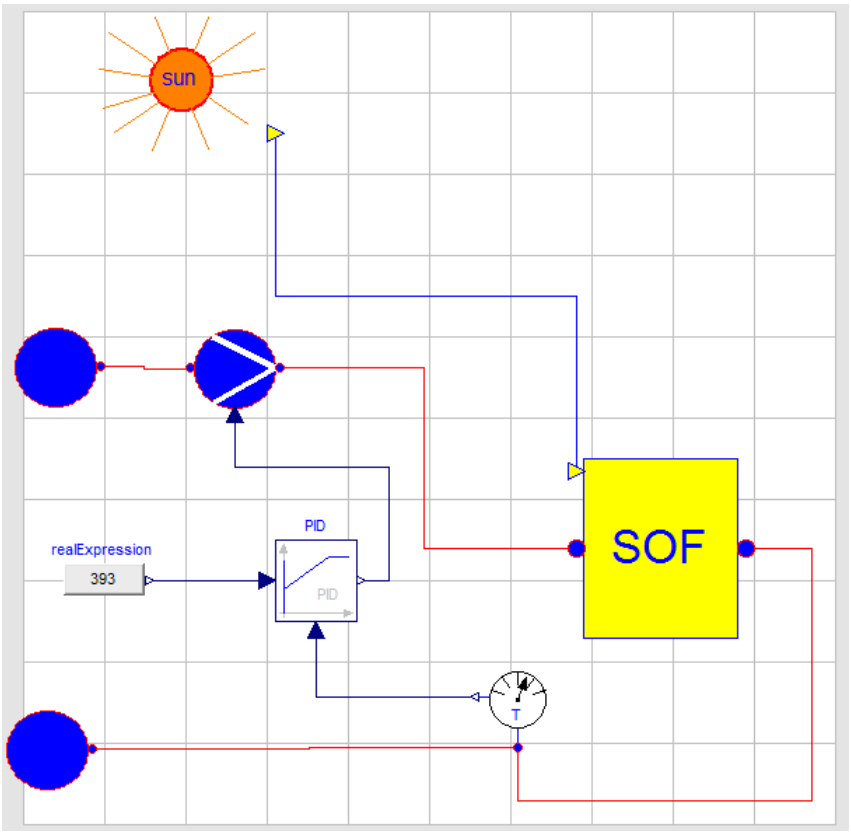


Figura 6.23: Composición gráfica del ejemplo *SOF_TempControl*

La composición gráfica del ejemplo mostrada en la Figura 6.23 es esencialmente la misma que la del modelo *SolarField_tControl* (véase apartado 6.4.7). Se establece un lazo de control que actúa sobre el caudal suministrado al campo solar por la bomba para controlar la temperatura de salida del mismo y que toma como punto de consigna un valor parametrizable de temperatura. Lo más usual en las plantas termosolares es establecer este punto de consigna en 393^{o}C (1).

La simulación arroja los siguientes resultados:

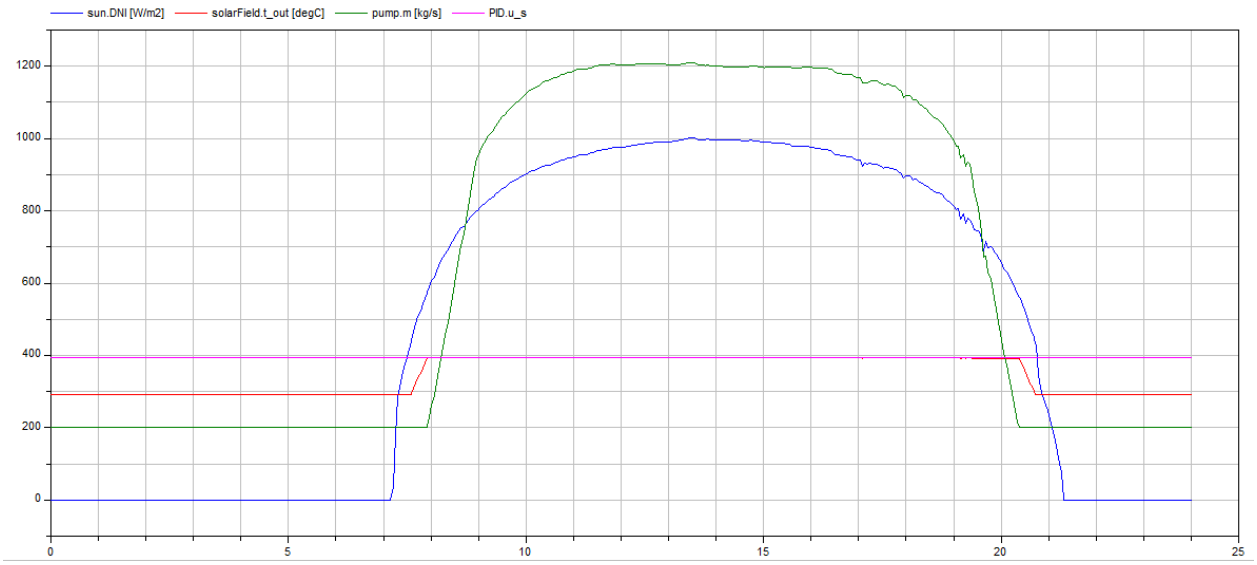


Figura 6.24: Control de temperatura del campo solar en un día soleado

La operación mostrada en la Figura 6.24 se ha conseguido con una sintonía del controlador PID de 25 para la ganancia, 0.05 para la constante de tiempo integral y la acción derivativa anulada [55]. Se observa la respuesta gradual de la bomba a medida que la radiación suministra más energía al campo solar. La temperatura de salida del campo solar (línea roja) permanece básicamente constante e igual al punto de consigna en el periodo de radiación, salvo al final de la tarde, en la que el decaimiento de la radiación unido a perturbaciones de pequeñas nubes (pequeños altibajos de la línea azul de DNI por la tarde) hace que la temperatura de salida del campo solar permanezca con un *offset* con respecto al punto de consigna durante un periodo de tiempo (aproximadamente una hora desde las 19:15 a las 20:15).

A continuación se repite el ejemplo pero utilizando un día parcialmente nublado como datos meteorológicos:

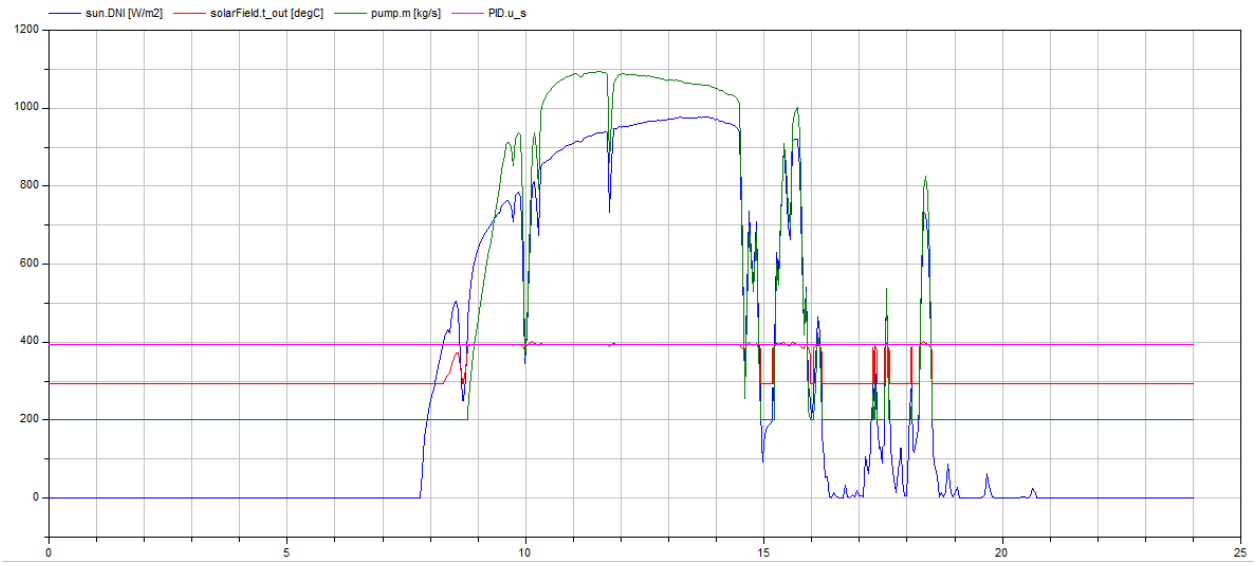


Figura 6.25: Control de temperatura del campo solar en un día parcialmente nublado

Se observa en la Figura 6.25 que el control de temperatura en este tipo de días resulta más complicado, resultando lógicamente imposible cuando la ausencia de radiación es total. Sin embargo, se consigue un resultado aceptable.

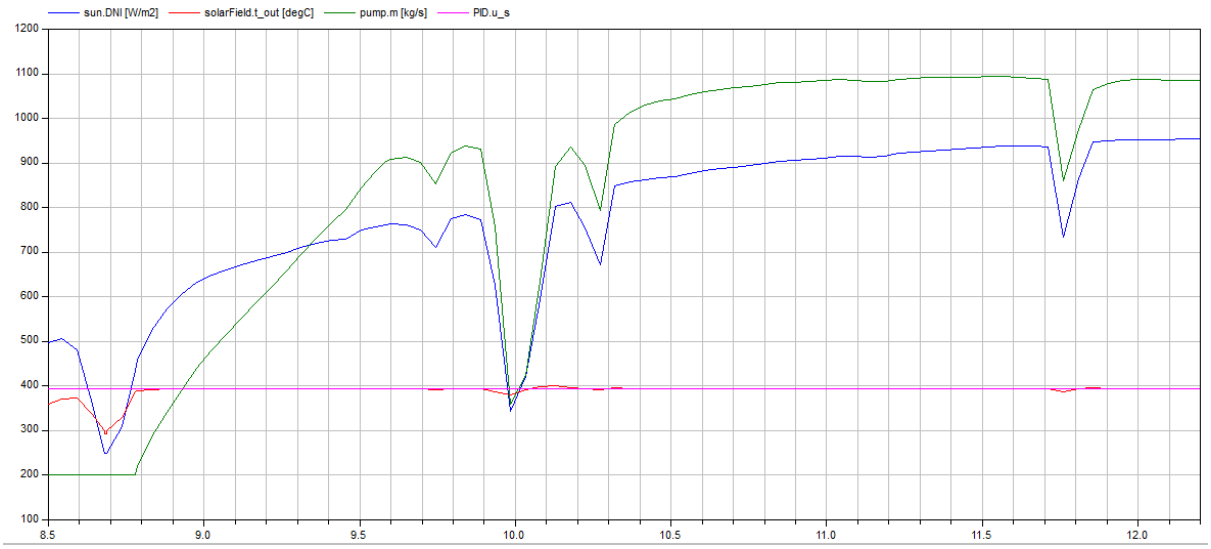


Figura 6.26: Detalle del control de temperatura del campo solar en presencia de nubes

En el detalle que constituye la Figura 6.26 se observan algunos sobrepasajes de la temperatura de salida de campo solar cuando aparecen nubes de considerable envergadura. Se deja al usuario la optimización del controlador para encontrar los parámetros de sintonía más adecuados para un día soleado, un día parcialmente

nublado y para mejorar rampas de arranque en la mañana y rampas de parada en la tarde.

6.6.4. Comparación de campo solar entre localizaciones (*SOF_location_comparison*)

Este ejemplo está destinado para comparar energía térmica de un campo solar de las mismas características bajo diferentes condiciones meteorológicas de localizaciones diferentes.

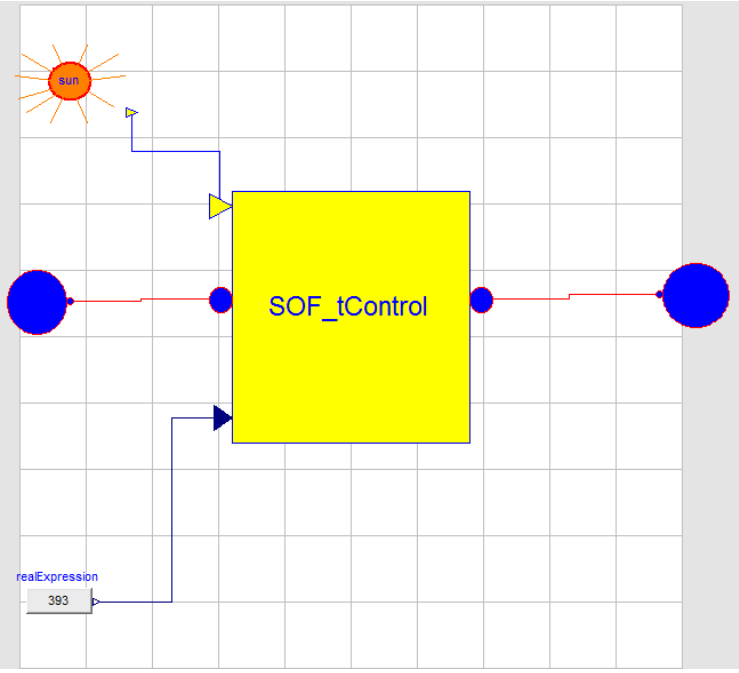


Figura 6.27: Composición gráfica del ejemplo *SOF_location_comparison*

Tal como se ilustra en la Figura 6.27, el ejemplo está compuesto gráficamente de un modelo *SolarField_tControl* (véase apartado 6.4.7), un modelo *Sun* (véase apartado 5.5.2), un modelo *Source_t* (véase apartado 4.7.2), un modelo *Sink_p* (véase apartado 4.7.3) y una entrada constante para el punto de consigna de la temperatura de salida del campo solar, ajustada a 393^{o}C.

La simulación se ha realizado a 3000 horas (para un periodo de un poco más de cuatro meses) desde el 1 de mayo para evaluar en los meses de mayor radiación de los TMY

de las localizaciones de Sevilla y Phoenix (Estados Unidos). Esta simulación arroja los siguientes resultados:

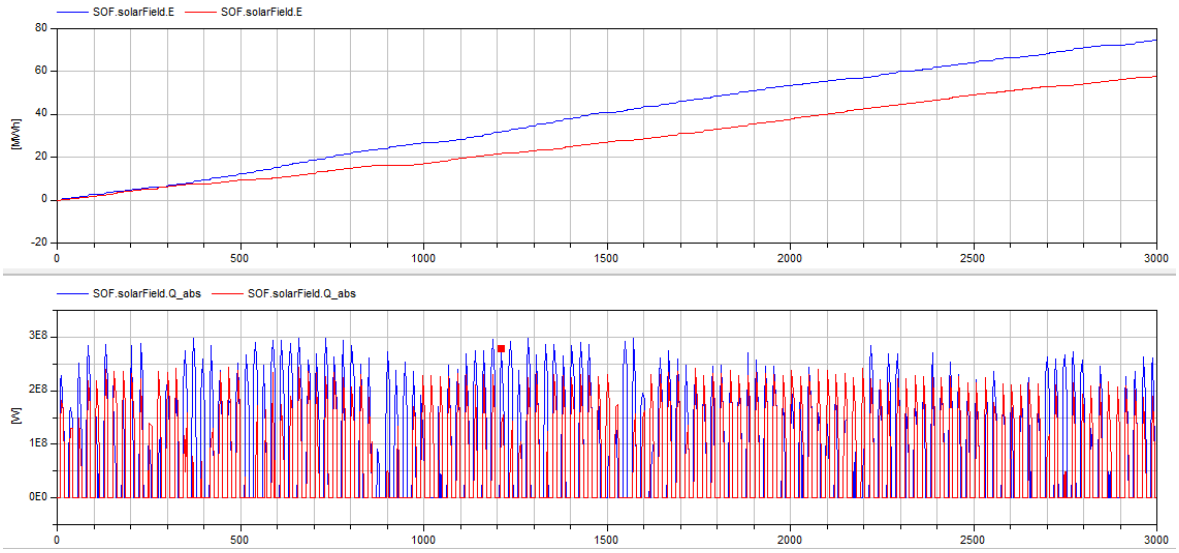


Figura 6.28: Energía acumulada y potencia del campo solar para Sevilla y Phoenix (Estados Unidos) durante el periodo de primavera-verano

Se observa en la Figura 6.28 que la localización más favorable para el campo solar específico del ejemplo (152 lazos de tipo *EuroTrough_ET150*, HCEs de tipo *Schott_PTR70* y fluido térmico *DowThermA*) es Phoenix (Esstados Unidos). La energía acumulada en los cuatro meses de primavera-verano es de 74.8 MWh mientras que en Sevilla se alcanza 57.9 MWh para el mismo periodo. Se deja al usuario que compare diferentes tipos de campo solar (modificando tipología de colector y HCE) en las diferentes localizaciones que incluye *ParaTrough* por defecto (véase apartado 5.4.2) e incluso añadiendo nuevas localizaciones desde SAM para profundizar en el estudio (véase Anexo B: Conversión de TMY a ParaTrough).

6.7.Futuras ampliaciones

En el futuro se trabajará en las siguientes líneas de ampliación: se programarán tubos colectores con ausencia de vacío para facilitar su detección en situaciones reales de planta, se mejorará el concepto de control de temperatura del lazo de campo solar para contemplar protecciones por alarma de alta temperatura y se ampliará el colector de campo solar al dominio mecánico, pudiendo estudiar los esfuerzos de compresión-tensión de los componentes ante los cambios de temperatura.

6.8.Conclusiones

Esta sub-librería actúa como motor térmico del sistema, pudiendo emplear el calor captado por el colector en suministrarlo al ciclo de potencia o al sistema de almacenamiento de sales fundidas. La variabilidad de modelos de diferentes marcas comerciales tanto de tubos absorbedores como de colectores solares, hace que existan numerosas posibilidades de modelado y estudio. Estas posibilidades se añaden a las ya numerosas provenientes de la librería solar.

7. Conclusiones y trabajos futuros

7.1. Introducción

En la librería *ParaTrough* se han modelado componentes para la determinación de los parámetros físico-químicos de las más importantes sustancias presentes en las plantas termosolares a través de la sub-librería *media*. Se han modelado componentes genéricos válidos y reutilizables tanto para las sub-librerías concernientes a los macrosistemas de las plantas termosolares como para otras librerías independientes través de la sub-librería *basics*. Y se han modelado componentes de dos de los macrosistemas más importantes de las plantas termosolares: el recurso solar (*A_SolarResource*) y el sistema del fluido de transferencia térmica (*B_HeatTransferFluid*).

7.2. Conclusiones

Con este trabajo, el ingeniero de procesos o analista de plantas y sistemas tiene una herramienta modular, fácil de utilizar, con flexibilidad, adaptabilidad y existencia de entornos de modelado gratuitos cuyo propósito es el de evaluar el rendimiento, detectar fallos, explorar nuevos modos de operación y optimizar el sistema de campo solar de una planta termosolar de colectores cilindro-parabólicos emplazada en cualquier parte del mundo. A continuación se analiza el cumplimiento de las tareas-objetivo definidas en el apartado 1.2:

Tabla 7.1: Tabla de cumplimiento de las tareas-objetivo en *ParaTrough*

Propósito para el ingeniero de procesos o analista de plantas	**Tareas-objetivo en *ParaTrough***	**Cumplimiento**
Para evaluación de rendimiento y detección de fallos	Asegurar una conectividad a datos públicos de recurso solar de diferentes localizaciones para asegurar que la librería se pueda utilizar en cualquier lugar del mundo y utilizando formatos meteorológiocos estándar.	**Sí**, conectividad con SAM utilizando formatos TMY e INTL
	Crear una base de datos típicos meteorológicos de diferentes localizaciones.	**Sí**, sub-librería *A_SolarResource.Meteo*
	Crear un modelo solar que calcule en cualquier momento y en cualquier localización terrestre la posición del sol.	**Sí**, el modelo *A_SolarResource.Models.Sun*
	Crear un modelo detallado de lazo solar para modelar la conversión de energía solar en energía térmica.	**Sí**, el modelo *B_HeatTransferFluid.Models.Loop*
	Crear un modelo simplificado de campo solar para simular plantas con numerosos lazos solares sin tener una carga computacional excesiva.	**Sí**, el modelo *B_HeatTransferFluid.Models.SolarField*
Para exploración de nuevos modos de operación y optimización de la planta	Crear una librería de los medios materiales más usados en las plantas termosolares, permitiendo su cómoda reutilización y sustitución en los modelos de los diferentes componentes.	**Sí**, la sub-librería *media*
	Crear una librería de sistemas básicos de flujo de fluidos e intercambio térmico reutilizables en todos los macrosistemas de las plantas termosolares.	**Sí**, la sub-librería *basics*
	Crear una base de datos de diferentes colectores solares y tubos absorbedores comerciales, para poder englobar las diferentes plantas existentes según la variabilidad de los fabricantes de los componentes.	**Sí**, la sub-librerías *B_HeatTransferFluid.Collectors* y *B_HeatTransferFluid.Receivers*
	Crear modelos por instanciación gráfica que tengan en cuenta el sistema de control.	**Sí**, el ejemplo *B_HeatTransferFluid.SOF_tempControl*

La funcionabilidad de la librería *ParaTrough* en su versión inicial queda limitada al modelado y simulación del campo solar (macrosistema que aúna el sistema de recurso solar y sistema de fluido de transferencia calorífica), que es el corazón de la planta termosolar.

7.3.Trabajos futuros

En futuros trabajos se programarán las sub-librerías ciclo de potencia (*C_PowerCycle*), almacenamiento térmico (*D_ThermalStorage*), sistemas auxiliarse (*E_Balance of Plant*) y planta térmica (*PowerPlant*) para completar todos los macrosistemas presentes en las plantas solares (véase Figura 1.2). En esta última se ensamblarán los macrosistemas para modelar el conjunto completo de una planta CSP en sus diferentes configuraciones típicas. La unión mediante reutilización de la librería de código abierto *Thermopower* (véase apartado 2.4) para contemplar el ciclo de potencia y de conversión en energía eléctrica también es una potencial opción de trabajo futuro como ampliación de *ParaTrough*. En las sub-librerías ya incluidas en *ParaTrough*, se trabajará en las siguientes líneas de mejora:

Tabla 7.2: Futuras líneas de mejora para *ParaTrough*

Sub-librería de *ParaTrough*	**Futuras líneas de mejora**
media	Definición de otros medios materiales usados en las plantas termosolares: nitrógeno, gas natural, diesel, ácido sulfúrico, hidróxido sódico, hipoclorito sódico,... Adaptación del modelo de chequeo de sustancias (*MediaChecking*) para fluidos compresibles.
basics	Programación de los modelos de intercambiador de calor, tanque, válvula,...
A_SolarResource	Mejora del sistema de adquisición de datos para importar datos directamente de estaciones meteorológicas reales.
B_HeatTransferFluid	Incluir el fenómeno de la pérdida de vacío y permeabilidad de hidrógeno en los tubos absorbedores HCE. Mejora del concepto de control de temperatura de los colectores solares. Ampliación del modelo del colector solar al dominio mecánico

Bibliografía

[1] **García Garrido, Santiago.** *Ingeniería de Centrales Termosolares CCP.* León : Ediciones Renovetec, 2010. pág. 37. ISBN: 978-84-614-4183-9.

[2] **García Casals, Xavier.** *SEGS in Barna: Análisis de Comportamiento de una Planta Termosolar de Colectores Cilindroparabólicos para Generación de Electricidad en Barcelona.* Barcelona : s.n., 2001. Vol. I. IIT-01-126I.

[3] **Rojas, Esther.** *Thermocline tanks: an option for the future?* Sevilla : Ciemat, 2014.

[4] **National Renewable Energy Laboratory.** *System Advisor Model. https://sam.nrel.gov/*

[5] **Selig, Martin.** *Commercial CSP plants based on Fresnel collector technology.* Karlsruhe : Novatec Solar.

[6] **Wei, He; Ting, Wu Y.; Fang, Ma. C y Yuan, Ma. G.** *Performance study on a new type of engine for dish solar power system.* Pekín : National Natural Science Foundation of China.

[7] **Wagner, M. J. y Zhu, G.** *A generic CSP performance model for NREL's System Advisor Model.* Golden, Colorado (EEUU) : SolarPACES, 2011.

[8] **Universidad de Wisconsin.** *A Transient Systems Simulation Program. http://sel.me.wisc.edu/trnsys/*

[9] **Patnode, M. Angela.** *Simulation and Performance Evaluation of Parabolic Trough Solar Power Plants.* Wisconsin : Universidad de Wisconsin, 2006.

[10] **Bouaichaoui, S; Nourrdine, S; Khallf, A; Benkedda, Y. y Belhamel, M.** *Modeling and simulation of solar tower combined cycle power plant in Algeria.* Argél (Argelia) : Universidad de Saad Dahleb.

[11] **Renovetec.** *Diseño de simuladores de centrales en Renovetec. http://www.renovetec.com/index.php*

[12] **SimTech.** *IPSEpro. http://www.simtechnology.com/CMS/*

[13] **Herrman, Ulf.** *The PCTrough Performance Model.* s.l. : FLABEG Solar Int. GmbH, 2002.

[14] **MathWorks.** *Matlab. http://es.mathworks.com/*

[15] **Vergura, S. y Di Fronzo, V.** *Matlab based Model of 40-MW Concentrating Solar Power Plant.* Bari (Italia) : International Conference on Renewable Energies and Power Quality, 2012.

[16] **Hernández-Lobón, D.; Valenzuela, L. y Zarza, E.** *Tool for simulating direct steam generation in parabolic trough solar collectors.* Almería : CIEMAT-Plataforma Solar de Almería.

[17] **Modelica Association.** *Modelica. https://www.modelica.org/*

[18] **Dassault Systemes.** *Catia Systems Engineering-Dymola. http://www.3ds.com/products-services/catia/products/dymola*

[19] **JModelica.org.** *JModelica. http://www.jmodelica.org/*

[20] **Scicos.** *Modelicac. http://www.scicos.org/scicosmodelica.html*

[21] **OpenModelica.** *OpenModelica. https://openmodelica.org/*

[22] **Casella, F. y Leva, A.** *Modelica open library for power plant simulation: design and experimental validation.* Linköping (Suecia) : Proceedings of the 2003 Modelica Conference, 2003.

[23] **Casella, F. y Leva, A**. *Modelling of Thermo-Hydraulic Power Generation Processes Using Modelica.* Milán (Italia) : Politecnico de Milano, 2006. págs. 19-33. Vol. 12.

[24] **Link, K.; Gall, L.; Bonifay, J. y Buggert, M.** *Testing Power Plant Control Systems in Modelica.* Múnich : Proceedings of the 10th International Modelica Conference, 2014. 10.3384/ECP140961067.

[25] **Österholm, R. y Palsson, J.** *Dynamic modelling of a parabolic trough solar power plant.* Lund (Suecia) : Proceedings of the 10th International Modelica Conference, 2014. 10.3384/ECP140961057.

[26] **Ciemat. Ministerio de Economía y Competitividad.** *Plataforma Solar de Almería. http://www.psa.es/webesp/index.php*

[27] **Yebra, L. J.; Berenguel, M; Dormido, S. y Romero, M.** *Modelling and simulation of central receiver solar thermal power plants.* Hamburgo (Alemania) : Proceedings of the 4th International Modelica Conference, 2005. págs. 413-421.

[28] **Yebra, L. J.; Berenguel, M; Zarza, E. y Dormido, S.** *Object Oriented Modelling of DISS Solar Thermal Power Plant.* Almería : Modelica 2006 4th-5th, 2006.

[29] **Álvarez, J. D.; Gernjak, W.; Malato, S; Berenguel, M; Fürhacker, M. y Yebra, L.J..** *Control de peróxido de hidrógeno en sistemas solares foto-Fenton.*

[30] **Beschi, M.; Dormido, S.; Sánchez, J.; Visioli, A. y Yebra, J. L.** *Event-Based PI Plus Feedforward Control Strategies for a Distributed Solar Collector Field.* s.l. : IEEE Transactions on Control Systems Technology, 20013. 10.1109/TCST.2013.227921.

[31] **The Dow Chemical Company.** *Dowtherm. Systhetic Organic Fluids. http://www.dow.com/heattrans/products/synthetics.htm*

[32] **Pacheco, J. E.; Moursund, C; Rogers, D. y Wasyluk, D.** *Conceptual Design of a 100 MWe Modullar Molten Salt Power Tower Plant.* Burbank (California, EEUU) : eSolar.

[33] **Elmqvist, H; Cellier, F. E. y Otter, M.** *Object-Oriented Modeling of Hybrid Systems.* Delft (Holanda) : European Simulation Symposium, 1993. págs. XXXI-XLI.

[34] **Urquía, A. y Martín, C.** *Modelado orientado a objetos y simulación de sistemas físicos.* Madrid : Dpto. Informática y Automática, ETS Ingeniería Informática, UNED.

[35] **Wikipedia.** *Leyes de Kirchhoff. https://es.wikipedia.org/wiki/Leyes_de_Kirchhoff*

[36] **Wikipedia**. *Factor de fricción de Darcy-Weisbach. https://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_Darcy-Weisbach*

[37] **Wikipedia.** *Número de Reynolds. https://es.wikipedia.org/wiki/N%C3%BAmero_de_Reynolds*

[38] **Wikipedia.** *Ecuación de Colebrook-White. https://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_Colebrook-White*

[39] **Pedrollo.** *Línea de productos Pedrollo. http://www.pedrollo.com/es/default_t1*

[40] **Cooper, P. I.** *The absorption of radiation in solar stills.* s.l. : Solar Energy, 1969. págs. 333-346. Vol. 12.

[41] **Iqbal, M.** *An Introduction to Solar Radiation.* Nueva York : Academic Press, 1983.

[42] **Google.** *Google Maps. https://www.google.es/maps*

[43] **NREL.** *Glossary of Solar Radiation Resource Terms. http://rredc.nrel.gov/solar/glossary/*

[44] **OHL Industrial.** *Planta termosolar Arenales de 50 MW. Sevilla. http://www.ohlindustrial.com/proyectos/planta-termosolar-50-mw-arenales-sevilla/*

[45] **Abengoa.** *Solana, Abengoa. http://www.abengoasolar.com/export/sites/abengoasolar/resources/pdf/Solana_factsheet_09092013.pdf*

[46] **Abengoa**. *KaXu Solar One. Abengoa. http://www.abengoasolar.com/export/sites/abengoasolar/resources/pdf/KaXuSolarOne-factsheet.pdf*

[47] **Duffie, J. A. y Beckman, W. A.** *Solar Engineering of Thermal Processes.* 1991. 978-0-470-87366-3.

[48] **Geyer, M.; Lüpfert, Eckhard; Osuna, R.; Esteban, A.; Schiel, W.; Schweitzer, A.; Zarza, E.; Nava, P.; Langenkamp, J. y Mandelberg, E.** *EUROTROUGH-Parabolic Trough Collector Developed for Cost Efficient Solar Power Generation.* Zurich (Suiza) : 11th SolarPACES International Symposium, 2002.

[49] **Dudley, V. E.; Kolg, G.J.; Sloan, M. y Kearney, D.** *SEGS LS-2 Solar Collector .* Oak Ridge (Tenesse, EEUU) : Categories UC-1302,1303, 1994.

[50] **Solargenix Energy.** *Solargenix Energy, the natural power for good. http://www.solargenix.com/*

[51] **Stuetzle, Thorsten.** *Automatic Control of the 30 MWe SEGS VI Parabolic Trough Plant.* Madison : Universidad de Wisconsin, 2002.

[52] **Lippke, Frank.** *Simulation of the Part-Load Behavior of a 30 MWe SEGS Plant.* Springfield (Virginia, EEUU) : Sandia National Laboratories, 1995.

[53] **Burkholder, F. y Kutscher, C.** *Heat Loss Testing of Schott's 2008 PTR70 Parabolic Trough Receiver.* Golden (Colorado, EEUU) : National Renewable Energy Laboratory, 2009. NREL/TP-550-45633.

[54] **Wade, H. L.** *Basic and Advanced Regulatory Control: System Design and Applicaton.* Carolina del Norte (EEUU) : ISA-The Instrumentation, Systems, and Automation Society, 2004. 1-55617-873-5.

[55] **O'Dwyer, A.** *Handbook of PI and PID Controller Tuning Rules.* Dublín (Irlanda) : Imperial College Press, 2009. ISBN-13 978-1-84816-242-6.

Lista de siglas, abreviaturas y acrónimos

CSP: Centrales termosolares de concentración

DIPPR: Design Institute for Physical Properties

DNI: *Direct Normal Irradiance*. Radiación normal directa

GMT: *Greenwich Meridian Time*. Hora respecto al meridiano de Greenwich.

HCE: *Heat Collector Element*. Elemento de colector de energía térmica

HTF: *Heat Transfer Fluid*. Fluido de transferencia térmica

IAM: *Incident Angle Modifier*. Modificador del ángulo de incidencia

MSL: *Modelica Standard Library*. Librería estándar de Modelica

NREL: National Renewable Energy Laboratory

PSA: Plataforma Solar de Almería

PPA: *Power Purchase Agreement* (acuerdo de compra de electricidad)

TFM: Trabajo Fin de Master

TMY: *Typical Meteorological Year*. Año Típico Meteorológico.

TRNSYS: Transient Systems Simulation Program

SCA: *Solar Collector Assembly*. Colector solar.

SCADA: *Supervisory Control And Data Acquisition*. Contro supervisado y adquisición de datos.

SCE: *Solar Collector Element*. Elemento de colector solar.

SAM: System Advisor Model

Anexo A: Código fuente en *Modelica*

Nota: el código fuente aparece íntegro en este anexo, salvo los datos de las tablas de los datos meteorológicos que para simplificar debido a su larga extensión aparecen como "..." en lugar de los valores explícitos.

```
package ParaTrough
  "Modelica library to model and simulate parabolic-trough thermosolar
powerplants"

  import Modelica.SIunits.*;
  import Modelica.SIunits.Conversions.*;
  import Modelica.SIunits.Conversions.NonSIunits.*;
  import Modelica.Math.*;
  import Modelica.Constants.*;

  package media
    package DowthermA
      extends Modelica.Media.Interfaces.PartialMedium(
        final mediumName =   "Dowtherm A",
        final substanceNames={"Diphenyl", "Biphenyl Dioxide"},
        final singleState=true,
        final reducedX =  true,
        Temperature(min=285, max=678, start=566));

      redeclare model extends BaseProperties
        "Base properties of medium"

      SpecificHeatCapacity cp;
      ThermalConductivity lambda;
      DynamicViscosity eta;

      equation
        d = density(state);
        h = specificEnthalpy(state);
        cp=specificHeatCapacityCp(state);
        lambda=thermalConductivity(state);
        eta=dynamicViscosity(state);
        u = h - p/d;
        p = state.p;
        T = state.T;
        MM = 0.024;
        R  = 8.3144/MM;
        X[1]=1;

      end BaseProperties;

       redeclare record ThermodynamicState
        "a selction of variables that uniquely defines the
thermodynamic state"
        Modelica.SIunits.AbsolutePressure p "Absolute pressure of
medium";
        Modelica.SIunits.Temperature T "Temperature of medium";
       end ThermodynamicState;

      redeclare function extends density "DIPPR Equation"
```

```
      algorithm
      d:=165.96*0.489470/(0.253971^(1+(1-state.T/770.150)^0.285714));
      end density;

      redeclare function extends specificEnthalpy
        "Liquid specific enthalpy, acc. DIPPR"
      algorithm
      h:=(-
341.201+0.974117600572074*state.T+0.00092262142491796*(state.T)^2+0.00
000030738835324444*(state.T)^3)*1000;
      end specificEnthalpy;

      redeclare function extends dynamicViscosity "DIPPR equation"
      algorithm
      eta:=exp(-79.785786+4828.61482/state.T+10.29375*log(state.T)-
0.00000751488*(state.T)^2);
      end dynamicViscosity;

      redeclare function extends thermalConductivity "DIPPR equation"
      algorithm
      lambda:=0.18558680-0.00016*state.T;
      end thermalConductivity;

      redeclare function extends specificHeatCapacityCp
        "DIPPR equation"
      algorithm
      cp:=(405553-2252.92*state.T+9.63379*(state.T)^2-1.484336E-
2*(state.T)^3+8.346512E-6*(state.T)^4)*6.0255E-3;
      end specificHeatCapacityCp;

      function vaporPressure "DIPPR"
      input ThermodynamicState state;
      output Pressure Pv;
      algorithm
      Pv:=exp(83.8179-10103.3/state.T-8.58108*log(state.T)+2.109038E-
6*(state.T)^2);
      end vaporPressure;

      function latentHeat_vap "DIPPR eqquation"
      input ThermodynamicState state;
      output Modelica.SIunits.SpecificEnthalpy H_vap;
      algorithm
      H_vap:=1.631742E8/165960*(1-state.T/770.15)^(4.31180-
9.44562*state.T/770.15+9.17872*(state.T/770.15)^2-
3.46613*(state.T/770.15)^3);
      end latentHeat_vap;

    end DowthermA;

    package HitecSolarSalt
      extends Modelica.Media.Interfaces.PartialMedium(
        final mediumName =   "Hitec Solar Salt",
        final substanceNames={"Potasium Nitrate", "Sodium Nitrate"},
        final singleState=true,
        final reducedX =  true,
        Temperature(min=511, max=866, start=543));

      redeclare model extends BaseProperties
        "Base properties of medium"

      SpecificHeatCapacity cp;
```

```
      ThermalConductivity lambda;
      DynamicViscosity eta;

      equation
        d = density(state);
        h = specificEnthalpy(state);
        cp=specificHeatCapacityCp(state);
        lambda=thermalConductivity(state);
        eta=dynamicViscosity(state);
        u = h - p/d;
        p = state.p;
        T = state.T;
        MM = 0.024;
        R  = 8.3144/MM;
        X[1]=1;

      end BaseProperties;

       redeclare record ThermodynamicState
        "a selction of variables that uniquely defines the
thermodynamic state"
        Modelica.SIunits.AbsolutePressure p "Absolute pressure of
medium";
        Modelica.SIunits.Temperature T "Temperature of medium";
       end ThermodynamicState;

      redeclare function extends density "DIPPR equation"
      algorithm
      d:=2090-0.636*to_degC(state.T);
      end density;

      redeclare function extends specificEnthalpy "DIPPR equation"
      algorithm
      h:=1443*to_degC(state.T)+0.086*(to_degC(state.T))^2;
      end specificEnthalpy;

      redeclare function extends dynamicViscosity "DIPPR equation"
      algorithm
      eta:=(22.714-
0.12*to_degC(state.T)+0.0002281*(to_degC(state.T))^2-
0.0000001474*(to_degC(state.T))^3)/1000;
      end dynamicViscosity;

      redeclare function extends thermalConductivity "DIPPR equation"
      algorithm
      lambda:=0.443+0.00019*to_degC(state.T);
      end thermalConductivity;

      redeclare function extends specificHeatCapacityCp
        "DIPPR equation"
      algorithm
      cp:=1443+0.172*to_degC(state.T);
      end specificHeatCapacityCp;

    end HitecSolarSalt;

  package Water "Water using the IF97 standard, explicit in p and T"
    extends Modelica.Media.Water.WaterIF97_base(
        final ph_explicit=false,
        final dT_explicit=false,
        final pT_explicit=true,
```

```
        final smoothModel=true,
        final onePhase=true);
  end Water;

    model MediaChecking "Model to check media properties"

    parameter Boolean isobaric=false
        "type true for isobaric mode and analyse media against
temperature";
    parameter Boolean isothermal=true
        "type true for isothermal mode and analyse media against
pressure";

    //Constant parameters for the variable that not varies

    parameter Pressure p_c=1E5;
    parameter CelsiusTemperature t_c=365;

    replaceable package Medium=ParaTrough.media.Water;
    Medium.BaseProperties medium;

    Pressure p=medium.p;
    CelsiusTemperature t=medium.T_degC;

    equation
    assert( isobaric and not isothermal or isothermal and not
isobaric, "Error, choose one of the two modes: isobaric or
isothermal");

    t=if isobaric then time else t_c;
    p=if isothermal then time else p_c;

    end MediaChecking;
  end media;

  package basics

    package units "Additional units defined in ParaTrough"
    type AccRadiation = Real (final unit="MWh/m2");
    type Energy_MWh = Real (final unit="MWh");
    type HeatFlowRate_m = Real (final unit="W/m");

    end units;

    package math
      function sind "sinus of a function expressed in deg"
      input Real b;
      output Real a;

      algorithm
      a:=sin(2*pi/360*b);

      end sind;

      function cosd "cosinus of a function expressed in deg"
      input Real b;
      output Real a;

      algorithm
      a:=cos(2*pi/360*b);
```

```
  end cosd;

  function tand "tangent of a function expressed in deg"
  input Real b;
  output Real a;

  algorithm
  a:=tan(2*pi/360*b);

  end tand;

  function to_hour "Convert from day to hour"
    extends ConversionIcon;
    input NonSIunits.Time_day d "day value";
    output NonSIunits.Time_hour h "second value";
  algorithm
    h := d*24;
  end to_hour;

  function to_day "Convert from hour to day"
    extends ConversionIcon;
    input NonSIunits.Time_hour h "second value";
    output NonSIunits.Time_day d "day value";

  algorithm
    d := h/24;
  end to_day;

  function from_min_to_hour "Convert from minute to hour"
    extends ConversionIcon;
    input NonSIunits.Time_minute min "day value";
    output NonSIunits.Time_hour h "second value";
  algorithm
    h := min/60;
  end from_min_to_hour;

  function mega "Convert to mega"
    extends ConversionIcon;
    input Real r "initial quantity in base-unit";
    output Real R "final quantity in mega-unit";
  algorithm
    R := r/1E6;
  end mega;

  function to_MWh "Convert from Joule to Mega Watt hour"
    extends ConversionIcon;
    input Energy J "Joule value";
    output units.Energy_MWh MWh "MWh value";
  algorithm
    MWh := J/3.6e9;
  end to_MWh;
end math;

package connectors

  connector heatPort "Heat exchanging port"

  flow HeatFlowRate Q;
  CelsiusTemperature t;

  end heatPort;
```

```
      connector hfPort "Heat and hydraulic fluid port"

      flow MassFlowRate m;
      Pressure p;
      CelsiusTemperature t;

      end hfPort;

      connector weather_connector
        "solar connector that contains position of the sun, DNI,
temperature and wind speed"

      Angle_deg delta;
      Angle_deg omega;
      Angle_deg zenith;
      Angle_deg alpha;
      HeatFlux DNI;
      CelsiusTemperature t;
      Velocity ws;

      end weather_connector;

      connector analog_input
        extends Modelica.Blocks.Interfaces.RealInput;
      end analog_input;

      connector analog_output
        extends Modelica.Blocks.Interfaces.RealOutput;
      end analog_output;

    end connectors;

    package generic_heat
      partial model conduction

      ThermalConductivity K "equivalent resistance to heat
conduction";
      HeatFlowRate Q;
      CelsiusTemperature dt=heatPort1.t-heatPort2.t;

        connectors.heatPort heatPort1;
        connectors.heatPort heatPort2;
      equation
      Q=-heatPort2.Q;
      Q=heatPort1.Q;

      Q=K*dt;

      end conduction;

      partial model convection

      ThermalConductivity H "equivalent resistance to heat
convection";
      HeatFlowRate Q;
      CelsiusTemperature dt=heatPort1.t-heatPort2.t;

        connectors.heatPort heatPort1;
        connectors.heatPort heatPort2;
      equation
```

```
      Q=-heatPort2.Q;
      Q=heatPort1.Q;

      Q=H*dt;

      end convection;

      partial model radiation

      CrossSection E "Equivalent resistance to heat radiation";
      HeatFlowRate Q;
      Temperature T_k1=from_degC(heatPort1.t);
      Temperature T_k2=from_degC(heatPort2.t);
      Temperature dT4=T_k1^4-T_k2^4;

      equation
      Q=-heatPort2.Q;
      Q=heatPort1.Q;

      Q=sigma*E*dT4;

      public
        connectors.heatPort heatPort1;
        connectors.heatPort heatPort2;
      end radiation;

      model Ambient_simple
        "Ambient heat sink at constant temperature"

      parameter CelsiusTemperature T_amb=20 "ambient temperature";
      parameter Velocity ws=5 "wind speed";
      parameter CelsiusTemperature T_sky=T_amb-8 "estimated sky
temperature";

        connectors.heatPort heatPort;

      equation
      heatPort.t=T_amb;

      end Ambient_simple;

      model Ambient
        "Ambient heat sink in dependency with meteorological data"

      CelsiusTemperature T_amb=weather_connector.t "Ambient
Temperature";
      Velocity ws=weather_connector.ws "wind speed";
      CelsiusTemperature T_sky=T_amb-8 "estimated sky temperature";

        connectors.heatPort heatPort1;

        connectors.heatPort heatPort2;
        connectors.weather_connector weather_connector;
      equation
      heatPort1.t=T_amb;
      heatPort2.t=T_sky;

      end Ambient;

      model IrradiationEfficiency
```

```
      Efficiency eta "irradiation efficiency";

        connectors.weather_connector weather_connector1;
        connectors.weather_connector weather_connector2;
      equation
      weather_connector2.DNI=eta*weather_connector1.DNI;

      weather_connector2.t=weather_connector1.t;
      weather_connector2.ws=weather_connector1.ws;

      weather_connector2.zenith=weather_connector1.zenith;
      weather_connector2.omega=weather_connector1.omega;
      weather_connector2.delta=weather_connector1.delta;
      weather_connector2.alpha=weather_connector1.alpha;

      end IrradiationEfficiency;

    end generic_heat;

    package heat

      model k_annulus
        "1D Heat conduction along a cylindrical annulus"
      extends generic_heat.conduction;

      parameter Diameter Do "External Diameter";
      parameter Diameter Di "Internal Diameter";
      parameter Length L "Length of the cylindrical annulus";
      ThermalConductivity k "Thermal conductivity. It must be
defined";

      equation
      K=2*pi*k/log(Do/Di)*L;

      end k_annulus;

      model h_cylinder "1D Heat convection along a cylinder"
      extends generic_heat.convection;

      parameter Diameter D "Diameter";
      CoefficientOfHeatTransfer h
          "Coefficient of convection. It must be defined";
      parameter Length L "Length of the cylinder";

      equation
      H=h*pi*D*L;

      end h_cylinder;

      model r_annulus "1D Heat radiation along a cylindrical annulus"
      extends generic_heat.radiation;

      parameter Diameter Do "External Diameter";
      parameter Diameter Di "Internal Diameter";
      Emissivity epsilon_i
          "Thermal Emissivity of the internal material. It must be
defined";
      Emissivity epsilon_o "Themal Emissivity of the external
material";
      parameter Length L "Length of the cylindrical annulus";
```

```
    equation
    E=pi*Di/(1/epsilon_i+(1-epsilon_o)/epsilon_o*Di/Do)*L;

    end r_annulus;

    model r_cylinder "1D Heat radiation along a cylinder"
    extends generic_heat.radiation;

    parameter Diameter D "Diameter";
    Emissivity epsilon "Emmissivity of the material";
    parameter Length L "Length of the cylinder";

    equation
    E=epsilon*pi*D*L;

    end r_cylinder;
  end heat;

  package hydraulics_heat
    "Couplings between heat and fluid hydraulics"
    model source_m_t "Source of mass flow and temperature"

    parameter MassFlowRate m=8;
    parameter CelsiusTemperature t=293;

      connectors.hfPort hfPort;
    equation
    hfPort.m=-m;
    hfPort.t=t;

    end source_m_t;

    model source_t "Source of temperature"

    parameter CelsiusTemperature t=293;

      connectors.hfPort hfPort;
    equation
    hfPort.t=t;

    end source_t;

    model sink_p "Sink of pressure"

    parameter Pressure p=1E5;

      connectors.hfPort hfPort;
    equation

    hfPort.p=p;

    end sink_p;

    model pipe "pipe"


      import ParaTrough.media.*;

    replaceable package Medium=DowthermA;
    Medium.BaseProperties medium1 "properties of the medium in port
1";
```

```
      Medium.BaseProperties medium2 "properties of the medium in port
2";

      MassFlowRate m "mass flow rate";
      Pressure dp "pressure loss";

      parameter Length L=4 "length of the pipe";
      parameter Diameter D=0.066 "inner diameter of the pipe";
      parameter Length k=2.4e-5 "roughness";

      CoefficientOfFriction f( start=1) "coefficient of friction";
      CrossSection A=pi/4*D^2 "Cross section of the pipe";
      Velocity v=m/medium2.d/A "velocity of the fluid along the pipe";
      ReynoldsNumber Re=v*D*medium2.d/medium2.eta "Reynolds number";

        connectors.hfPort hfPort1;
        connectors.hfPort hfPort2;

      equation
      //Mass Balance
      m=-hfPort2.m;
      m=hfPort1.m;

      //Pressure loss
      dp=hfPort1.p-hfPort2.p;
      dp=f*L*v^2*medium2.d/(2*D) "Equation of Darcy-Weisbach";

      //Coefficient of friction

      f=if Re<=2300 then 64/Re else (1/(-
2*log10(k/D/3.7+2.51/Re/sqrt(f))))^2
          "Equation of Colebrook-White";

      medium1.T_degC=hfPort1.t;
      medium1.p=hfPort1.p;

      medium2.T_degC=hfPort2.t;
      medium2.p=hfPort2.p;

      end pipe;

      model pipe_heat "pipe with heat exchange"


      extends pipe;

      CelsiusTemperature t_avg=(hfPort1.t+hfPort2.t)/2;
      replaceable HeatFlowRate Q=heatPort.Q;
      Enthalpy H1=hfPort1.m*medium1.h;
      Enthalpy H2=-hfPort2.m*medium2.h;

        connectors.heatPort heatPort;
        connectors.hfPort hfPort1;
        connectors.hfPort hfPort2;

      equation
      //Heat Balance
      H2-H1=Q;
      heatPort.t=t_avg;

      end pipe_heat;
```

```
      model pipe_heat_rad
        "pipe with heat exchange and radiation input"


      extends pipe_heat(Q=weather_connector.DNI+heatPort.Q);

        connectors.hfPort hfPort1;
        connectors.hfPort hfPort2;

        connectors.weather_connector weather_connector;

      end pipe_heat_rad;

      model pump "Pump"
      import ParaTrough.media.*;

      replaceable package Medium=DowthermA;
      Medium.BaseProperties medium;

      parameter MassFlowRate m_const=2000;
      MassFlowRate m;
      Pressure dp=hfPort2.p-hfPort1.p;

      //Performance Curve
      parameter Length deltaH_0=20;
      parameter Length deltaH_1=5;

      parameter VolumeFlowRate q_0=0.01;
      parameter VolumeFlowRate q_1=2;

      VolumeFlowRate q=m/medium.d "volume flow rate";
      Length deltaH=dp/medium.d/g_n "head";

        connectors.analog_input analog_input;
      equation
      m=hfPort1.m;
      m=-hfPort2.m;

      hfPort1.t=hfPort2.t;

      m=analog_input;
      if cardinality(analog_input)==0 then
      analog_input=m_const;
      end if;

      deltaH=if m<small then 0 else deltaH_0+(deltaH_0-deltaH_1)/(q_0-
q_1)*q;

      medium.T_degC=hfPort2.t;
      medium.p=hfPort2.p;

      public
        connectors.hfPort hfPort1;
        connectors.hfPort hfPort2;
      end pump;

    end hydraulics_heat;

    package instruments
      model temperature_sensor
```

```
      equation
      hfPort.t=analog_output;
      hfPort.m=0;

      public
        connectors.hfPort hfPort;
        connectors.analog_output analog_output;
      end temperature_sensor;

      model pressure_sensor

      equation
      hfPort.p=analog_output;
      hfPort.m=0;

      public
        connectors.hfPort hfPort;
        connectors.analog_output analog_output;
      end pressure_sensor;

      model massflow_sensor

        connectors.hfPort hfPort2;
      equation
      hfPort1.m=analog_output;
      hfPort2.m=-analog_output;

      public
        connectors.hfPort hfPort1;
        connectors.analog_output analog_output;
      equation
        connect(hfPort2, hfPort2);
      end massflow_sensor;
    end instruments;

    package control
      block LimPID_direct
        "P, PI, PD, and PID controller with limited output, anti-
windup compensation and setpoint weighting"
        import Modelica.Blocks.Types.InitPID;
        import Modelica.Blocks.Types.SimpleController;
        extends Modelica.Blocks.Interfaces.SVcontrol;
        output Real controlError = u_s - u_m
          "Control error (set point - measurement)";

        parameter SimpleController.Temp controllerType=
               Modelica.Blocks.Types.SimpleController.PID "Type of
controller";
        parameter Real k(min=0) = 1 "Gain of controller";
        parameter Modelica.SIunits.Time
Ti(min=Modelica.Constants.small)=0.5
          "Time constant of Integrator block";
        parameter Modelica.SIunits.Time Td(min=0)=0.1
          "Time constant of Derivative block";
        parameter Real yMax=1 "Upper limit of output";
        parameter Real yMin=-yMax "Lower limit of output";
        parameter Real wp(min=0) = 1
          "Set-point weight for Proportional block (0..1)";
        parameter Real wd(min=0) = 0
          "Set-point weight for Derivative block (0..1)";
```

```
        parameter Real Ni(min=100*Modelica.Constants.eps) = 0.9
          "Ni*Ti is time constant of anti-windup compensation";
        parameter Real Nd(min=100*Modelica.Constants.eps) = 10
          "The higher Nd, the more ideal the derivative block";
        parameter InitPID.Temp
initType=Modelica.Blocks.Types.InitPID.DoNotUse_InitialIntegratorState
          "Type of initialization";
        parameter Boolean limitsAtInit = true
          "= false, if limits are ignored during initializiation";
        parameter Real xi_start=0
          "Initial or guess value value for integrator output (=
integrator state)";
        parameter Real xd_start=0
          "Initial or guess value for state of derivative block";
        parameter Real y_start=0 "Initial value of output";

        Modelica.Blocks.Math.Add addP(k1=wp, k2=-1);
        Modelica.Blocks.Math.Add addD(k1=wd, k2=-1) if with_D;
        Modelica.Blocks.Math.Gain P;
        Modelica.Blocks.Continuous.Integrator I(
          k=1/Ti,
          y_start=xi_start,
          initType=if initType == InitPID.SteadyState then
InitPID.SteadyState else
                   if initType == InitPID.InitialState or initType ==
InitPID.DoNotUse_InitialIntegratorState then
                   InitPID.InitialState else InitPID.NoInit) if
with_I;
        Modelica.Blocks.Continuous.Derivative D(
          k=Td,
          T=max([Td/Nd,1.e-14]),
          x_start=xd_start,
          initType=if initType == InitPID.SteadyState or initType ==
InitPID.InitialOutput then
                   InitPID.SteadyState else if initType ==
InitPID.InitialState then
                   InitPID.InitialState else InitPID.NoInit) if
with_D;
        Modelica.Blocks.Math.Gain gainPID(k=k);
        Modelica.Blocks.Math.Add3 addPID(k1=+1);
        Modelica.Blocks.Math.Add3 addI(k2=-1) if with_I;
        Modelica.Blocks.Math.Add addSat(k1=+1, k2=-1) if with_I;
        Modelica.Blocks.Math.Gain gainTrack(k=1/(k*Ni)) if with_I;
        Modelica.Blocks.Nonlinear.Limiter limiter(
          uMax=yMax,
          uMin=yMin,
          limitsAtInit=limitsAtInit);
      protected
        parameter Boolean with_I = controllerType==SimpleController.PI
or

controllerType==SimpleController.PID;
        parameter Boolean with_D = controllerType==SimpleController.PD
or

controllerType==SimpleController.PID;
      public
        Modelica.Blocks.Sources.Constant Dzero(k=0) if not with_D;
        Modelica.Blocks.Sources.Constant Izero(k=0) if not with_I;
      initial equation
        if initType==InitPID.InitialOutput then
```

```
          y = y_start;
        end if;
      equation
        assert(yMax >= yMin, "LimPID: Limits must be consistent.
However, yMax (=" + String(yMax) +
                              ") < yMin (=" + String(yMin) + ")");
        if initType == InitPID.InitialOutput and (y_start < yMin or
y_start > yMax) then
          Modelica.Utilities.Streams.error("LimPID: Start value
y_start (=" + String(y_start) +
            ") is outside of the limits of yMin (=" + String(yMin)
+") and yMax (=" + String(yMax) + ")");
        end if;
        assert(limitsAtInit or not limitsAtInit and y >= yMin and y <=
yMax,
            "LimPID: During initialization the limits have been
switched off.\n" +
            "After initialization, the output y (=" + String(y) +
            ") is outside of the limits of yMin (=" + String(yMin)
+") and yMax (=" + String(yMax) + ")");

        connect(u_s, addP.u1);
        connect(u_s, addD.u1);
        connect(u_s, addI.u1);
        connect(addP.y, P.u);
        connect(addD.y, D.u);
        connect(addI.y, I.u);
        connect(P.y, addPID.u1);
        connect(D.y, addPID.u2);
        connect(I.y, addPID.u3);
        connect(addPID.y, gainPID.u);
        connect(gainPID.y, addSat.u2);
        connect(gainPID.y, limiter.u);
        connect(limiter.y, addSat.u1);
        connect(limiter.y, y);
        connect(addSat.y, gainTrack.u);
        connect(gainTrack.y, addI.u3);
        connect(u_m, addP.u2);
        connect(u_m, addD.u2);
        connect(u_m, addI.u2);
        connect(Dzero.y, addPID.u2);
        connect(Izero.y, addPID.u3);
      end LimPID_direct;

      block LimPID_indirect
        "P, PI, PD, and PID controller with limited output, anti-
windup compensation and setpoint weighting"
        import Modelica.Blocks.Types.InitPID;
        import Modelica.Blocks.Types.SimpleController;
        extends Modelica.Blocks.Interfaces.SVcontrol;
        output Real controlError = u_s - u_m
          "Control error (set point - measurement)";

        parameter SimpleController.Temp controllerType=
            Modelica.Blocks.Types.SimpleController.PID "Type of
controller";
        parameter Real k(min=0) = 1 "Gain of controller";
        parameter Modelica.SIunits.Time
Ti(min=Modelica.Constants.small)=0.5
          "Time constant of Integrator block";
        parameter Modelica.SIunits.Time Td(min=0)=0.1
```

```
          "Time constant of Derivative block";
        parameter Real yMax=1 "Upper limit of output";
        parameter Real yMin=-yMax "Lower limit of output";
        parameter Real wp(min=0) = 1
          "Set-point weight for Proportional block (0..1)";
        parameter Real wd(min=0) = 0
          "Set-point weight for Derivative block (0..1)";
        parameter Real Ni(min=100*Modelica.Constants.eps) = 0.9
          "Ni*Ti is time constant of anti-windup compensation";
        parameter Real Nd(min=100*Modelica.Constants.eps) = 10
          "The higher Nd, the more ideal the derivative block";
        parameter InitPID.Temp
initType=Modelica.Blocks.Types.InitPID.DoNotUse_InitialIntegratorState
          "Type of initialization";
        parameter Boolean limitsAtInit = true
          "= false, if limits are ignored during initializiation";
        parameter Real xi_start=0
          "Initial or guess value value for integrator output (=
integrator state)";
        parameter Real xd_start=0
          "Initial or guess value for state of derivative block";
        parameter Real y_start=0 "Initial value of output";

        Modelica.Blocks.Math.Add addP(k1=-wp, k2=+1);
        Modelica.Blocks.Math.Add addD(k1=-wd, k2=+1) if
                                                    with_D;
        Modelica.Blocks.Math.Gain P;
        Modelica.Blocks.Continuous.Integrator I(
          k=1/Ti,
          y_start=xi_start,
          initType=if initType == InitPID.SteadyState then
InitPID.SteadyState else
                   if initType == InitPID.InitialState or initType ==
InitPID.DoNotUse_InitialIntegratorState then
                   InitPID.InitialState else InitPID.NoInit) if
with_I;
        Modelica.Blocks.Continuous.Derivative D(
          k=Td,
          T=max([Td/Nd,1.e-14]),
          x_start=xd_start,
          initType=if initType == InitPID.SteadyState or initType ==
InitPID.InitialOutput then
                   InitPID.SteadyState else if initType ==
InitPID.InitialState then
                   InitPID.InitialState else InitPID.NoInit) if
with_D;
        Modelica.Blocks.Math.Gain gainPID(k=k);
        Modelica.Blocks.Math.Add3 addPID(
          k1=+1,
          k2=+1,
          k3=+1);
        Modelica.Blocks.Math.Add3 addI(k1=-1, k2=+1) if
                                                  with_I;
        Modelica.Blocks.Math.Add addSat(k1=+1, k2=-1) if with_I;
        Modelica.Blocks.Math.Gain gainTrack(k=1/(k*Ni)) if with_I;
        Modelica.Blocks.Nonlinear.Limiter limiter(
          uMax=yMax,
          uMin=yMin,
          limitsAtInit=limitsAtInit);
      protected
```

```
        parameter Boolean with_I = controllerType==SimpleController.PI
or

controllerType==SimpleController.PID;
        parameter Boolean with_D = controllerType==SimpleController.PD
or

controllerType==SimpleController.PID;
      public
        Modelica.Blocks.Sources.Constant Dzero(k=0) if not with_D;
        Modelica.Blocks.Sources.Constant Izero(k=0) if not with_I;
      initial equation
        if initType==InitPID.InitialOutput then
           y = y_start;
        end if;
      equation
        assert(yMax >= yMin, "LimPID: Limits must be consistent.
However, yMax (=" + String(yMax) +
                             ") < yMin (=" + String(yMin) + ")");
        if initType == InitPID.InitialOutput and (y_start < yMin or
y_start > yMax) then
            Modelica.Utilities.Streams.error("LimPID: Start value
y_start (=" + String(y_start) +
              ") is outside of the limits of yMin (=" + String(yMin)
+") and yMax (=" + String(yMax) + ")");
        end if;
        assert(limitsAtInit or not limitsAtInit and y >= yMin and y <=
yMax,
              "LimPID: During initialization the limits have been
switched off.\n" +
              "After initialization, the output y (=" + String(y) +
              ") is outside of the limits of yMin (=" + String(yMin)
+") and yMax (=" + String(yMax) + ")");

        connect(u_s, addP.u1);
        connect(u_s, addD.u1);
        connect(u_s, addI.u1);
        connect(addP.y, P.u);
        connect(addD.y, D.u);
        connect(addI.y, I.u);
        connect(P.y, addPID.u1);
        connect(D.y, addPID.u2);
        connect(I.y, addPID.u3);
        connect(addPID.y, gainPID.u);
        connect(gainPID.y, addSat.u2);
        connect(gainPID.y, limiter.u);
        connect(limiter.y, addSat.u1);
        connect(limiter.y, y);
        connect(addSat.y, gainTrack.u);
        connect(gainTrack.y, addI.u3);
        connect(u_m, addP.u2);
        connect(u_m, addD.u2);
        connect(u_m, addI.u2);
        connect(Dzero.y, addPID.u2);
        connect(Izero.y, addPID.u3);
      end LimPID_indirect;

    end control;

  end basics;
```

```
  package A_SolarResource
    "Solar position model troughout the year and weather conditions in
different locations"

    import ParaTrough.basics.math.*;
    import ParaTrough.basics.units.*;

    package Tables
      model Generic1DDataTable
        "Generic interpolation tool to interpolete data from a 1D
table. It also serves to upload new data to ParaTrough"

        Modelica.Blocks.Tables.CombiTable1D combiTable1D;
        Modelica.Blocks.Interfaces.RealInput u;
        Modelica.Blocks.Interfaces.RealOutput y;
      equation

        connect(combiTable1D.y[1], y);
        connect(u, combiTable1D.u[1]);
      end Generic1DDataTable;

      model Generic2DDataTable
        "Generic interpolation tool to interpolete data from a 2D
table. It also serves to upload new data to ParaTrough"

        Modelica.Blocks.Tables.CombiTable2D CombiTable2D(

smoothness=Modelica.Blocks.Types.Smoothness.LinearSegments,
table=fill(0,0,0));
        Modelica.Blocks.Interfaces.RealInput u1;
        Modelica.Blocks.Interfaces.RealInput u2;
        Modelica.Blocks.Interfaces.RealOutput y;
      equation
        connect(u1, CombiTable2D.u1);
        connect(u2, CombiTable2D.u2);
        connect(CombiTable2D.y, y);
        connect(y, y);

      end Generic2DDataTable;

      model dayOfYear
        "Table to calculate day of year from day (rows) and month
(columns)"
        extends
Generic2DDataTable(CombiTable2D(table=[0,1,2,3,4,5,6,7,8,9,10,11,12;
                  1,1,32,60,91,121,152,182,213,244,274,305,335;
2,2,33,61,92,122,153,
                  183,214,245,275,306,336;
3,3,34,62,93,123,154,184,215,246,276,307,337;
                  4,4,35,63,94,124,155,185,216,247,277,308,338;
5,5,36,64,95,125,156,
                  186,217,248,278,309,339;
6,6,37,65,96,126,157,187,218,249,279,310,340;
                  7,7,38,66,97,127,158,188,219,250,280,311,341;
8,8,39,67,98,128,159,
                  189,220,251,281,312,342;
9,9,40,68,99,129,160,190,221,252,282,313,343;
                  10,10,41,69,100,130,161,191,222,253,283,314,344;
11,11,42,70,101,131,
                  162,192,223,254,284,315,345;
12,12,43,71,102,132,163,193,224,255,285,
```

```
                316,346;
13,13,44,72,103,133,164,194,225,256,286,317,347; 14,14,45,73,
                104,134,165,195,226,257,287,318,348;
15,15,46,74,105,135,166,196,227,
                258,288,319,349;
16,16,47,75,106,136,167,197,228,259,289,320,350; 17,
                17,48,76,107,137,168,198,229,260,290,321,351;
18,18,49,77,108,138,169,
                199,230,261,291,322,352;
19,19,50,78,109,139,170,200,231,262,292,323,
                353; 20,20,51,79,110,140,171,201,232,263,293,324,354;
21,21,52,80,111,
                141,172,202,233,264,294,325,355;
22,22,53,81,112,142,173,203,234,265,
                295,326,356;
23,23,54,82,113,143,174,204,235,266,296,327,357; 24,24,
                55,83,114,144,175,205,236,267,297,328,358;
25,25,56,84,115,145,176,
                206,237,268,298,329,359;
26,26,57,85,116,146,177,207,238,269,299,330,
                360; 27,27,58,86,117,147,178,208,239,270,300,331,361;
28,28,59,87,118,
                148,179,209,240,271,301,332,362;
29,29,0,88,119,149,180,210,241,272,
                302,333,363;
30,30,0,89,120,150,181,211,242,273,303,334,364; 31,31,0,
                90,0,151,0,212,243,0,304,0,365]));

      Integer n "day of year";

      equation
      n=integer(y+to_day(time));

      end dayOfYear;

      model day_from_n "Table to calculate day from day of year (n)"
        extends Generic1DDataTable(combiTable1D(table=[1,1; 2,2; 3,3;
4,4; 5,5;
                6,6; 7,7; 8,8; 9,9; 10,10; 11,11; 12,12; 13,13; 14,14;
15,15;
                16,16; 17,17; 18,18; 19,19; 20,20; 21,21; 22,22;
23,23; 24,24;
                25,25; 26,26; 27,27; 28,28; 29,29; 30,30; 31,31; 32,1;
33,2; 34,
                3; 35,4; 36,5; 37,6; 38,7; 39,8; 40,9; 41,10; 42,11;
43,12; 44,
                13; 45,14; 46,15; 47,16; 48,17; 49,18; 50,19; 51,20;
52,21; 53,
                22; 54,23; 55,24; 56,25; 57,26; 58,27; 59,28; 60,1;
61,2; 62,3;
                63,4; 64,5; 65,6; 66,7; 67,8; 68,9; 69,10; 70,11;
71,12; 72,13;
                73,14; 74,15; 75,16; 76,17; 77,18; 78,19; 79,20;
80,21; 81,22;
                82,23; 83,24; 84,25; 85,26; 86,27; 87,28; 88,29;
89,30; 90,31;
                91,1; 92,2; 93,3; 94,4; 95,5; 96,6; 97,7; 98,8; 99,9;
100,10;
                101,11; 102,12; 103,13; 104,14; 105,15; 106,16;
107,17; 108,18;
```

```
                109,19; 110,20; 111,21; 112,22; 113,23; 114,24;
115,25; 116,26;
                117,27; 118,28; 119,29; 120,30; 121,1; 122,2; 123,3;
124,4; 125,
                5; 126,6; 127,7; 128,8; 129,9; 130,10; 131,11; 132,12;
133,13;
                134,14; 135,15; 136,16; 137,17; 138,18; 139,19;
140,20; 141,21;
                142,22; 143,23; 144,24; 145,25; 146,26; 147,27;
148,28; 149,29;
                150,30; 151,31; 152,1; 153,2; 154,3; 155,4; 156,5;
157,6; 158,7;
                159,8; 160,9; 161,10; 162,11; 163,12; 164,13; 165,14;
166,15;
                167,16; 168,17; 169,18; 170,19; 171,20; 172,21;
173,22; 174,23;
                175,24; 176,25; 177,26; 178,27; 179,28; 180,29;
181,30; 182,1;
                183,2; 184,3; 185,4; 186,5; 187,6; 188,7; 189,8;
190,9; 191,10;
                192,11; 193,12; 194,13; 195,14; 196,15; 197,16;
198,17; 199,18;
                200,19; 201,20; 202,21; 203,22; 204,23; 205,24;
206,25; 207,26;
                208,27; 209,28; 210,29; 211,30; 212,31; 213,1; 214,2;
215,3;
                216,4; 217,5; 218,6; 219,7; 220,8; 221,9; 222,10;
223,11; 224,
                12; 225,13; 226,14; 227,15; 228,16; 229,17; 230,18;
231,19; 232,
                20; 233,21; 234,22; 235,23; 236,24; 237,25; 238,26;
239,27; 240,
                28; 241,29; 242,30; 243,31; 244,1; 245,2; 246,3;
247,4; 248,5;
                249,6; 250,7; 251,8; 252,9; 253,10; 254,11; 255,12;
256,13; 257,
                14; 258,15; 259,16; 260,17; 261,18; 262,19; 263,20;
264,21; 265,
                22; 266,23; 267,24; 268,25; 269,26; 270,27; 271,28;
272,29; 273,
                30; 274,1; 275,2; 276,3; 277,4; 278,5; 279,6; 280,7;
281,8; 282,
                9; 283,10; 284,11; 285,12; 286,13; 287,14; 288,15;
289,16; 290,
                17; 291,18; 292,19; 293,20; 294,21; 295,22; 296,23;
297,24; 298,
                25; 299,26; 300,27; 301,28; 302,29; 303,30; 304,31;
305,1; 306,
                2; 307,3; 308,4; 309,5; 310,6; 311,7; 312,8; 313,9;
314,10; 315,
                11; 316,12; 317,13; 318,14; 319,15; 320,16; 321,17;
322,18; 323,
                19; 324,20; 325,21; 326,22; 327,23; 328,24; 329,25;
330,26; 331,
                27; 332,28; 333,29; 334,30; 335,1; 336,2; 337,3;
338,4; 339,5;
                340,6; 341,7; 342,8; 343,9; 344,10; 345,11; 346,12;
347,13; 348,
                14; 349,15; 350,16; 351,17; 352,18; 353,19; 354,20;
355,21; 356,
```

```
                22; 357,23; 358,24; 359,25; 360,26; 361,27; 362,28;
363,29; 364,
                30; 365,31]));
      end day_from_n;

      model month_from_n
        "Table to calculate day from day of year (n)"
        extends Generic1DDataTable(combiTable1D(table=[1,1; 2,1; 3,1;
4,1; 5,1;
                6,1; 7,1; 8,1; 9,1; 10,1; 11,1; 12,1; 13,1; 14,1;
15,1; 16,1;
                17,1; 18,1; 19,1; 20,1; 21,1; 22,1; 23,1; 24,1; 25,1;
26,1; 27,
                1; 28,1; 29,1; 30,1; 31,1; 32,2; 33,2; 34,2; 35,2;
36,2; 37,2;
                38,2; 39,2; 40,2; 41,2; 42,2; 43,2; 44,2; 45,2; 46,2;
47,2; 48,
                2; 49,2; 50,2; 51,2; 52,2; 53,2; 54,2; 55,2; 56,2;
57,2; 58,2;
                59,2; 60,3; 61,3; 62,3; 63,3; 64,3; 65,3; 66,3; 67,3;
68,3; 69,
                3; 70,3; 71,3; 72,3; 73,3; 74,3; 75,3; 76,3; 77,3;
78,3; 79,3;
                80,3; 81,3; 82,3; 83,3; 84,3; 85,3; 86,3; 87,3; 88,3;
89,3; 90,
                3; 91,4; 92,4; 93,4; 94,4; 95,4; 96,4; 97,4; 98,4;
99,4; 100,4;
                101,4; 102,4; 103,4; 104,4; 105,4; 106,4; 107,4;
108,4; 109,4;
                110,4; 111,4; 112,4; 113,4; 114,4; 115,4; 116,4;
117,4; 118,4;
                119,4; 120,4; 121,5; 122,5; 123,5; 124,5; 125,5;
126,5; 127,5;
                128,5; 129,5; 130,5; 131,5; 132,5; 133,5; 134,5;
135,5; 136,5;
                137,5; 138,5; 139,5; 140,5; 141,5; 142,5; 143,5;
144,5; 145,5;
                146,5; 147,5; 148,5; 149,5; 150,5; 151,5; 152,6;
153,6; 154,6;
                155,6; 156,6; 157,6; 158,6; 159,6; 160,6; 161,6;
162,6; 163,6;
                164,6; 165,6; 166,6; 167,6; 168,6; 169,6; 170,6;
171,6; 172,6;
                173,6; 174,6; 175,6; 176,6; 177,6; 178,6; 179,6;
180,6; 181,6;
                182,7; 183,7; 184,7; 185,7; 186,7; 187,7; 188,7;
189,7; 190,7;
                191,7; 192,7; 193,7; 194,7; 195,7; 196,7; 197,7;
198,7; 199,7;
                200,7; 201,7; 202,7; 203,7; 204,7; 205,7; 206,7;
207,7; 208,7;
                209,7; 210,7; 211,7; 212,7; 213,8; 214,8; 215,8;
216,8; 217,8;
                218,8; 219,8; 220,8; 221,8; 222,8; 223,8; 224,8;
225,8; 226,8;
                227,8; 228,8; 229,8; 230,8; 231,8; 232,8; 233,8;
234,8; 235,8;
                236,8; 237,8; 238,8; 239,8; 240,8; 241,8; 242,8;
243,8; 244,9;
                245,9; 246,9; 247,9; 248,9; 249,9; 250,9; 251,9;
252,9; 253,9;
```

```
                254,9; 255,9; 256,9; 257,9; 258,9; 259,9; 260,9;
261,9; 262,9;
                263,9; 264,9; 265,9; 266,9; 267,9; 268,9; 269,9;
270,9; 271,9;
                272,9; 273,9; 274,10; 275,10; 276,10; 277,10; 278,10;
279,10;
                280,10; 281,10; 282,10; 283,10; 284,10; 285,10;
286,10; 287,10;
                288,10; 289,10; 290,10; 291,10; 292,10; 293,10;
294,10; 295,10;
                296,10; 297,10; 298,10; 299,10; 300,10; 301,10;
302,10; 303,10;
                304,10; 305,11; 306,11; 307,11; 308,11; 309,11;
310,11; 311,11;
                312,11; 313,11; 314,11; 315,11; 316,11; 317,11;
318,11; 319,11;
                320,11; 321,11; 322,11; 323,11; 324,11; 325,11;
326,11; 327,11;
                328,11; 329,11; 330,11; 331,11; 332,11; 333,11;
334,11; 335,12;
                336,12; 337,12; 338,12; 339,12; 340,12; 341,12;
342,12; 343,12;
                344,12; 345,12; 346,12; 347,12; 348,12; 349,12;
350,12; 351,12;
                352,12; 353,12; 354,12; 355,12; 356,12; 357,12;
358,12; 359,12;
                360,12; 361,12; 362,12; 363,12; 364,12; 365,12]));
      end month_from_n;
    end Tables;

    package Functions "Functions to model the position of the sun"

      function declinationangle "Declination angle of the Earth"
      input Integer n
          "day of the year, from 1 (Juanuary 1st) to 365 (December
31st)";
      output Angle_deg delta "Declination angle of the Earth";

      algorithm
        delta := 23.45*sind(360*(284 + n)/365) "P.I.Cooper,1969";

      end declinationangle;

      function hourangle
        "Hour angle resulted from the rotation of Earth, which spins
on its axis at a rate of 15º per hour"
      input Time_hour SoTime "Solar Time";
      output Angle_deg omega "Hour angle";

      algorithm
        omega:=(SoTime - 12)*15;

      end hourangle;

      function equationoftime
        "Equation of time takes into account the effect of the Earth's
elliptical path around the sun that impacts the day length"
      input Integer n
          "day of the year, from 1 (Juanuary 1st) to 365 (December
31st)";
      output Time_minute E "equation of time";
```

```
      algorithm
      E:=229.18*(0.000075+0.001868*cosd(360/365*(n-1))-
0.032077*sind(360/365*(n-1))-0.014615*cosd(2*(360/365*(n-1)))-
0.04089*sind(2*(360/365*(n-1))))
          "Spencer (as cited by Iqbal,1983)";

      end equationoftime;

    end Functions;

    package Meteo

      record data

      constant Angle_deg latitude "latitude of the plant location";
      constant Angle_deg longitude "local meridian of the collector
site";
      constant Angle_deg meridian "standard meridian for the local
time hour";

      constant Real DNI[:, :]=fill(0,0,0)
          "2D table that correlates time,day of year and DNI";
      constant Real AmbTemp[:,:]=fill(0,0,0)
          "2D table tahat correlates time, day of year and ambient
temperature";
      constant Real WindSpeed[:,:]=fill(0,0,0)
          "2D table that correlates time, day of year and ambient
temperature";

      end data;

      record Sevilla
      extends data(
          latitude=37.12,
          longitude=-5.56,
          meridian=15,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end Sevilla;

      record Phoenix_EEUU
      extends data(
          latitude=32.93,
          longitude=-112.98,
          meridian=-105,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end Phoenix_EEUU;

      record Harare_Zimbabwe
      extends data(
          latitude=-17.92,
          longitude=31.13,
          meridian=30,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end Harare_Zimbabwe;
```

```
      record Madras_India
      extends data(
          latitude=13,
          longitude=80.18,
          meridian=82.5,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end Madras_India;

      record Antofagasta_Chile
      extends data(
          latitude=-23.43,
          longitude=-70.43,
          meridian=-60,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end Antofagasta_Chile;

      record FullySunnyDay_140529
        "1 minute resolution data of a fully sunny day in Aldeire on
29th of May of 2014"
      extends data(
          latitude=37.2,
          longitude=-3.1,
          meridian=15,
          meridian=-60,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
        end FullySunnyDay_140529;

      record PartialCloudyDay_140417
        "1 minute resolution data of a fully sunny day in Aldeire on
17th of April of 2014"
      extends data(
          latitude=37.2,
          longitude=-3.1,
          meridian=15,
          DNI=["..."],
          AmbTemp=["..."],
          WindSpeed=["..."]);
      end PartialCloudyDay_140417;
    end Meteo;

    package Models

      import ParaTrough.basics.math.*;

      model DNI
        "Interpolation tool for DNI (Direct Normal Irradiation)"

      extends Tables.Generic2DDataTable;

      end DNI;

      model AmbTemp "Interpolation tool ambient temperature"
      extends Tables.Generic2DDataTable;
```

```
      end AmbTemp;

      model WindSpeed "Interpolation tool for wind speed"

      extends Tables.Generic2DDataTable;

      end WindSpeed;

      model WeatherEvaluation
        "Interpolation tool to evaluate DNI, ambient temperature and
wind speed at the same time"

      extends Modelica.Icons.Record;

        Modelica.Blocks.Interfaces.RealInput StTime;
        ParaTrough.A_SolarResource.Models.DNI dNI;
        ParaTrough.A_SolarResource.Models.AmbTemp ambTemp;
        ParaTrough.A_SolarResource.Models.WindSpeed windSpeed;
        Modelica.Blocks.Interfaces.RealInput n;
      equation

        connect(StTime, dNI.u1);
        connect(StTime, ambTemp.u1);
        connect(StTime, windSpeed.u1);
        connect(n, dNI.u2);
        connect(n, ambTemp.u2);
        connect(n, windSpeed.u2);
      end WeatherEvaluation;

      model Sun "Model that predicts the position of the sun "
        import ParaTrough.A_SolarResource.Functions.*;
        import ParaTrough.A_SolarResource.Meteo.*;

      //Replaceable data
      replaceable record data=FullySunnyDay_140529 extends data;

      //Parameters
      parameter Integer day_0=29 "day of evaluation";
      parameter Integer month_0=5 "month of evaluation";

      //Public Variables
      Integer day=integer(Day_from_n.y);
      Integer month=integer(Month_from_n.y);
      Time_hour StTime=WeatherEvaluator.StTime "standard time";
      HeatFlux DNI=if WeatherEvaluator.dNI.y<0 then 0 else
WeatherEvaluator.dNI.y
          "Direct Normal Irradiation";
      Angle_deg alpha=weather_connector.alpha "solar height";
      CelsiusTemperature AmbTemp=WeatherEvaluator.ambTemp.y
          "Ambient Temperature";
      Velocity ws=if WeatherEvaluator.windSpeed.y<0 then 0 else
WeatherEvaluator.windSpeed.y
          "Wind Speed";
      AccRadiation DNI_acc "Accumulated DNI since simulation starts";

      //Protected SubModels
      protected
      Models.WeatherEvaluation WeatherEvaluator(
          dNI(CombiTable2D(table=data.DNI)),
          ambTemp(CombiTable2D(table=data.AmbTemp)),
          windSpeed(CombiTable2D(table=data.WindSpeed)));
```

```
      Tables.dayOfYear DayOfYear;
      Tables.day_from_n Day_from_n;
      Tables.month_from_n Month_from_n;

      //Protected Constants
      constant Integer Rotation_rate=15 "Rotation rate of the Earth,
º/hour";
      constant Integer SummerTime=1 "period of summer time (+1 hour)";
      constant Integer WinterTime=0 "period of winter time (+0 hour)";
      constant Integer SummerTime1stDay=86
          "day in which starts summer time, March 27th";
      constant Integer WinterTime1stDay=300
          "day in which starts winter time, October 27th";

      //Protected Variables
      Integer n=integer(WeatherEvaluator.n)
          "day of the year, from 1 (Juanuary 1st) to 365 (December
31st)";
      //Real day_prima=WeatherEvaluator.Day "Day of the year in
dependency with time";
      Integer n_0=integer(DayOfYear.y);
      Angle_deg omega=weather_connector.omega "hour angle";
      Angle_deg zenith=weather_connector.zenith "zenith angle";
      Angle_deg delta=weather_connector.delta "declination angle";
      Time_hour SoTime=StTime-DST+(data.longitude-
data.meridian)/Rotation_rate+from_min_to_hour(equationoftime(n))
          "Solar time: corrected time that fix 12:00 when the sun
aligns with the local meridian";
      Integer DST=if n>SummerTime1stDay and n<WinterTime1stDay then
SummerTime else WinterTime
          "Daylight Savings Time, summer time between March 27th and
October 27th";

      public
        ParaTrough.basics.connectors.weather_connector
weather_connector;
      equation
      day_0=DayOfYear.u1;
      month_0=DayOfYear.u2;
      DayOfYear.n=WeatherEvaluator.n;
      DayOfYear.n=Day_from_n.u;
      DayOfYear.n=Month_from_n.u;

      StTime = time-to_hour(n-n_0);

      delta=declinationangle(n);
      omega=hourangle(SoTime);

      cosd(zenith) = cosd(delta)*cosd(data.latitude)*cosd(omega) +
sind(delta)*sind(data.latitude);

      alpha=if zenith>90 then 0 else 90-zenith;

      der(DNI_acc)=mega(DNI);

      DNI=weather_connector.DNI;
      AmbTemp=weather_connector.t;
      ws=weather_connector.ws;

      end Sun;
```

```
    end Models;

    package Examples
      model DaysComparison
        "Comparison of weather conditions in different types of days"

        Models.Sun FullySunnyDay(
          redeclare record data = Meteo.FullySunnyDay_140529,
          day_0=29,
          month_0=5);
        replaceable record data = Meteo.Sevilla;
        Models.Sun PartialCloudyDay(
          redeclare record data = Meteo.PartialCloudyDay_140417,
          day_0=17,
          month_0=4);
      end DaysComparison;

      model LocationComparison
        "Comparison of the weather conditions of different locations"

      //Parameters
      parameter Integer day_comp=1 "day to compare";
      parameter Integer month_comp=1 "month to compare";

      //SubModels
        Models.Sun Sevilla(
          redeclare record data = Meteo.Sevilla,
          day_0=day_comp,
          month_0=month_comp);
        replaceable record data = Meteo.Sevilla;
        Models.Sun Phoenix(
          redeclare record data = Meteo.Phoenix_EEUU,
          day_0=day_comp,
          month_0=month_comp);
        Models.Sun Madras(
          redeclare record data = Meteo.Madras_India,
          day_0=day_comp,
          month_0=month_comp);
      end LocationComparison;
    end Examples;
  end A_SolarResource;

  package B_HeatTransferFluid
    "Heat tranfer fluid system, including the Solar Field"

    import ParaTrough.basics.math.*;

    package Collectors
      record collector "generic collector"
      constant Area ApArea "Reflective Aperture Area";
      constant Length W "Collector aperture width";
      constant Length L "Length of collector assembly";
      constant Integer N_SCE "Number of modules SCE per Assembly";
      constant Integer N_HCE "Number of HCE per SCE";
      constant Length L_row "length of spacing between trouhgs";
      constant Length f "Average surface-to-focus path length";
      constant Efficiency TrackEff "Tracking Efficiency";
      constant Efficiency GeomEffects "Geometry effects";
      constant ReflectionCoefficient MirrReflectance "Mirror
Reflectance";
```

```
      constant Efficiency OptiEff "General Optical Efficiency";
      constant Real IAM0
          "Independent term coefficient of the Incidence Angle
Modifier ";
      constant Real IAM1
          "First degree coefficient of the Incidence Angle Modifier ";
      constant Real IAM2
          "Second degree coefficient of the Incidence Angle Modifier
";
      end collector;

      record EuroTrough_ET150 "EuroTrough ET150"
      extends collector(
ApArea=817.5,W=5.77,L=148.6,N_SCE=12,N_HCE=3,L_row=15,f=2.11,
TrackEff=0.99, GeomEffects=0.98, MirrReflectance=0.935, OptiEff=0.99,
IAM0=1,IAM1=0.506e-3,IAM2=-0.1763e-4);
      end EuroTrough_ET150;

      record Luz_LS2 "Luz LS-2"
      extends collector(
ApArea=235,W=5,L=49,N_SCE=6,N_HCE=3,L_row=15,f=1.8,TrackEff=0.99,
GeomEffects=0.98, MirrReflectance=0.935,
OptiEff=0.99,IAM0=1,IAM1=0.506e-3,IAM2=-0.1763e-4);
      end Luz_LS2;

      record Luz_LS3 "Luz LS-3"
      extends collector(
ApArea=545,W=5.75,L=100,N_SCE=12,N_HCE=3,L_row=15,f=2.11,
TrackEff=0.99, GeomEffects=0.98, MirrReflectance=0.935,
OptiEff=0.99,IAM0=1,IAM1=0.506e-3,IAM2=-0.1763e-4);
      end Luz_LS3;

      record Solargenix_SGX1 "Solargenix SGX-1"
      extends collector(
ApArea=470.3,W=5,L=100,N_SCE=12,N_HCE=3,L_row=15,f=1.8,
TrackEff=0.994, GeomEffects=0.98, MirrReflectance=0.935,
OptiEff=0.99,IAM0=1,IAM1=0.506e-3,IAM2=-0.1763e-4);
      end Solargenix_SGX1;

      record AlbiasaTrough_AT150 "AlbiasaTrough AT150"
      extends collector(
ApArea=817.5,W=5.774,L=150,N_SCE=12,N_HCE=3,L_row=15,f=2.11,
TrackEff=0.99, GeomEffects=0.98, MirrReflectance=0.935,
OptiEff=0.99,IAM0=1,IAM1=0.506e-3,IAM2=-0.1763e-4);
      end AlbiasaTrough_AT150;

      record SiemensSunField6 "Siemens SunField 6"
      extends collector(
ApArea=545,W=5,L=95.2,N_SCE=8,N_HCE=3,L_row=15,f=2.17, TrackEff=0.99,
GeomEffects=0.968, MirrReflectance=0.925, OptiEff=1,IAM0=1,IAM1=-
0.0753e-3,IAM2=-0.03698e-4);
      end SiemensSunField6;

      record SkyTrough "SkyFuel Sky Trough collector"
      extends collector(
ApArea=656,W=6,L=115,N_SCE=8,N_HCE=3,L_row=15,f=2.15, TrackEff=0.988,
GeomEffects=0.952, MirrReflectance=0.93,
OptiEff=1,IAM0=1,IAM1=0.0327e-3,IAM2=-0.1351e-4);
      end SkyTrough;
    end Collectors;
```

```
    package Receivers
      record receiver "Generic Receiver"
      constant Length L "Length of the receiver";
      constant Diameter Di "Inner Diameter";
      constant Diameter Do "Outer Diameter";
      constant Diameter Dg_i "Glass Envelope Inner Diameter";
      constant Diameter Dg_o "Glass Envelope Outer Diameter";
      constant Emissivity eps_gl "Emissivity of the borosilicate
envelope";
      constant ThermalConductivity k_gl
          "Thermal conductivity of the borosilicate envelope";
      constant Real HCEdust
          "Factor counting losses due to shading of HCE by dust on the
envelope";
      constant Real BelShad
          "Factor counting losses from shading of ends of HCEs due to
bellows";
      constant TransmissionCoefficient EnvTrans
          "Transmissivity of the glass envelope";
      constant Real HCEabs "absorbtivity of the HCE selective
coating";
      constant Real HCEmisc
          "miscellaneous factor to adjust for other HCE losses";
      constant Length k "wall roughness of the steel tube";
      constant basics.units.HeatFlowRate_m Qloss
          "Estimated average heat loss, in W/m";

      end receiver;

      record Schott_PTR70 "Shott PTR70"
      extends receiver(L=4, Di=0.066, Do=0.07,  Dg_i=0.115,
Dg_o=0.12, eps_gl=0.89, k_gl=1.1,
HCEdust=0.98,BelShad=0.96,EnvTrans=0.963, HCEabs=0.96, HCEmisc=0.96,
k=2.4e-5, Qloss=190);
      end Schott_PTR70;

      record Schott_PTR70_2008 "Shott PTR70 2008"
      extends receiver(L=4, Di=0.066, Do=0.07,  Dg_i=0.115,
Dg_o=0.12,eps_gl=0.89,
k_gl=1.1,HCEdust=0.98,BelShad=0.96,EnvTrans=0.963, HCEabs=0.95,
HCEmisc=0.96, k=2.4e-5,Qloss=150);
      end Schott_PTR70_2008;

      record Solel_UVAC3 "Solel UVAC 3"
      extends receiver(L=4, Di=0.066, Do=0.07,  Dg_i=0.115,
Dg_o=0.121,eps_gl=0.89,
k_gl=1.1,HCEdust=0.98,BelShad=0.971,EnvTrans=0.96, HCEabs=0.96,
HCEmisc=0.96, k=2.4e-5,Qloss=175);
      end Solel_UVAC3;

      record Siemens_UVAC2010 "Siemens UVAC 2010"
      extends receiver(L=4, Di=0.066, Do=0.07,  Dg_i=0.109,
Dg_o=0.115,eps_gl=0.89,
k_gl=1.1,HCEdust=1,BelShad=0.963,EnvTrans=0.965, HCEabs=0.96,
HCEmisc=0.96, k=2.4e-5,Qloss=192);
      end Siemens_UVAC2010;

      record Schott_PTR80 "Schott PTR80"
      extends receiver(L=4, Di=0.066, Do=0.08,  Dg_i=0.115,
Dg_o=0.12,eps_gl=0.89,
```

```
k_gl=1.1,HCEdust=0.98,BelShad=0.935,EnvTrans=0.964, HCEabs=0.963,
HCEmisc=0.96, k=2.4e-5,Qloss=190);
      end Schott_PTR80;
    end Receivers;

    package Models

      model Mirrors

        import ParaTrough.B_HeatTransferFluid.Collectors.*;

      replaceable record collector = EuroTrough_ET150;

      Angle_deg theta;
      Efficiency eta "Efficiency factor of the parabollic trough";

      protected
      Real RowShadow = if W_eff>collector.W then 1 else if W_eff<=0
then 0 else W_eff/collector.W
          "factor of shadowing effect against the mirrors";
      Real EndLoss=1-
collector.f*tand(theta)/collector.L*collector.N_SCE*collector.N_HCE
          "factor of end-pipe losses due to high incident angles";
      Real
IAM=collector.IAM0+collector.IAM1*theta+collector.IAM2*theta^2
          "Incidence Angle Modifier";
      Length W_eff=
collector.L_row*cosd(weather_connector1.zenith)/cosd(theta)
          "effective unshaded width of mirror aperture";

      public
        basics.connectors.weather_connector weather_connector1;
        basics.connectors.weather_connector weather_connector2;
      equation
      cosd(theta)=
sqrt(cosd(weather_connector1.zenith)^2+cosd(weather_connector1.delta)^
2*sind(weather_connector1.omega)^2);

eta=cosd(theta)*IAM*RowShadow*EndLoss*collector.TrackEff*collector.Geo
mEffects*collector.MirrReflectance*collector.OptiEff;

weather_connector2.DNI=weather_connector1.DNI*eta*collector.ApArea/col
lector.N_SCE/collector.N_HCE;

      weather_connector2.t=weather_connector1.t;
      weather_connector2.ws=weather_connector1.ws;
      weather_connector2.delta=weather_connector1.delta;
      weather_connector2.zenith=weather_connector1.zenith;
      weather_connector2.omega=weather_connector1.omega;
      weather_connector2.alpha=weather_connector1.alpha;

      end Mirrors;

      model HCE "1D HCE"
        import ParaTrough.B_HeatTransferFluid.Receivers.*;

      replaceable record receiver = Schott_PTR80;

      protected
      SurfaceCoefficientOfHeatTransfer
h_abs_HTF=522+478*pipe_heat_rad.hfPort1.m
```

```
          "Convection coefficient between HTF and absorber wall";
      ThermalConductivity
k_abs=14.8+0.0153*(receiverWall.heatPort1.t+receiverWall.heatPort2.t)/
2
          "Conduction coefficient of the absorber wall";
      Emissivity eps_abs=0.062+2e-
7*((receiverWall.heatPort1.t+receiverWall.heatPort2.t)/2)^2
          "Emissivity of the absorber";
      SurfaceCoefficientOfHeatTransfer h_amb=4.9+4.9*ambient.ws-
0.18*ambient.ws^2
          "Convection coefficient between borosilicate and the
ambient";

      Efficiency
eta_HCE=receiver.HCEdust*receiver.BelShad*receiver.EnvTrans*receiver.H
CEabs*receiver.HCEmisc
          "efficiency factor of the HCE";

      public
        basics.heat.h_cylinder convAmbient(        h=h_amb,
          D=receiver.Dg_o,
          L=receiver.L);
        basics.heat.k_annulus glassEnvelope(
          Do=receiver.Dg_o,
          Di=receiver.Dg_i,
          k=receiver.k_gl,
          L=receiver.L);
        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.heat.k_annulus receiverWall(
          k=k_abs,
          L=receiver.L,
          Do=receiver.Do,
          Di=receiver.Di);
        basics.heat.h_cylinder convPipe(
          h=h_abs_HTF,
          D=receiver.Di,
          L=receiver.L);

        basics.generic_heat.Ambient ambient;
        basics.connectors.weather_connector weather_connector1;
        basics.hydraulics_heat.pipe_heat_rad pipe_heat_rad(
          L=receiver.L,
          D=receiver.Di,
          k=receiver.k);
        basics.generic_heat.IrradiationEfficiency HCEefficiency(

eta=eta_HCE);
        basics.heat.r_cylinder radSky(
          D=receiver.Dg_o,
          L=receiver.L,
          epsilon=receiver.eps_gl);
        basics.heat.r_annulus VacuumChamber(
          Do=receiver.Dg_o,
          Di=receiver.Do,
          epsilon_o=receiver.eps_gl,
          L=receiver.L,
          epsilon_i=eps_abs);

      equation
        connect(glassEnvelope.heatPort2, convAmbient.heatPort1);
```

```
        connect(convAmbient.heatPort2, ambient.heatPort1);
        connect(weather_connector1, ambient.weather_connector);
        connect(receiverWall.heatPort1, convPipe.heatPort2);
        connect(convPipe.heatPort1, pipe_heat_rad.heatPort);
        connect(hfPort1, pipe_heat_rad.hfPort1);
        connect(pipe_heat_rad.hfPort2, hfPort2);
        connect(weather_connector1, HCEefficiency.weather_connector1);
        connect(HCEefficiency.weather_connector2,
pipe_heat_rad.weather_connector);
        connect(glassEnvelope.heatPort2, radSky.heatPort1);
        connect(radSky.heatPort2, ambient.heatPort2);
        connect(receiverWall.heatPort2, VacuumChamber.heatPort1);
        connect(VacuumChamber.heatPort2, glassEnvelope.heatPort1);
      end HCE;

      model SCE "Solar Collector Element"
        import ParaTrough;

        basics.connectors.weather_connector weather_connector;
        ParaTrough.B_HeatTransferFluid.Models.Mirrors Mirrors;
        HCE hCE1;
        HCE hCE2;
        HCE hCE3;
        ParaTrough.basics.connectors.hfPort hfPort2;
        ParaTrough.basics.connectors.hfPort hfPort1;
      equation

        connect(hCE1.hfPort1, hfPort1);
        connect(hCE1.hfPort2, hCE2.hfPort1);
        connect(hCE2.hfPort2, hCE3.hfPort1);
        connect(hCE3.hfPort2, hfPort2);
        connect(hfPort1, hfPort1);
        connect(weather_connector, Mirrors.weather_connector1);
        connect(Mirrors.weather_connector2, hCE3.weather_connector1);
        connect(hCE2.weather_connector1, hCE3.weather_connector1);
        connect(hCE1.weather_connector1, hCE2.weather_connector1);
      end SCE;

      model SCA "Solar Collector Assembly"
        import ParaTrough.B_HeatTransferFluid.Collectors.*;

      replaceable record collector=EuroTrough_ET150;

      SCE sCE[collector.N_SCE];

      CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
          "Central Temperature of the SCA";

        basics.connectors.weather_connector weather_connector;
        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;

      equation

      for i in 1:collector.N_SCE-1 loop
      connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
      connect(hfPort1,sCE[1].hfPort1);
      connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
```

```
      connect(sCE[collector.N_SCE].hfPort2,hfPort2);
      end for;

      end SCA;

      model Loop "Loop composed by 4 SCAs"

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        Models.SCA sCA1;
        Models.SCA sCA2;
        Models.SCA sCA3;
        Models.SCA sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy_kWh E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);
      der(E)=to_kWh(ThermalPower);

      end Loop;

      model SolarField "Simplified Solar Field model"
      import ParaTrough.B_HeatTransferFluid.Collectors.*;
      import ParaTrough.B_HeatTransferFluid.Receivers.*;
      import ParaTrough.media.*;

      replaceable record collector = EuroTrough_ET150;
      replaceable record receiver = Schott_PTR80;
      replaceable package Medium=DowthermA;
      Medium.BaseProperties medium1;
      Medium.BaseProperties medium2;

      parameter Integer N=152 "Number of loops";
      parameter Efficiency CF=0.98 "Cleanliness factor";

      HeatFlowRate Q_abs "Heat absorbed by the Solar Field";
      basics.units.Energy_MWh E "Accumulated energy of a loop";

      Efficiency eta_SOF "Solar field performance factor";
      MassFlowRate m "Mass flow";
      CelsiusTemperature t_in( start=293)=hfPort1.t "Inlet
temperature";
```

```
      CelsiusTemperature t_out( start=393)=hfPort2.t "Outlet
temperature";

      Velocity v=m/medium2.d/A "velocity of the fluid along the pipe";
      Pressure dp=hfPort1.p-hfPort2.p "pressure loss in Solar Field";

      protected
      CoefficientOfFriction f( start=1) "coefficient of friction";
      CrossSection A=pi/4*receiver.Di^2 "Cross section of the pipe";
      ReynoldsNumber Re=v*receiver.Di*medium2.d/medium2.eta "Reynolds
number";
      HeatFlux Qloss=receiver.Qloss/collector.f
          "Estimated average heat losses in the HCEs";
      Enthalpy H1=hfPort1.m*medium1.h "Inlet enthaply";
      Enthalpy H2=-hfPort2.m*medium2.h "Outlet enthalpy";
      Efficiency
eta=cosd(theta)*IAM*RowShadow*EndLoss*collector.TrackEff*collector.Geo
mEffects*collector.MirrReflectance*collector.OptiEff
          "Efficiency factor of the mirrors";
      Efficiency
eta_HCE=receiver.HCEdust*receiver.BelShad*receiver.EnvTrans*receiver.H
CEabs*receiver.HCEmisc
          "Efficiency factor of the HCEs";
      Efficiency eta_prima=eta*eta_HCE
          "Efficiency factor of the SCAs without taking into account
thermal losses in the HCEs";
      Angle_deg theta "Incidence angle";
      Integer N_coll=N*4 "Total number of collectors";

      Real
IAM=collector.IAM0+collector.IAM1*theta+collector.IAM2*theta^2
          "Incidence Angle Modifier";
      Real RowShadow = if W_eff>collector.W then 1 else if W_eff<=0
then 0 else W_eff/collector.W
          "factor of shadowing effect against the mirrors";
      Real EndLoss=1-collector.f*tand(theta)/collector.L
          "factor of end-pipe losses due to high incident angles";
      Length W_eff=
collector.L_row*cosd(weather_connector.zenith)/cosd(theta)
          "effective (unshaded width of mirror aperture";

      equation
      Q_abs=eta_SOF*weather_connector.DNI*collector.ApArea*N_coll;
      cosd(theta)=
sqrt(cosd(weather_connector.zenith)^2+cosd(weather_connector.delta)^2*
sind(weather_connector.omega)^2);

      eta_SOF=if Qloss>eta_prima*CF*weather_connector.DNI then 0 else
eta_prima*CF-Qloss/(weather_connector.DNI);

      H2-H1=Q_abs;
      der(E)=basics.math.to_MWh(Q_abs);

      dp=f*receiver.L*v^2*medium2.d/(2*receiver.Di)
          "Equation of Darcy-Weisbach";

      f=if Re<=2300 then 64/Re else (1/(-
2*log10(receiver.k/receiver.Di/3.7+2.51/Re/sqrt(f))))^2
          "Equation of Colebrook-White";

      medium1.T_degC=hfPort1.t;
```

```
      medium1.p=hfPort1.p;

      medium2.T_degC=hfPort2.t;
      medium2.p=hfPort2.p;

      m=-hfPort2.m;
      m=hfPort1.m;

      public
        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
      end SolarField;

      model SolarField_tControl
        "Simplified solar fielld model with temperature control"
        SolarField solarField;
        basics.hydraulics_heat.pump pump;
        basics.instruments.temperature_sensor temperature_sensor;
        basics.control.LimPID_indirect PID(
          yMax=5000,
          yMin=200,
          k=25,
          Td=10,
          controllerType=Modelica.Blocks.Types.SimpleController.PI,
          Ti=0.05);
        basics.connectors.weather_connector weather_connector;
        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.analog_input analog_input;
      equation
        connect(solarField.hfPort2, temperature_sensor.hfPort);
        connect(pump.hfPort2, solarField.hfPort1);
        connect(PID.y, pump.analog_input);
        connect(temperature_sensor.analog_output, PID.u_m);
        connect(solarField.weather_connector, weather_connector);
        connect(hfPort1, hfPort1);
        connect(analog_input, PID.u_s);
        connect(hfPort1, pump.hfPort1);
        connect(solarField.hfPort2, hfPort2);
        connect(hfPort2, hfPort2);
      end SolarField_tControl;
    end Models;

    package Mirrors
      model EuroTrough_ET150
      extends Models.Mirrors( redeclare record collector =
              Collectors.EuroTrough_ET150);
      equation

      end EuroTrough_ET150;

      model Luz_LS2
      extends Models.Mirrors( redeclare record
collector=Collectors.Luz_LS2);
      equation

      end Luz_LS2;

      model Luz_LS3
```

```
      extends Models.Mirrors( redeclare record
collector=Collectors.Luz_LS3);
      equation

      end Luz_LS3;

      model Solargenix_SGX1
      extends Models.Mirrors( redeclare record collector =
              Collectors.Solargenix_SGX1);
      equation

      end Solargenix_SGX1;

      model AlbiasaTrough_AT150
      extends Models.Mirrors( redeclare record collector =
              Collectors.AlbiasaTrough_AT150);
      equation

      end AlbiasaTrough_AT150;

      model SiemensSunField6
      extends Models.Mirrors( redeclare record collector =
              Collectors.SiemensSunField6);
      equation

      end SiemensSunField6;

      model SkyTrough
      extends Models.Mirrors( redeclare record collector =
              Collectors.SkyTrough);
      equation

      end SkyTrough;
    end Mirrors;

    package HCEs
      model Schott_PTR70
      extends Models.HCE( redeclare record
receiver=Receivers.Schott_PTR70);

      end Schott_PTR70;

      model Schott_PTR70_2008
      extends Models.HCE( redeclare record
receiver=Receivers.Schott_PTR70_2008);

      end Schott_PTR70_2008;

      model Solel_UVAC3
      extends Models.HCE( redeclare record
receiver=Receivers.Solel_UVAC3);

      end Solel_UVAC3;

      model Siemens_UVAC2010
      extends Models.HCE( redeclare record
receiver=Receivers.Siemens_UVAC2010);

      end Siemens_UVAC2010;

      model Schott_PTR80
```

```
      extends Models.HCE( redeclare record
receiver=Receivers.Schott_PTR80);

      end Schott_PTR80;
    end HCEs;

    package SCEs
      package EuroTrough_ET150
        model ET150_PTR70
          "SCE composed by mirrors of type EuroTrough ET150 and HCEs
of type Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.EuroTrough_ET150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end ET150_PTR70;

        model ET150_PTR70_2008
          "SCE composed by mirrors of type EuroTrough ET150 and HCEs
of type Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.EuroTrough_ET150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end ET150_PTR70_2008;
```

```
        model ET150_UVAC3
          "SCE composed by mirrors of type EuroTrough ET150 and HCEs
of type Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.EuroTrough_ET150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end ET150_UVAC3;

        model ET150_UVAC2010
          "SCE composed by mirrors of type EuroTrough_ET150 and HCEs
of type Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.EuroTrough_ET150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end ET150_UVAC2010;

        model ET150_PTR80
          "SCE composed by mirrors of type EuroTrough_ET150 and HCEs
of type Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
```

```
          ParaTrough.B_HeatTransferFluid.Mirrors.EuroTrough_ET150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end ET150_PTR80;
      end EuroTrough_ET150;

      package Luz_LS2
        model LS2_PTR70
          "SCE composed by mirrors of type Luz LS2 and HCEs of type
Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS2 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS2_PTR70;

        model LS2_PTR70_2008
          "SCE composed by mirrors of type Luz LS2 and HCEs of type
Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS2 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
```

```
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS2_PTR70_2008;

        model LS2_UVAC3
          "SCE composed by mirrors of type Luz LS2 and HCEs of type
Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS2 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS2_UVAC3;

        model LS2_UVAC2010
          "SCE composed by mirrors of type Luz LS2 and HCEs of type
Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS2 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
```

```
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS2_UVAC2010;

        model LS2_PTR80
          "SCE composed by mirrors of type Luz LS2 and HCEs of type
Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS2 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS2_PTR80;
      end Luz_LS2;

      package Luz_LS3
        model LS3_PTR70
          "SCE composed by mirrors of type Luz LS3 and HCEs of type
Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS3 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS3_PTR70;

        model LS3_PTR70_2008
```

```
          "SCE composed by mirrors of type Luz LS3 and HCEs of type
Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS3 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS3_PTR70_2008;

        model LS3_UVAC3
          "SCE composed by mirrors of type Luz LS3 and HCEs of type
Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS3 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS3_UVAC3;

        model LS3_UVAC2010
          "SCE composed by mirrors of type Luz LS3 and HCEs of type
Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS3 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
```

```
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS3_UVAC2010;

        model LS3_PTR80
          "SCE composed by mirrors of type Luz LS3 and HCEs of type
Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Luz_LS3 Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end LS3_PTR80;
      end Luz_LS3;

      package Solargenix_SGX1
        model SGX1_PTR70
          "SCE composed by mirrors of type Solargenix SGX1 and HCEs of
type Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Solargenix_SGX1
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
```

```
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SGX1_PTR70;

        model SGX1_PTR70_2008
          "SCE composed by mirrors of type Solargenix SGX1 and HCEs of
type Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Solargenix_SGX1
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SGX1_PTR70_2008;

        model SGX1_UVAC3
          "SCE composed by mirrors of type Solargenix SGX1 and HCEs of
type Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Solargenix_SGX1
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
```

```
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SGX1_UVAC3;

        model SGX1_UVAC2010
          "SCE composed by mirrors of type Solargenix SGX1 and HCEs of
type Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Solargenix_SGX1
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SGX1_UVAC2010;

        model SGX1_PTR80
          "SCE composed by mirrors of type Solargenix SGX1 and HCEs of
type Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.Solargenix_SGX1
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SGX1_PTR80;
      end Solargenix_SGX1;

      package AlbiasaTrough_AT150
        model AT150_PTR70
```

```
          "SCE composed by mirrors of type AlbiasaTrough AT150 and
HCEs of type Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.AlbiasaTrough_AT150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end AT150_PTR70;

        model AT150_PTR70_2008
          "SCE composed by mirrors of type AlbiasaTrough AT150 and
HCEs of type Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.AlbiasaTrough_AT150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end AT150_PTR70_2008;

        model AT150_UVAC3
          "SCE composed by mirrors of type AlbiasaTrough AT150 and
HCEs of type Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.AlbiasaTrough_AT150
Mirrors;
```

```
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end AT150_UVAC3;

        model AT150_UVAC2010
          "SCE composed by mirrors of type AlbiasaTrough AT150 and
HCEs of type Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.AlbiasaTrough_AT150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end AT150_UVAC2010;

        model AT150_PTR80
          "SCE composed by mirrors of type AlbiasaTrough AT150 and
HCEs of type Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.AlbiasaTrough_AT150
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation
```

```
          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end AT150_PTR80;
      end AlbiasaTrough_AT150;

      package SiemensSunField6
        model SF6_PTR70
          "SCE composed by mirrors of type SiemensSunField6 and HCEs
of type Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SiemensSunField6
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SF6_PTR70;

        model SF6_PTR70_2008
          "SCE composed by mirrors of type SiemensSunField6 and HCEs
of type Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SiemensSunField6
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
```

```
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SF6_PTR70_2008;

        model SF6_UVAC3
          "SCE composed by mirrors of type SiemensSunField6 and HCEs
of type Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SiemensSunField6
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SF6_UVAC3;

        model SF6_UVAC2010
          "SCE composed by mirrors of type SiemensSunField6 and HCEs
of type Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SiemensSunField6
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SF6_UVAC2010;
```

```
        model SF6_PTR80
          "SCE composed by mirrors of type SiemensSunField6 and HCEs
of type Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SiemensSunField6
Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end SF6_PTR80;
      end SiemensSunField6;

      package SkyTrough
        model Sky_PTR70
          "SCE composed by mirrors of type SkyTrough and HCEs of type
Schott PTR70"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SkyTrough Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end Sky_PTR70;

        model Sky_PTR70_2008
          "SCE composed by mirrors of type SkyTrough and HCEs of type
Schott PTR70 2008"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
```

```
          ParaTrough.B_HeatTransferFluid.Mirrors.SkyTrough Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR70_2008 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end Sky_PTR70_2008;

        model Sky_UVAC3
          "SCE composed by mirrors of type SkyTrough and HCEs of type
Solel UVAC3"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SkyTrough Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Solel_UVAC3 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end Sky_UVAC3;

        model Sky_UVAC2010
          "SCE composed by mirrors of type SkyTrough and HCEs of type
Siemens UVAC 2010"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SkyTrough Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Siemens_UVAC2010 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
```

```
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end Sky_UVAC2010;

        model Sky_PTR80
          "SCE composed by mirrors of type SkyTrough and HCEs of type
Schott PTR80"
          import ParaTrough;

          basics.connectors.weather_connector weather_connector;
          ParaTrough.B_HeatTransferFluid.Mirrors.SkyTrough Mirrors;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE1;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE2;
          ParaTrough.B_HeatTransferFluid.HCEs.Schott_PTR80 hCE3;
          ParaTrough.basics.connectors.hfPort hfPort2;
          ParaTrough.basics.connectors.hfPort hfPort1;
        equation

          connect(hCE1.hfPort1, hfPort1);
          connect(hCE1.hfPort2, hCE2.hfPort1);
          connect(hCE2.hfPort2, hCE3.hfPort1);
          connect(hCE3.hfPort2, hfPort2);
          connect(hfPort1, hfPort1);
          connect(weather_connector, Mirrors.weather_connector1);
          connect(Mirrors.weather_connector2,
hCE3.weather_connector1);
          connect(hCE2.weather_connector1, hCE3.weather_connector1);
          connect(hCE1.weather_connector1, hCE2.weather_connector1);
        end Sky_PTR80;
      end SkyTrough;
    end SCEs;

    package SCAs
      package EuroTrough_ET150
        model ET150_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=EuroTrough_ET150;

        SCEs.EuroTrough_ET150.ET150_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);
```

```
connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end ET150_PTR70;

        model ET150_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=EuroTrough_ET150;

        SCEs.EuroTrough_ET150.ET150_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end ET150_PTR70_2008;

        model ET150_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=EuroTrough_ET150;

        SCEs.EuroTrough_ET150.ET150_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
```

```
        end for;

        end ET150_UVAC3;

        model ET150_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=EuroTrough_ET150;

        SCEs.EuroTrough_ET150.ET150_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end ET150_UVAC2010;

        model ET150_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=EuroTrough_ET150;

        SCEs.EuroTrough_ET150.ET150_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end ET150_PTR80;
      end EuroTrough_ET150;
```

```
      package Luz_LS2
        model LS2_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS2;

        SCEs.Luz_LS2.LS2_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS2_PTR70;

        model LS2_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS2;

        SCEs.Luz_LS2.LS2_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS2_PTR70_2008;

        model LS2_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS2;
```

```
        SCEs.Luz_LS2.LS2_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS2_UVAC3;

        model LS2_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS2;

        SCEs.Luz_LS2.LS2_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS2_UVAC2010;

        model LS2_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS2;

        SCEs.Luz_LS2.LS2_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
```

```
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS2_PTR80;
      end Luz_LS2;

      package Luz_LS3
        model LS3_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS3;

        SCEs.Luz_LS3.LS3_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS3_PTR70;

        model LS3_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS3;

        SCEs.Luz_LS3.LS3_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
```

```
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS3_PTR70_2008;

        model LS3_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS3;

        SCEs.Luz_LS3.LS3_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS3_UVAC3;

        model LS3_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS3;

        SCEs.Luz_LS3.LS3_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation
```

```
        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS3_UVAC2010;

        model LS3_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Luz_LS3;

        SCEs.Luz_LS3.LS3_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end LS3_PTR80;
      end Luz_LS3;

      package Solargenix_SGX1
        model SGX1_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Solargenix_SGX1;

        SCEs.Solargenix_SGX1.SGX1_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);
```

```
connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SGX1_PTR70;

        model SGX1_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Solargenix_SGX1;

        SCEs.Solargenix_SGX1.SGX1_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SGX1_PTR70_2008;

        model SGX1_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Solargenix_SGX1;

        SCEs.Solargenix_SGX1.SGX1_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
```

```
        end for;

        end SGX1_UVAC3;

        model SGX1_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Solargenix_SGX1;

        SCEs.Solargenix_SGX1.SGX1_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SGX1_UVAC2010;

        model SGX1_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=Solargenix_SGX1;

        SCEs.Solargenix_SGX1.SGX1_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SGX1_PTR80;
      end Solargenix_SGX1;
```

```
      package AlbiasaTrough_AT150
        model AT150_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=AlbiasaTrough_AT150;

        SCEs.AlbiasaTrough_AT150.AT150_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end AT150_PTR70;

        model AT150_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=AlbiasaTrough_AT150;

        SCEs.AlbiasaTrough_AT150.AT150_PTR70_2008
sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end AT150_PTR70_2008;

        model AT150_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;
```

```
        record collector=AlbiasaTrough_AT150;

        SCEs.AlbiasaTrough_AT150.AT150_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end AT150_UVAC3;

        model AT150_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=AlbiasaTrough_AT150;

        SCEs.AlbiasaTrough_AT150.AT150_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end AT150_UVAC2010;

        model AT150_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=AlbiasaTrough_AT150;

        SCEs.AlbiasaTrough_AT150.AT150_PTR80 sCE[collector.N_SCE];
```

```
        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end AT150_PTR80;
      end AlbiasaTrough_AT150;

      package SiemensSunField6
        model SF6_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SiemensSunField6;

        SCEs.SiemensSunField6.SF6_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SF6_PTR70;

        model SF6_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SiemensSunField6;

        SCEs.SiemensSunField6.SF6_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";
```

```
          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SF6_PTR70_2008;

        model SF6_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SiemensSunField6;

        SCEs.SiemensSunField6.SF6_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SF6_UVAC3;

        model SF6_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SiemensSunField6;

        SCEs.SiemensSunField6.SF6_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;
```

```
        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SF6_UVAC2010;

        model SF6_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SiemensSunField6;

        SCEs.SiemensSunField6.SF6_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end SF6_PTR80;
      end SiemensSunField6;

      package SkyTrough
        model Sky_PTR70 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SkyTrough;

        SCEs.SkyTrough.Sky_PTR70 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
```

```
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end Sky_PTR70;

        model Sky_PTR70_2008 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SkyTrough;

        SCEs.SkyTrough.Sky_PTR70_2008 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end Sky_PTR70_2008;

        model Sky_UVAC3 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SkyTrough;

        SCEs.SkyTrough.Sky_UVAC3 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
```

```
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end Sky_UVAC3;

        model Sky_UVAC2010 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SkyTrough;

        SCEs.SkyTrough.Sky_UVAC2010 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end Sky_UVAC2010;

        model Sky_PTR80 "Solar Collector Assembly"
          import ParaTrough.B_HeatTransferFluid.Collectors.*;

        record collector=SkyTrough;

        SCEs.SkyTrough.Sky_PTR80 sCE[collector.N_SCE];

        CelsiusTemperature
t=sCE[integer(collector.N_SCE/2)].hCE3.hfPort2.t
            "Central Temperature of the SCA";

          basics.connectors.weather_connector weather_connector;
          basics.connectors.hfPort hfPort1;
          basics.connectors.hfPort hfPort2;

        equation

        for i in 1:collector.N_SCE-1 loop
        connect(weather_connector,sCE[i].weather_connector);

connect(weather_connector,sCE[collector.N_SCE].weather_connector);
        connect(hfPort1,sCE[1].hfPort1);
        connect(sCE[i].hfPort2,sCE[i+1].hfPort1);
        connect(sCE[collector.N_SCE].hfPort2,hfPort2);
        end for;

        end Sky_PTR80;
      end SkyTrough;
```

```
    end SCAs;

    package Loops
      model EuroTrough_ET150

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        SCAs.EuroTrough_ET150.ET150_PTR70 sCA1;
        SCAs.EuroTrough_ET150.ET150_PTR70 sCA2;
        SCAs.EuroTrough_ET150.ET150_PTR70 sCA3;
        SCAs.EuroTrough_ET150.ET150_PTR70 sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);

      der(E)=ThermalPower;

      end EuroTrough_ET150;

      model Luz_LS2

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        SCAs.Luz_LS2.LS2_PTR70 sCA1;
        SCAs.Luz_LS2.LS2_PTR70 sCA2;
        SCAs.Luz_LS2.LS2_PTR70 sCA3;
        SCAs.Luz_LS2.LS2_PTR70 sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
```

```
      connect(weather_connector, sCA1.weather_connector);
      connect(sCA1.weather_connector, sCA2.weather_connector);
      connect(sCA2.weather_connector, sCA3.weather_connector);
      connect(sCA3.weather_connector, sCA4.weather_connector);
      connect(hfPort1, sCA1.hfPort1);
      connect(sCA1.hfPort2, sCA2.hfPort1);
      connect(sCA2.hfPort2, sCA3.hfPort1);
      connect(sCA3.hfPort2, sCA4.hfPort1);
      connect(sCA4.hfPort2, hfPort2);

    der(E)=ThermalPower;

    end Luz_LS2;

    model Luz_LS3

      basics.connectors.hfPort hfPort1;
      basics.connectors.hfPort hfPort2;
      basics.connectors.weather_connector weather_connector;
      SCAs.Luz_LS3.LS3_PTR70 sCA1;
      SCAs.Luz_LS3.LS3_PTR70 sCA2;
      SCAs.Luz_LS3.LS3_PTR70 sCA3;
      SCAs.Luz_LS3.LS3_PTR70 sCA4;

    CelsiusTemperature t_in=hfPort1.t;
    CelsiusTemperature t_out=hfPort2.t;

    HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
        "Thermal power of the loop";
    Energy E "Accumulated energy of a loop";

    equation
      connect(hfPort1, hfPort1);
      connect(hfPort2, hfPort2);
      connect(weather_connector, sCA1.weather_connector);
      connect(sCA1.weather_connector, sCA2.weather_connector);
      connect(sCA2.weather_connector, sCA3.weather_connector);
      connect(sCA3.weather_connector, sCA4.weather_connector);
      connect(hfPort1, sCA1.hfPort1);
      connect(sCA1.hfPort2, sCA2.hfPort1);
      connect(sCA2.hfPort2, sCA3.hfPort1);
      connect(sCA3.hfPort2, sCA4.hfPort1);
      connect(sCA4.hfPort2, hfPort2);

    der(E)=ThermalPower;

    end Luz_LS3;

    model Solargenix_SGX1

      basics.connectors.hfPort hfPort1;
      basics.connectors.hfPort hfPort2;
      basics.connectors.weather_connector weather_connector;
      SCAs.Solargenix_SGX1.SGX1_PTR70 sCA1;
      SCAs.Solargenix_SGX1.SGX1_PTR70 sCA2;
      SCAs.Solargenix_SGX1.SGX1_PTR70 sCA3;
      SCAs.Solargenix_SGX1.SGX1_PTR70 sCA4;

    CelsiusTemperature t_in=hfPort1.t;
```

```
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);

      der(E)=ThermalPower;

      end Solargenix_SGX1;

      model AlbiasaTrough_AT150

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        SCAs.AlbiasaTrough_AT150.AT150_PTR70 sCA1;
        SCAs.AlbiasaTrough_AT150.AT150_PTR70 sCA2;
        SCAs.AlbiasaTrough_AT150.AT150_PTR70 sCA3;
        SCAs.AlbiasaTrough_AT150.AT150_PTR70 sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);

      der(E)=ThermalPower;

      end AlbiasaTrough_AT150;
```

```
      model SiemensSunField6

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        SCAs.SiemensSunField6.SF6_PTR70 sCA1;
        SCAs.SiemensSunField6.SF6_PTR70 sCA2;
        SCAs.SiemensSunField6.SF6_PTR70 sCA3;
        SCAs.SiemensSunField6.SF6_PTR70 sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);

      der(E)=ThermalPower;

      end SiemensSunField6;

      model SkyTrough

        basics.connectors.hfPort hfPort1;
        basics.connectors.hfPort hfPort2;
        basics.connectors.weather_connector weather_connector;
        SCAs.SkyTrough.Sky_PTR70 sCA1;
        SCAs.SkyTrough.Sky_PTR70 sCA2;
        SCAs.SkyTrough.Sky_PTR70 sCA3;
        SCAs.SkyTrough.Sky_PTR70 sCA4;

      CelsiusTemperature t_in=hfPort1.t;
      CelsiusTemperature t_out=hfPort2.t;

      HeatFlowRate
ThermalPower=hfPort1.m*(sCA4.sCE[1].hCE3.pipe_heat_rad.H2-
sCA1.sCE[sCA1.collector.N_SCE].hCE1.pipe_heat_rad.H1)
          "Thermal power of the loop";
      Energy E "Accumulated energy of a loop";

      equation
        connect(hfPort1, hfPort1);
        connect(hfPort2, hfPort2);
        connect(weather_connector, sCA1.weather_connector);
        connect(sCA1.weather_connector, sCA2.weather_connector);
        connect(sCA2.weather_connector, sCA3.weather_connector);
```

```
        connect(sCA3.weather_connector, sCA4.weather_connector);
        connect(hfPort1, sCA1.hfPort1);
        connect(sCA1.hfPort2, sCA2.hfPort1);
        connect(sCA2.hfPort2, sCA3.hfPort1);
        connect(sCA3.hfPort2, sCA4.hfPort1);
        connect(sCA4.hfPort2, hfPort2);

      der(E)=ThermalPower;

      end SkyTrough;
    end Loops;

    package Examples

      model Loop_comparison "Loop operation"

        Loops.EuroTrough_ET150 euroTrough_ET150_1;
        Loops.Luz_LS2 luz_LS2_1;
        Loops.Luz_LS3 luz_LS3_1;
        Loops.Solargenix_SGX1 solargenix_SGX1_1;
        Loops.AlbiasaTrough_AT150 albiasaTrough_AT150_1;
        Loops.SiemensSunField6 siemensSunField6_1;
        Loops.SkyTrough skyTrough;
        basics.hydraulics_heat.source_m_t source_m_t1;
        basics.hydraulics_heat.source_m_t source_m_t2;
        basics.hydraulics_heat.source_m_t source_m_t3;
        basics.hydraulics_heat.source_m_t source_m_t4;
        basics.hydraulics_heat.source_m_t source_m_t5;
        basics.hydraulics_heat.source_m_t source_m_t6;
        basics.hydraulics_heat.source_m_t source_m_t7;
        basics.hydraulics_heat.sink_p sink_p1(p=10e5);
        basics.hydraulics_heat.sink_p sink_p2(p=10e5);
        basics.hydraulics_heat.sink_p sink_p3(p=10e5);
        basics.hydraulics_heat.sink_p sink_p4(p=10e5);
        basics.hydraulics_heat.sink_p sink_p5(p=10e5);
        basics.hydraulics_heat.sink_p sink_p6(p=10e5);
        basics.hydraulics_heat.sink_p sink_p7(p=10e5);
        A_SolarResource.Models.Sun sun;
      equation
        connect(sun.weather_connector,
euroTrough_ET150_1.weather_connector);
        connect(luz_LS2_1.weather_connector, sun.weather_connector);
        connect(luz_LS3_1.weather_connector, sun.weather_connector);
        connect(sun.weather_connector,
solargenix_SGX1_1.weather_connector);
        connect(siemensSunField6_1.weather_connector,
sun.weather_connector);
        connect(skyTrough.weather_connector, sun.weather_connector);
        connect(albiasaTrough_AT150_1.weather_connector,
sun.weather_connector);
        connect(source_m_t1.hfPort, euroTrough_ET150_1.hfPort1);
        connect(sink_p1.hfPort, euroTrough_ET150_1.hfPort2);
        connect(source_m_t2.hfPort, luz_LS2_1.hfPort1);
        connect(sink_p2.hfPort, luz_LS2_1.hfPort2);
        connect(source_m_t3.hfPort, luz_LS3_1.hfPort1);
        connect(sink_p3.hfPort, luz_LS3_1.hfPort2);
        connect(source_m_t4.hfPort, solargenix_SGX1_1.hfPort1);
        connect(sink_p4.hfPort, solargenix_SGX1_1.hfPort2);
        connect(source_m_t5.hfPort, albiasaTrough_AT150_1.hfPort1);
        connect(sink_p5.hfPort, albiasaTrough_AT150_1.hfPort2);
        connect(source_m_t6.hfPort, siemensSunField6_1.hfPort1);
```

```
    connect(sink_p6.hfPort, siemensSunField6_1.hfPort2);
    connect(source_m_t7.hfPort, skyTrough.hfPort1);
    connect(sink_p7.hfPort, skyTrough.hfPort2);
  end Loop_comparison;

  model Loop_operation "Loop operation"

    A_SolarResource.Models.Sun sun(
      redeclare record data = data,
      month_0=5,
      day_0=29);
    Models.Loop Loop;
    basics.hydraulics_heat.pump pump(
                                m_const=8);
    basics.hydraulics_heat.sink_p sink_hf;
    basics.hydraulics_heat.source_t source_hf_t;
    replaceable record data =
A_SolarResource.Meteo.FullySunnyDay_140529;
  equation
    connect(sink_hf.hfPort, Loop.hfPort2);
    connect(sun.weather_connector, Loop.weather_connector);
    connect(source_hf_t.hfPort, pump.hfPort1);
    connect(pump.hfPort2, Loop.hfPort1);
  end Loop_operation;

  model SOF_tempControl "Solar Field with a temperature control"

    Models.SolarField solarField;
    basics.hydraulics_heat.pump pump;
    basics.instruments.temperature_sensor temperature_sensor;
    basics.control.LimPID_indirect PID(
      yMax=5000,
      yMin=200,
      k=25,
      Td=10,
      controllerType=Modelica.Blocks.Types.SimpleController.PI,
      Ti=0.05);
    Modelica.Blocks.Sources.RealExpression realExpression(y=393);
    A_SolarResource.Models.Sun sun(
      redeclare record data = data,
      day_0=29,
      month_0=5);
    basics.hydraulics_heat.source_t source_t;
    basics.hydraulics_heat.sink_p sink_p;
    replaceable record data =
A_SolarResource.Meteo.FullySunnyDay_140529;
  equation
    connect(pump.hfPort2, solarField.hfPort1);
    connect(solarField.hfPort2, temperature_sensor.hfPort);
    connect(realExpression.y, PID.u_s);
    connect(PID.y, pump.analog_input);
    connect(PID.u_m, temperature_sensor.analog_output);
    connect(sun.weather_connector, solarField.weather_connector);
    connect(source_t.hfPort, pump.hfPort1);
    connect(sink_p.hfPort, temperature_sensor.hfPort);
  end SOF_tempControl;

  model SOF_location_comparison

    Models.SolarField_tControl SOF;
    Modelica.Blocks.Sources.RealExpression realExpression(y=393);
```

```
        basics.hydraulics_heat.source_t source_t1;
        basics.hydraulics_heat.sink_p sink_p1(p=10e5);
        replaceable record data = A_SolarResource.Meteo.Phoenix_EEUU;
        A_SolarResource.Models.Sun sun(
          day_0=1,
          redeclare record data = data,
          month_0=5);
      equation
        connect(realExpression.y, SOF.analog_input);
        connect(sun.weather_connector, SOF.weather_connector);
        connect(SOF.hfPort2, sink_p1.hfPort);
        connect(source_t1.hfPort, SOF.hfPort1);
      end SOF_location_comparison;
    end Examples;
  end B_HeatTransferFluid;


end ParaTrough;

partial package Modelica.Media.Interfaces.PartialMedium
  "Partial medium properties (base package of all media packages)"

  import SI = Modelica.SIunits;
  extends Modelica.Icons.Library;

  // Constants to be set in Medium
  constant String mediumName = "unusablePartialMedium" "Name of the
medium";
  constant String substanceNames[:]={mediumName}
    "Names of the mixture substances. Set substanceNames={mediumName}
if only one substance.";
  constant String extraPropertiesNames[:]=fill("", 0)
    "Names of the additional (extra) transported properties. Set
extraPropertiesNames=fill(\"\",0) if unused";
  constant Boolean singleState
    "= true, if u and d are not a function of pressure";
  constant Boolean reducedX=true
    "= true if medium contains the equation sum(X) = 1.0; set
reducedX=true if only one substance (see docu for details)";
  constant Boolean fixedX=false
    "= true if medium contains the equation X = reference_X";
  constant AbsolutePressure reference_p=101325
    "Reference pressure of Medium: default 1 atmosphere";
  constant Temperature reference_T=298.15
    "Reference temperature of Medium: default 25 deg Celsius";
  constant MassFraction reference_X[nX]= if nX == 0 then fill(0,nX)
else fill(1/nX, nX)
    "Default mass fractions of medium";
  constant AbsolutePressure p_default=101325
    "Default value for pressure of medium (for initialization)";
  constant Temperature T_default =
Modelica.SIunits.Conversions.from_degC(20)
    "Default value for temperature of medium (for initialization)";
  constant SpecificEnthalpy h_default =
specificEnthalpy_pTX(p_default, T_default, X_default)
    "Default value for specific enthalpy of medium (for
initialization)";
  constant MassFraction X_default[nX]=reference_X
    "Default value for mass fractions of medium (for initialization)";
```

```
  final constant Integer nS=size(substanceNames, 1) "Number of
substances";
  final constant Integer nX=if nS == 1 then 0 else nS
    "Number of mass fractions (= 0, if only one substance)";
  final constant Integer nXi=if fixedX then 0 else if reducedX then nS
- 1 else nX
    "Number of structurally independent mass fractions (see docu for
details)";

  final constant Integer nC=size(extraPropertiesNames, 1)
    "Number of extra (outside of standard mass-balance) transported
properties";

  replaceable record FluidConstants
    "critical, triple, molecular and other standard data of fluid"
    extends Modelica.Icons.Record;
    String iupacName "complete IUPAC name (or common name, if non-
existent)";
    String casRegistryNumber
      "chemical abstracts sequencing number (if it exists)";
    String chemicalFormula
      "Chemical formula, (brutto, nomenclature according to Hill";
    String structureFormula "Chemical structure formula";
    MolarMass molarMass "molar mass";
  end FluidConstants;

  replaceable record ThermodynamicState
    "Minimal variable set that is available as input argument to every
medium function"
    extends Modelica.Icons.Record;
  end ThermodynamicState;

  replaceable record BasePropertiesRecord
    "Variables contained in every instance of BaseProperties"
    extends Modelica.Icons.Record;
    AbsolutePressure p "Absolute pressure of medium";
    Density d "Density of medium";
    Temperature T "Temperature of medium";
    MassFraction[nX] X(start=reference_X)
      "Mass fractions (= (component mass)/total mass  m_i/m)";
    MassFraction[nXi] Xi(start=reference_X[1:nXi])
      "Structurally independent mass fractions";
    SpecificEnthalpy h "Specific enthalpy of medium";
    SpecificInternalEnergy u "Specific internal energy of medium";
    SpecificHeatCapacity R "Gas constant (of mixture if applicable)";
    MolarMass MM "Molar mass (of mixture or single fluid)";
  end BasePropertiesRecord;

  replaceable partial model BaseProperties
    "Base properties (p, d, T, h, u, R, MM and, if applicable, X) of a
medium"
    extends BasePropertiesRecord;
    ThermodynamicState state
      "thermodynamic state variables for optional functions";
    parameter Boolean preferredMediumStates=false
      "= true if StateSelect.prefer shall be used for the independent
property variables of the medium";
    SI.Conversions.NonSIunits.Temperature_degC T_degC=
        Modelica.SIunits.Conversions.to_degC(T)
      "Temperature of medium in [degC]";
    SI.Conversions.NonSIunits.Pressure_bar p_bar=
```

```
        Modelica.SIunits.Conversions.to_bar(p)
      "Absolute pressure of medium in [bar]";
  equation
    Xi = X[1:nXi];
    if nX > 1 then
      if fixedX then
        X = reference_X;
      elseif reducedX then
        X[nX] = 1 - sum(Xi);
      end if;
      for i in 1:nX loop
        assert(X[i] >= -1.e-5 and X[i] <= 1 + 1.e-5, "Mass fraction
X[" +
          String(i) + "] = " + String(X[i]) + "of substance " +
substanceNames[
          i] + "\nof medium " + mediumName + " is not in the range
0..1");
      end for;
    end if;

    assert(p >= 0.0, "Pressure (= " + String(p) + " Pa) of medium \""
+
      mediumName + "\" is negative\n(Temperature = " + String(T) + "
K)");

  end BaseProperties;

  replaceable partial function setState_pTX
    "Return thermodynamic state as function of p, T and composition X
or Xi"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    input MassFraction X[:]=reference_X "Mass fractions";
    output ThermodynamicState state;
  end setState_pTX;

  replaceable partial function setState_phX
    "Return thermodynamic state as function of p, h and composition X
or Xi"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input MassFraction X[:]=reference_X "Mass fractions";
    output ThermodynamicState state;
  end setState_phX;

  replaceable partial function setState_psX
    "Return thermodynamic state as function of p, s and composition X
or Xi"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input MassFraction X[:]=reference_X "Mass fractions";
    output ThermodynamicState state;
  end setState_psX;

  replaceable partial function setState_dTX
    "Return thermodynamic state as function of d, T and composition X
or Xi"
    extends Modelica.Icons.Function;
```

```
    input Density d "density";
    input Temperature T "Temperature";
    input MassFraction X[:]=reference_X "Mass fractions";
    output ThermodynamicState state;
  end setState_dTX;

  replaceable partial function dynamicViscosity
    "Return dynamic viscosity"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output DynamicViscosity eta "Dynamic viscosity";
  end dynamicViscosity;

  replaceable partial function thermalConductivity
    "Return thermal conductivity"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output ThermalConductivity lambda "Thermal conductivity";
  end thermalConductivity;

  replaceable function prandtlNumber "Return the Prandtl number"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output PrandtlNumber Pr "Prandtl number";
  algorithm
    Pr :=
dynamicViscosity(state)*specificHeatCapacityCp(state)/thermalConductiv
ity(
      state);
  end prandtlNumber;

  replaceable partial function pressure "Return pressure"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output AbsolutePressure p "Pressure";
  end pressure;

  replaceable partial function temperature "Return temperature"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output Temperature T "Temperature";
  end temperature;

  replaceable partial function density "Return density"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output Density d "Density";
  end density;

  replaceable partial function specificEnthalpy
    "Return specific enthalpy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificEnthalpy h "Specific enthalpy";
  end specificEnthalpy;

  replaceable partial function specificInternalEnergy
    "Return specific internal energy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificEnergy u "Specific internal energy";
```

```
  end specificInternalEnergy;

  replaceable partial function specificEntropy
    "Return specific entropy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificEntropy s "Specific entropy";
  end specificEntropy;

  replaceable partial function specificGibbsEnergy
    "Return specific Gibbs energy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificEnergy g "Specific Gibbs energy";
  end specificGibbsEnergy;

  replaceable partial function specificHelmholtzEnergy
    "Return specific Helmholtz energy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificEnergy f "Specific Helmholtz energy";
  end specificHelmholtzEnergy;

  replaceable partial function specificHeatCapacityCp
    "Return specific heat capacity at constant pressure"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificHeatCapacity cp
      "Specific heat capacity at constant pressure";
  end specificHeatCapacityCp;

  function heatCapacity_cp = specificHeatCapacityCp "alias for
deprecated name";

  replaceable partial function specificHeatCapacityCv
    "Return specific heat capacity at constant volume"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SpecificHeatCapacity cv "Specific heat capacity at constant
volume";
  end specificHeatCapacityCv;

  function heatCapacity_cv = specificHeatCapacityCv "alias for
deprecated name";

  replaceable partial function isentropicExponent
    "Return isentropic exponent"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output IsentropicExponent gamma "Isentropic exponent";
  end isentropicExponent;

  replaceable partial function isentropicEnthalpy
    "Return isentropic enthalpy"
    extends Modelica.Icons.Function;
    input AbsolutePressure p_downstream "downstream pressure";
    input ThermodynamicState refState "reference state for entropy";
    output SpecificEnthalpy h_is "Isentropic enthalpy";
  end isentropicEnthalpy;

  replaceable partial function velocityOfSound
```

```
    "Return velocity of sound"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output VelocityOfSound a "Velocity of sound";
  end velocityOfSound;

  replaceable partial function isobaricExpansionCoefficient
    "Return overall the isobaric expansion coefficient beta"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output IsobaricExpansionCoefficient beta "Isobaric expansion
coefficient";
  end isobaricExpansionCoefficient;

  function beta = isobaricExpansionCoefficient
    "alias for isobaricExpansionCoefficient for user convenience";

  replaceable partial function isothermalCompressibility
    "Return overall the isothermal compressibility factor"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output SI.IsothermalCompressibility kappa "Isothermal
compressibility";
  end isothermalCompressibility;

  function kappa = isothermalCompressibility
    "alias of isothermalCompressibility for user convenience";

  // explicit derivative functions for finite element models
  replaceable partial function density_derp_h
    "Return density derivative wrt pressure at const specific
enthalpy"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output DerDensityByPressure ddph "Density derivative wrt
pressure";
  end density_derp_h;

  replaceable partial function density_derh_p
    "Return density derivative wrt specific enthalpy at constant
pressure"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output DerDensityByEnthalpy ddhp "Density derivative wrt specific
enthalpy";
  end density_derh_p;

  replaceable partial function density_derp_T
    "Return density derivative wrt pressure at const temperature"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output DerDensityByPressure ddpT "Density derivative wrt
pressure";
  end density_derp_T;

  replaceable partial function density_derT_p
    "Return density derivative wrt temperature at constant pressure"
    extends Modelica.Icons.Function;
    input ThermodynamicState state;
    output DerDensityByTemperature ddTp "Density derivative wrt
temperature";
```

```
end density_derT_p;

replaceable partial function density_derX
  "Return density derivative wrt mass fraction"
  extends Modelica.Icons.Function;
  input ThermodynamicState state;
  output Density[nX] dddX "Derivative of density wrt mass fraction";
end density_derX;

replaceable partial function molarMass
  "Return the molar mass of the medium"
  extends Modelica.Icons.Function;
  input ThermodynamicState state;
  output MolarMass MM "Mixture molar mass";
end molarMass;

replaceable function specificEnthalpy_pTX
  "Return specific enthalpy from p, T, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input Temperature T "Temperature";
  input MassFraction X[:]=reference_X "Mass fractions";
  output SpecificEnthalpy h "Specific enthalpy";
algorithm
  h := specificEnthalpy(setState_pTX(p,T,X));
end specificEnthalpy_pTX;

replaceable function density_pTX
  "Return density from p, T, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input Temperature T "Temperature";
  input MassFraction X[:] "Mass fractions";
  output Density d "Density";
algorithm
  d := density(setState_pTX(p,T,X));
end density_pTX;

replaceable function temperature_phX
  "Return temperature from p, h, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEnthalpy h "Specific enthalpy";
  input MassFraction X[:]=reference_X "Mass fractions";
  output Temperature T "Temperature";
algorithm
  T := temperature(setState_phX(p,h,X));
end temperature_phX;

replaceable function density_phX
  "Return density from p, h, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEnthalpy h "Specific enthalpy";
  input MassFraction X[:]=reference_X "Mass fractions";
  output Density d "Density";
algorithm
  d := density(setState_phX(p,h,X));
end density_phX;

replaceable function temperature_psX
```

```
  "Return temperature from p,s, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEntropy s "Specific entropy";
  input MassFraction X[:]=reference_X "Mass fractions";
  output Temperature T "Temperature";
algorithm
  T := temperature(setState_psX(p,s,X));
end temperature_psX;

replaceable function density_psX
  "Return density from p, s, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEntropy s "Specific entropy";
  input MassFraction X[:]=reference_X "Mass fractions";
  output Density d "Density";
algorithm
  d := density(setState_psX(p,s,X));
end density_psX;

replaceable function specificEnthalpy_psX
  "Return specific enthalpy from p, s, and X or Xi"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEntropy s "Specific entropy";
  input MassFraction X[:]=reference_X "Mass fractions";
  output SpecificEnthalpy h "Specific enthalpy";
algorithm
  h := specificEnthalpy(setState_psX(p,s,X));
end specificEnthalpy_psX;

type AbsolutePressure = SI.AbsolutePressure (
    min=0,
    max=1.e8,
    nominal=1.e5,
    start=1.e5) "Absolute pressure with medium specific attributes";

type Density = SI.Density (
    min=0,
    max=1.e5,
    nominal=1,
    start=1);
type DynamicViscosity = SI.DynamicViscosity (
    min=0,
    max=1.e8,
    nominal=1.e-3,
    start=1.e-3);
type EnthalpyFlowRate = SI.EnthalpyFlowRate (
    nominal=1000.0,
    min=-1.0e8,
    max=1.e8);
type MassFlowRate = SI.MassFlowRate (
    quantity="MassFlowRate." + mediumName,
    min=-1.0e5,
    max=1.e5);
type MassFraction = Real (
    quantity="MassFraction",
    final unit="kg/kg",
    min=0,
    max=1,
```

```
    nominal=0.1);
type MoleFraction = Real (
    quantity="MoleFraction",
    final unit="mol/mol",
    min=0,
    max=1,
    nominal=0.1);
type MolarMass = SI.MolarMass (
    min=0.001,
    max=0.25,
    nominal=0.032);
type MolarVolume = SI.MolarVolume (
    min=1e-6,
    max=1.0e6,
    nominal=1.0);
type IsentropicExponent = SI.RatioOfSpecificHeatCapacities (
    min=1,
    max=1.7,
    nominal=1.2,
    start=1.2);
type SpecificEnergy = SI.SpecificEnergy (
    min=-1.0e8,
    max=1.e8,
    nominal=1.e6);
type SpecificInternalEnergy = SpecificEnergy;
type SpecificEnthalpy = SI.SpecificEnthalpy (
    min=-1.0e8,
    max=1.e8,
    nominal=1.e6);
type SpecificEntropy = SI.SpecificEntropy (
    min=-1.e6,
    max=1.e6,
    nominal=1.e3);
type SpecificHeatCapacity = SI.SpecificHeatCapacity (
    min=0,
    max=1.e6,
    nominal=1.e3,
    start=1.e3);
type SurfaceTension = SI.SurfaceTension;
type Temperature = SI.Temperature (
    min=1,
    max=1.e4,
    nominal=300,
    start=300);
type ThermalConductivity = SI.ThermalConductivity (
    min=0,
    max=500,
    nominal=1,
    start=1);
type PrandtlNumber = SI.PrandtlNumber (
    min=1e-3,
    max=1e5,
    nominal=1.0);
type VelocityOfSound = SI.Velocity (
    min=0,
    max=1.e5,
    nominal=1000,
    start=1000);
type ExtraProperty = Real (min=0.0, start=1.0)
  "unspecified, mass-specific property transported by flow";
type CumulativeExtraProperty = Real (min=0.0, start=1.0)
```

```
    "conserved integral of unspecified, mass specific property";
  type ExtraPropertyFlowRate = Real
    "flow rate of unspecified, mass-specific property";
  type IsobaricExpansionCoefficient = Real (
      min=1e-8,
      max=1.0e8,
      unit="1/K") "isobaric expansion coefficient";
  type DipoleMoment = Real (
      min=0.0,
      max=2.0,
      unit="debye",
      quantity="ElectricDipoleMoment");

  type DerDensityByPressure = SI.DerDensityByPressure;
  type DerDensityByEnthalpy = SI.DerDensityByEnthalpy;
  type DerEnthalpyByPressure = SI.DerEnthalpyByPressure;
  type DerDensityByTemperature = SI.DerDensityByTemperature;

  package Choices "Types, constants to define menu choices"
    package Init
      "Type, constants and menu choices to define initialization, as
temporary solution until enumerations are available"


      extends Modelica.Icons.Library;
      constant Integer NoInit=1;
      constant Integer InitialStates=2;
      constant Integer SteadyState=3;
      constant Integer SteadyMass=4;
      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"

        extends Integer;
      end Temp;
    end Init;

    package ReferenceEnthalpy
      "Type, constants and menu choices to define reference enthalpy,
as temporary solution until enumerations are available"


      extends Modelica.Icons.Library;
      constant Integer ZeroAt0K=1;
      constant Integer ZeroAt25C=2;
      constant Integer UserDefined=3;
      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"

        extends Integer;

      end Temp;
    end ReferenceEnthalpy;

    package ReferenceEntropy
      "Type, constants and menu choices to define reference entropy,
as temporary solution until enumerations are available"


      extends Modelica.Icons.Library;
```

```
      constant Integer ZeroAt0K=1;
      constant Integer ZeroAt0C=2;
      constant Integer UserDefined=3;
      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"

        extends Integer;

      end Temp;
    end ReferenceEntropy;

    package pd
      "Type, constants and menu choices to define whether p or d are
known, as temporary solution until enumerations are available"


      extends Modelica.Icons.Library;
      constant Integer default=1;
      constant Integer p_known=2;
      constant Integer d_known=3;

      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"

        extends Integer;
      end Temp;
    end pd;

    package Th
      "Type, constants and menu choices to define whether T or h are
known, as temporary solution until enumerations are available"


      extends Modelica.Icons.Library;
      constant Integer default=1;
      constant Integer T_known=2;
      constant Integer h_known=3;

      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"

        extends Integer;
      end Temp;
    end Th;

    package Explicit
      "Type, constants and menu choices to define the explicitly given
state variable inputs"
      extends Modelica.Icons.Enumeration;

      constant Integer dT_explicit=0 "explicit in density and
temperature";
      constant Integer ph_explicit=1
        "explicit in pressure and specific enthalpy";
      constant Integer ps_explicit=2
        "explicit in pressure and specific entropy";
      constant Integer pT_explicit=3 "explicit in pressure and
temperature";
```

```
      type Temp
        "Temporary type with choices for menus (until enumerations are
available)"
        extends Integer(min=0,max=3);
      end Temp;
    end Explicit;
  end Choices;
end PartialMedium;

partial record Modelica.Icons.Record "Icon for a record"
end Record;

partial function Modelica.Icons.Function "Icon for a function"
algorithm

end Function;

partial class Modelica.Icons.Enumeration
  "Icon for an enumeration (emulated by a package)"

end Enumeration;

partial package Modelica.Media.Water.WaterIF97_base
  "Water: Steam properties as defined by IAPWS/IF97 standard"

  extends Interfaces.PartialTwoPhaseMedium(
     mediumName="WaterIF97",
     substanceNames={"water"},
     final reducedX=true,
     singleState=false,
     SpecificEnthalpy(start=1.0e5, nominal=5.0e5),
     Density(start=150, nominal=500),
     AbsolutePressure(start=50e5, nominal=10e5),
     Temperature(start=500, nominal=500),
     smoothModel=false,
     onePhase=false,
     fluidConstants = waterConstants);

  redeclare record extends ThermodynamicState "thermodynamic state"
    SpecificEnthalpy h "specific enthalpy";
    Density d "density";
    Temperature T "temperature";
    AbsolutePressure p "pressure";
  end ThermodynamicState;

  constant Boolean ph_explicit
    "true if explicit in pressure and specific enthalpy";
  constant Boolean dT_explicit "true if explicit in density and
temperature";
  constant Boolean pT_explicit "true if explicit in pressure and
temperature";

  redeclare replaceable model extends BaseProperties(
    h(stateSelect=if ph_explicit and preferredMediumStates then
StateSelect.prefer else StateSelect.default),
    d(stateSelect=if dT_explicit and preferredMediumStates then
StateSelect.prefer else StateSelect.default),
    T(stateSelect=if (pT_explicit or dT_explicit) and
preferredMediumStates then StateSelect.prefer else
StateSelect.default),
```

```
    p(stateSelect=if (pT_explicit or ph_explicit) and
preferredMediumStates then StateSelect.prefer else
StateSelect.default))
    "Base properties of water"
    Integer phase(min=0, max=2, start=1,fixed=false)
      "2 for two-phase, 1 for one-phase, 0 if not known";
    SaturationProperties sat(Tsat(start=300.0), psat(start=1.0e5))
      "saturation temperature and pressure";
  equation
    MM = fluidConstants[1].molarMass;
    if smoothModel then
      if onePhase then
        phase = 1;
        if ph_explicit then
          assert(((h < bubbleEnthalpy(sat) or h > dewEnthalpy(sat)) or
p >
    fluidConstants[1].criticalPressure),
   "With onePhase=true this model may only be called with one-phase
states h < hl or h > hv!"
   + "(p = " + String(p) + ", h = " + String(h) + ")");
        else
   if dT_explicit then
     assert(not ((d < bubbleDensity(sat) and d > dewDensity(sat)) and
T <
   fluidConstants[1].criticalTemperature),
     "With onePhase=true this model may only be called with one-phase
states d > dl or d < dv!"
     + "(d = " + String(d) + ", T = " + String(T) + ")");
   else
     assert(true,"no events for pT-model");
   end if;
        end if;
      else
        phase = 0;
      end if;
    else
      if ph_explicit then
        phase = if ((h < bubbleEnthalpy(sat) or h > dewEnthalpy(sat))
or p >
          fluidConstants[1].criticalPressure) then 1 else 2;
      elseif dT_explicit then
        phase = if not ((d < bubbleDensity(sat) and d >
dewDensity(sat)) and T
           < fluidConstants[1].criticalTemperature) then 1 else 2;
      else
        phase = 1;
        //this is for the one-phase only case pT
      end if;
    end if;
    if dT_explicit then
      p = pressure_dT(d, T, phase);
      h = specificEnthalpy_dT(d, T, phase);
      sat.Tsat = T;
      sat.psat = saturationPressure(T);
    elseif ph_explicit then
      d = density_ph(p, h, phase);
      T = temperature_ph(p, h, phase);
      sat.Tsat = saturationTemperature(p);
      sat.psat = p;
    else
      h = specificEnthalpy_pT(p, T);
```

```
      d = density_pT(p, T);
      sat.psat = p;
      sat.Tsat = saturationTemperature(p);
    end if;
    u = h - p/d;
    R = Modelica.Constants.R/fluidConstants[1].molarMass;
    h = state.h;
    p = state.p;
    T = state.T;
    d = state.d;
    phase = state.phase;
  end BaseProperties;

  redeclare function density_ph
    "Computes density as a function of pressure and specific enthalpy"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "Density";
  algorithm
    d := IF97_Utilities.rho_ph(p, h, phase);
  end density_ph;

  redeclare function temperature_ph
    "Computes temperature as a function of pressure and specific
enthalpy"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Temperature T "Temperature";
  algorithm
    T := IF97_Utilities.T_ph(p, h, phase);
  end temperature_ph;

  redeclare function temperature_ps
    "Compute temperature from pressure and specific enthalpy"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Temperature T "Temperature";
  algorithm
    T := IF97_Utilities.T_ps(p, s, phase);
  end temperature_ps;

  redeclare function density_ps
    "Computes density as a function of pressure and specific enthalpy"
      extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "density";
  algorithm
    d := IF97_Utilities.rho_ps(p, s, phase);
  end density_ps;
```

```
  redeclare function pressure_dT
    "Computes pressure as a function of density and temperature"
    extends Modelica.Icons.Function;
    input Density d "Density";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output AbsolutePressure p "Pressure";
  algorithm
    p := IF97_Utilities.p_dT(d, T, phase);
  end pressure_dT;

  redeclare function specificEnthalpy_dT
    "Computes specific enthalpy as a function of density and
temperature"
    extends Modelica.Icons.Function;
    input Density d "Density";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := IF97_Utilities.h_dT(d, T, phase);
  end specificEnthalpy_dT;

  redeclare function specificEnthalpy_pT
    "Computes specific enthalpy as a function of pressure and
temperature"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := IF97_Utilities.h_pT(p, T);
  end specificEnthalpy_pT;

  redeclare function specificEnthalpy_ps
    "Computes specific enthalpy as a function of pressure and
temperature"
      extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := IF97_Utilities.h_ps(p, s, phase);
  end specificEnthalpy_ps;

  redeclare function density_pT
    "Computes density as a function of pressure and temperature"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "Density";
  algorithm
```

```
    d := IF97_Utilities.rho_pT(p, T);
  end density_pT;

  redeclare function extends setDewState
    "set the thermodynamic state on the dew line"
  algorithm
    state := ThermodynamicState(
       phase=  phase,
       p=  sat.psat,
       T=  sat.Tsat,
       h=  dewEnthalpy(sat),
       d=  dewDensity(sat));
  end setDewState;

  redeclare function extends setBubbleState
    "set the thermodynamic state on the bubble line"
  algorithm
    state := ThermodynamicState(
       phase=  phase,
       p=  sat.psat,
       T=  sat.Tsat,
       h=  bubbleEnthalpy(sat),
       d=  bubbleDensity(sat));
  end setBubbleState;

  redeclare function extends dynamicViscosity
    "Dynamic viscosity of water"
  algorithm
    eta := IF97_Utilities.dynamicViscosity(state.d, state.T, state.p,
state.
      phase);
  end dynamicViscosity;

  redeclare function extends thermalConductivity
    "Thermal conductivity of water"
  algorithm
    lambda := IF97_Utilities.thermalConductivity(state.d, state.T,
state.p,
      state.phase);
  end thermalConductivity;

  redeclare function extends surfaceTension
    "Surface tension in two phase region of water"
  algorithm
    sigma := IF97_Utilities.surfaceTension(sat.Tsat);
  end surfaceTension;

  redeclare function extends pressure "return pressure of ideal gas"
  algorithm
    p := state.p;
  end pressure;

  redeclare function extends temperature
    "return temperature of ideal gas"
  algorithm
    T := state.T;
  end temperature;

  redeclare function extends density "return density of ideal gas"
  algorithm
    d := state.d;
```

```
end density;

redeclare function extends specificEnthalpy
  "Return specific enthalpy"
  extends Modelica.Icons.Function;
algorithm
  h := state.h;
end specificEnthalpy;

redeclare function extends specificInternalEnergy
  "Return specific internal energy"
  extends Modelica.Icons.Function;
algorithm
  u := state.h  - state.p/state.d;
end specificInternalEnergy;

redeclare function extends specificGibbsEnergy
  "Return specific Gibbs energy"
  extends Modelica.Icons.Function;
algorithm
  g := state.h - state.T*specificEntropy(state);
end specificGibbsEnergy;

redeclare function extends specificHelmholtzEnergy
  "Return specific Helmholtz energy"
  extends Modelica.Icons.Function;
algorithm
  f := state.h - state.p/state.d - state.T*specificEntropy(state);
end specificHelmholtzEnergy;

redeclare function extends specificEntropy
  "specific entropy of water"
algorithm
  if dT_explicit then
    s := IF97_Utilities.s_dT(state.d, state.T, state.phase);
  elseif pT_explicit then
    s := IF97_Utilities.s_pT(state.p, state.T);
  else
    s := IF97_Utilities.s_ph(state.p, state.h, state.phase);
  end if;
end specificEntropy;

redeclare function extends specificHeatCapacityCp
  "specific heat capacity at constant pressure of water"

algorithm
  if dT_explicit then
    cp := IF97_Utilities.cp_dT(state.d, state.T, state.phase);
  elseif pT_explicit then
    cp := IF97_Utilities.cp_pT(state.p, state.T);
  else
    cp := IF97_Utilities.cp_ph(state.p, state.h, state.phase);
  end if;
end specificHeatCapacityCp;

redeclare function extends specificHeatCapacityCv
  "specific heat capacity at constant volume of water"
algorithm
  if dT_explicit then
    cv := IF97_Utilities.cv_dT(state.d, state.T, state.phase);
  elseif pT_explicit then
```

```
      cv := IF97_Utilities.cv_pT(state.p, state.T);
    else
      cv := IF97_Utilities.cv_ph(state.p, state.h, state.phase);
    end if;
  end specificHeatCapacityCv;

  redeclare function extends isentropicExponent
    "Return isentropic exponent"
  algorithm
    if dT_explicit then
      gamma := IF97_Utilities.isentropicExponent_dT(state.d, state.T,
state.
        phase);
    elseif pT_explicit then
      gamma := IF97_Utilities.isentropicExponent_pT(state.p, state.T);
    else
      gamma := IF97_Utilities.isentropicExponent_ph(state.p, state.h,
state.
        phase);
    end if;
  end isentropicExponent;

  redeclare function extends isothermalCompressibility
    "Isothermal compressibility of water"
  algorithm
    if dT_explicit then
      kappa := IF97_Utilities.kappa_dT(state.d, state.T, state.phase);
    elseif pT_explicit then
      kappa := IF97_Utilities.kappa_pT(state.p, state.T);
    else
      kappa := IF97_Utilities.kappa_ph(state.p, state.h, state.phase);
    end if;
  end isothermalCompressibility;

  redeclare function extends isobaricExpansionCoefficient
    "isobaric expansion coefficient of water"
  algorithm
    if dT_explicit then
      beta := IF97_Utilities.beta_dT(state.d, state.T, state.phase);
    elseif pT_explicit then
      beta := IF97_Utilities.beta_pT(state.p, state.T);
    else
      beta := IF97_Utilities.beta_ph(state.p, state.h, state.phase);
    end if;
  end isobaricExpansionCoefficient;

  redeclare function extends velocityOfSound
  algorithm
    if dT_explicit then
      a := IF97_Utilities.velocityOfSound_dT(state.d, state.T,
state.phase);
    elseif pT_explicit then
      a := IF97_Utilities.velocityOfSound_pT(state.p, state.T);
    else
      a := IF97_Utilities.velocityOfSound_ph(state.p, state.h,
state.phase);
    end if;
  end velocityOfSound;

  redeclare function extends isentropicEnthalpy "compute h(p,s)"
  algorithm
```

```
    h_is := IF97_Utilities.isentropicEnthalpy(p_downstream,
specificEntropy(
      refState), 0);
  end isentropicEnthalpy;

  redeclare function extends density_derh_p
    "density derivative by specific enthalpy"
  algorithm
    ddhp := IF97_Utilities.ddhp(state.p, state.h, state.phase);
  end density_derh_p;

  redeclare function extends density_derp_h
    "density derivative by pressure"
  algorithm
    ddph := IF97_Utilities.ddph(state.p, state.h, state.phase);
  end density_derp_h;

  redeclare function extends bubbleEnthalpy
    "boiling curve specific enthalpy of water"
  algorithm
    hl := IF97_Utilities.BaseIF97.Regions.hl_p(sat.psat);
  end bubbleEnthalpy;

  redeclare function extends dewEnthalpy
    "dew curve specific enthalpy of water"
  algorithm
    hv := IF97_Utilities.BaseIF97.Regions.hv_p(sat.psat);
  end dewEnthalpy;

  redeclare function extends bubbleEntropy
    "boiling curve specific entropy of water"
  algorithm
    sl := IF97_Utilities.BaseIF97.Regions.sl_p(sat.psat);
  end bubbleEntropy;

  redeclare function extends dewEntropy
    "dew curve specific entropy of water"
  algorithm
    sv := IF97_Utilities.BaseIF97.Regions.sv_p(sat.psat);
  end dewEntropy;

  redeclare function extends bubbleDensity
    "boiling curve specific density of water"
  algorithm
    if ph_explicit or pT_explicit then
      dl := IF97_Utilities.BaseIF97.Regions.rhol_p(sat.psat);
    else
      dl := IF97_Utilities.BaseIF97.Regions.rhol_T(sat.Tsat);
    end if;
  end bubbleDensity;

  redeclare function extends dewDensity
    "dew curve specific density of water"
  algorithm
    if ph_explicit or pT_explicit then
      dv := IF97_Utilities.BaseIF97.Regions.rhov_p(sat.psat);
    else
      dv := IF97_Utilities.BaseIF97.Regions.rhov_T(sat.Tsat);
    end if;
  end dewDensity;
```

```
redeclare function extends saturationTemperature
  "saturation temperature of water"
algorithm
  T := IF97_Utilities.BaseIF97.Basic.tsat(p);
end saturationTemperature;

redeclare function extends saturationTemperature_derp
  "derivative of saturation temperature w.r.t. pressure"
algorithm
  dTp := IF97_Utilities.BaseIF97.Basic.dtsatofp(p);
end saturationTemperature_derp;

redeclare function extends saturationPressure
  "saturation pressure of water"
algorithm
  p := IF97_Utilities.BaseIF97.Basic.psat(T);
end saturationPressure;

redeclare function extends dBubbleDensity_dPressure
  "bubble point density derivative"
algorithm
  ddldp := IF97_Utilities.BaseIF97.Regions.drhol_dp(sat.psat);
end dBubbleDensity_dPressure;

redeclare function extends dDewDensity_dPressure
  "dew point density derivative"
algorithm
  ddvdp := IF97_Utilities.BaseIF97.Regions.drhov_dp(sat.psat);
end dDewDensity_dPressure;

redeclare function extends dBubbleEnthalpy_dPressure
  "bubble point specific enthalpy derivative"
algorithm
  dhldp := IF97_Utilities.BaseIF97.Regions.dhl_dp(sat.psat);
end dBubbleEnthalpy_dPressure;

redeclare function extends dDewEnthalpy_dPressure
  "dew point specific enthalpy derivative"
algorithm
  dhvdp := IF97_Utilities.BaseIF97.Regions.dhv_dp(sat.psat);
end dDewEnthalpy_dPressure;

redeclare function extends setState_dTX
algorithm
  state := ThermodynamicState(
    d=d,
    T=T,
    phase=IF97_Utilities.phase_dT(d,T),
    h=specificEnthalpy_dT(d,T),
    p=pressure_dT(d,T));
end setState_dTX;

redeclare function extends setState_phX
algorithm
  state := ThermodynamicState(
    d=density_ph(p,h),
    T=temperature_ph(p,h),
    phase=IF97_Utilities.phase_ph(p,h),
    h=h,
    p=p);
end setState_phX;
```

```
  redeclare function extends setState_psX
  algorithm
    state := ThermodynamicState(
      d=density_ps(p,s),
      T=temperature_ps(p,s),
      phase=IF97_Utilities.phase_ps(p,s),
      h=specificEnthalpy_ps(p,s),
      p=p);
  end setState_psX;

  redeclare function extends setState_pTX
  algorithm
    state := ThermodynamicState(
      d=density_pT(p,T),
      T=T,
      phase=1,
      h=specificEnthalpy_pT(p,T),
      p=p);
  end setState_pTX;
end WaterIF97_base;

partial package Modelica.Media.Interfaces.PartialTwoPhaseMedium
  extends PartialPureSubstance;

  constant Boolean smoothModel
    "true if the (derived) model should not generate state events";
  constant Boolean onePhase
    "true if the (derived) model should never be called with two-phase
inputs";

  record FluidLimits "validity limits for fluid model"
    extends Modelica.Icons.Record;
    Temperature TMIN "minimum temperature";
    Temperature TMAX "maximum temperature";
    Density DMIN "minimum density";
    Density DMAX "maximum density";
    AbsolutePressure PMIN "minimum pressure";
    AbsolutePressure PMAX "maximum pressure";
    SpecificEnthalpy HMIN "minimum enthalpy";
    SpecificEnthalpy HMAX "maximum enthalpy";
    SpecificEntropy SMIN "minimum entropy";
    SpecificEntropy SMAX "maximum entropy";
  end FluidLimits;

  redeclare replaceable record extends FluidConstants
    "extended fluid constants"
    Temperature criticalTemperature "critical temperature";
    AbsolutePressure criticalPressure "critical pressure";
    MolarVolume criticalMolarVolume "critical molar Volume";
    Real acentricFactor "Pitzer acentric factor";
    Temperature triplePointTemperature "triple point temperature";
    AbsolutePressure triplePointPressure "triple point pressure";
    Temperature meltingPoint "melting point at 101325 Pa";
    Temperature normalBoilingPoint "normal boiling point (at 101325
Pa)";
    DipoleMoment dipoleMoment
      "dipole moment of molecule in Debye (1 debye = 3.33564e10-30
C.m)";
    Boolean hasIdealGasHeatCapacity=false
      "true if ideal gas heat capacity is available";
```

```
    Boolean hasCriticalData=false "true if critical data are known";
    Boolean hasDipoleMoment=false "true if a dipole moment known";
    Boolean hasFundamentalEquation=false "true if a fundamental
equation";
    Boolean hasLiquidHeatCapacity=false
      "true if liquid heat capacity is available";
    Boolean hasSolidHeatCapacity=false
      "true if solid heat capacity is available";
    Boolean hasAccurateViscosityData=false
      "true if accurate data for a viscosity function is available";
    Boolean hasAccurateConductivityData=false
      "true if accurate data for thermal conductivity is available";
    Boolean hasVapourPressureCurve=false
      "true if vapour pressure data, e.g. Antoine coefficents are
known";
    Boolean hasAcentricFactor=false "true if Pitzer accentric factor
is known";
    SpecificEnthalpy HCRIT0=0.0
      "Critical specific enthalpy of the fundamental equation";
    SpecificEntropy SCRIT0=0.0
      "Critical specific entropy of the fundamental equation";
    SpecificEnthalpy deltah=0.0
      "Difference between specific enthalpy model (h_m) and f.eq.
(h_f) (h_m - h_f)";
    SpecificEntropy deltas=0.0
      "Difference between specific enthalpy model (s_m) and f.eq.
(s_f) (s_m - s_f)";
  end FluidConstants;

constant FluidConstants[nS] fluidConstants "constant data for the
fluid";

redeclare replaceable record extends ThermodynamicState
    FixedPhase phase(min=0, max=2)
      "phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not
known, e.g. interactive use";
end ThermodynamicState;

  replaceable record SaturationProperties
    extends Modelica.Icons.Record;
    AbsolutePressure psat "saturation pressure";
    Temperature Tsat "saturation temperature";
  end SaturationProperties;

  type FixedPhase = Integer(min=0,max=2)
    "phase of the fluid: 1 for 1-phase, 2 for two-phase, 0 for not
known, e.g. interactive use";

  replaceable partial function setDewState
    "set the thermodynamic state on the dew line"
    extends Modelica.Icons.Function;
    input SaturationProperties sat "saturation point";
    input FixedPhase phase =  1 "phase: default is one phase";
    output ThermodynamicState state "complete thermodynamic state
info";
  end setDewState;

  replaceable partial function setBubbleState
    "set the thermodynamic state on the bubble line"
    extends Modelica.Icons.Function;
    input SaturationProperties sat "saturation point";
```

```
    input FixedPhase phase =  1 "phase: default is one phase";
    output ThermodynamicState state "complete thermodynamic state
info";
  end setBubbleState;

  redeclare replaceable partial function extends setState_dTX
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
  end setState_dTX;

  redeclare replaceable partial function extends setState_phX
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
  end setState_phX;

  redeclare replaceable partial function extends setState_psX
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
  end setState_psX;

  redeclare replaceable partial function extends setState_pTX
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
  end setState_pTX;

  replaceable function setSat_T
    "set saturation property record from temperature"
    extends Modelica.Icons.Function;
    input Temperature T "temperature";
    output SaturationProperties sat "saturation property record";
  algorithm
    sat.Tsat := T;
    sat.psat := saturationPressure(T);
  end setSat_T;

  replaceable function setSat_p
    "set saturation property record from pressure"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "pressure";
    output SaturationProperties sat "saturation property record";
  algorithm
    sat.psat := p;
    sat.Tsat := saturationTemperature(p);
  end setSat_p;

  replaceable partial function bubbleEnthalpy
    "Returns bubble point specific enthalpy"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output SI.SpecificEnthalpy hl "boiling curve specific enthalpy";
  end bubbleEnthalpy;

    replaceable partial function dewEnthalpy
    "Returns dew point specific enthalpy"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output SI.SpecificEnthalpy hv "dew curve specific enthalpy";
    end dewEnthalpy;

    replaceable partial function bubbleEntropy
    "Returns bubble point specific entropy"
```

```
    extends Modelica.Icons.Function;
    input SaturationProperties sat "saturation property record";
    output SI.SpecificEntropy sl "boiling curve specific entropy";
    end bubbleEntropy;

    replaceable partial function dewEntropy
    "Returns dew point specific entropy"
    extends Modelica.Icons.Function;
    input SaturationProperties sat "saturation property record";
    output SI.SpecificEntropy sv "dew curve specific entropy";
    end dewEntropy;

    replaceable partial function bubbleDensity
    "Returns bubble point density"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output Density dl "boiling curve density";
    end bubbleDensity;

    replaceable partial function dewDensity
    "Returns dew point density"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output Density dv "dew curve density";
    end dewDensity;

    replaceable partial function saturationPressure
    "Returns saturation pressure"
      extends Modelica.Icons.Function;
      input Temperature T "temperature";
      output AbsolutePressure p "saturation pressure";
    end saturationPressure;

    replaceable partial function saturationTemperature
    "Returns saturation temperature"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "pressure";
      output Temperature T "saturation temperature";
    end saturationTemperature;

    replaceable partial function saturationTemperature_derp
    "Returns derivatives of saturation temperature w.r.t pressure"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "pressure";
      output Real dTp "derivatives of saturation temperature w.r.t
pressure";
    end saturationTemperature_derp;

  replaceable partial function surfaceTension
    "Return surface tension sigma in the two phase region"
    extends Modelica.Icons.Function;
    input SaturationProperties sat "saturation property record";
    output SurfaceTension sigma "Surface tension sigma in the two
phase region";
  end surfaceTension;

    replaceable partial function dBubbleDensity_dPressure
    "Returns bubble point density derivative"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
```

```
      output DerDensityByPressure ddldp "boiling curve density
derivative";
    end dBubbleDensity_dPressure;

    replaceable partial function dDewDensity_dPressure
    "Returns dew point density derivative"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output DerDensityByPressure ddvdp "saturated steam density
derivative";
    end dDewDensity_dPressure;

    replaceable partial function dBubbleEnthalpy_dPressure
    "Returns bubble point specific enthalpy derivative"
      extends Modelica.Icons.Function;
      input SaturationProperties sat "saturation property record";
      output DerEnthalpyByPressure dhldp
      "boiling curve specific enthalpy derivative";
    end dBubbleEnthalpy_dPressure;

    replaceable partial function dDewEnthalpy_dPressure
    "Returns dew point specific enthalpy derivative"
      extends Modelica.Icons.Function;

      input SaturationProperties sat "saturation property record";
      output DerEnthalpyByPressure dhvdp
      "saturated steam specific enthalpy derivative";
    end dDewEnthalpy_dPressure;

    redeclare replaceable function specificEnthalpy_pTX
    "Compute specific enthalpy from pressure, temperature and mass
fraction"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input Temperature T "Temperature";
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output SpecificEnthalpy h "Specific enthalpy at p, T, X";
    algorithm
      h := specificEnthalpy(setState_pTX(p,T,X,phase));
    end specificEnthalpy_pTX;

    redeclare replaceable function temperature_phX
    "Return temperature from p, h, and X or Xi"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input SpecificEnthalpy h "Specific enthalpy";
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output Temperature T "Temperature";
    algorithm
      T := temperature(setState_phX(p,h,X,phase));
    end temperature_phX;

    redeclare replaceable function density_phX
    "Return density from p, h, and X or Xi"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input SpecificEnthalpy h "Specific enthalpy";
```

```
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output Density d "density";
    algorithm
      d := density(setState_phX(p,h,X,phase));
    end density_phX;

    redeclare replaceable function temperature_psX
    "Return temperature from p, s, and X or Xi"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input SpecificEntropy s "Specific entropy";
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output Temperature T "Temperature";
    algorithm
      T := temperature(setState_psX(p,s,X,phase));
    end temperature_psX;

    redeclare replaceable function density_psX
    "Return density from p, s, and X or Xi"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input SpecificEntropy s "Specific entropy";
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output Density d "Density";
    algorithm
      d := density(setState_psX(p,s,X,phase));
    end density_psX;

    redeclare replaceable function specificEnthalpy_psX
    "Return specific enthalpy from p, s, and X or Xi"
      extends Modelica.Icons.Function;
      input AbsolutePressure p "Pressure";
      input SpecificEntropy s "Specific entropy";
      input MassFraction X[nX] "Mass fractions";
      input FixedPhase phase=0
      "2 for two-phase, 1 for one-phase, 0 if not known";
      output SpecificEnthalpy h "specific enthalpy";
    algorithm
      h := specificEnthalpy(setState_psX(p,s,X,phase));
    end specificEnthalpy_psX;

  redeclare replaceable function setState_pT
    "Return thermodynamic state from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output ThermodynamicState state;
  algorithm
    state := setState_pTX(p,T,fill(0,0),phase);
  end setState_pT;

  redeclare replaceable function setState_ph
    "Return thermodynamic state from p and h"
```

```
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output ThermodynamicState state;
  algorithm
    state := setState_phX(p,h,fill(0, 0),phase);
  end setState_ph;

  redeclare replaceable function setState_ps
    "Return thermodynamic state from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output ThermodynamicState state;
  algorithm
    state := setState_psX(p,s,fill(0,0),phase);
  end setState_ps;

  redeclare replaceable function setState_dT
    "Return thermodynamic state from d and T"
    extends Modelica.Icons.Function;
    input Density d "density";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output ThermodynamicState state;
  algorithm
    state := setState_dTX(d,T,fill(0,0),phase);
  end setState_dT;

  redeclare replaceable function density_ph
    "Return density from p and h"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "Density";
  algorithm
    d := density_phX(p, h, fill(0,0), phase);
  end density_ph;

  redeclare replaceable function temperature_ph
    "Return temperature from p and h"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Temperature T "Temperature";
  algorithm
    T := temperature_phX(p, h, fill(0,0),phase);
  end temperature_ph;

  redeclare replaceable function pressure_dT
    "Return pressure from d and T"
    extends Modelica.Icons.Function;
```

```
    input Density d "Density";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output AbsolutePressure p "Pressure";
  algorithm
    p := pressure(setState_dTX(d, T, fill(0,0),phase));
  end pressure_dT;

  redeclare replaceable function specificEnthalpy_dT
    "Return specific enthalpy from d and T"
    extends Modelica.Icons.Function;
    input Density d "Density";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := specificEnthalpy(setState_dTX(d, T, fill(0,0),phase));
  end specificEnthalpy_dT;

  redeclare replaceable function specificEnthalpy_ps
    "Return specific enthalpy from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := specificEnthalpy_psX(p,s,fill(0,0));
  end specificEnthalpy_ps;

  redeclare replaceable function temperature_ps
    "Return temperature from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Temperature T "Temperature";
  algorithm
    T := temperature_psX(p,s,fill(0,0),phase);
  end temperature_ps;

  redeclare replaceable function density_ps
    "Return density from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "Density";
  algorithm
    d := density_psX(p, s, fill(0,0), phase);
  end density_ps;

  redeclare replaceable function specificEnthalpy_pT
    "Return specific enthalpy from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
```

```
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := specificEnthalpy_pTX(p, T, fill(0,0),phase);
  end specificEnthalpy_pT;

  redeclare replaceable function density_pT
    "Return density from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    input FixedPhase phase=0 "2 for two-phase, 1 for one-phase, 0 if
not known";
    output Density d "Density";
  algorithm
    d := density(setState_pTX(p, T, fill(0,0),phase));
  end density_pT;
end PartialTwoPhaseMedium;

partial package Modelica.Media.Interfaces.PartialPureSubstance
  "base class for pure substances of one chemical substance"
  extends PartialMedium;

  replaceable function setState_pT
    "Return thermodynamic state from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    output ThermodynamicState state;
  algorithm
    state := setState_pTX(p,T,fill(0,0));
  end setState_pT;

  replaceable function setState_ph
    "Return thermodynamic state from p and h"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEnthalpy h "Specific enthalpy";
    output ThermodynamicState state;
  algorithm
    state := setState_phX(p,h,fill(0, 0));
  end setState_ph;

  replaceable function setState_ps
    "Return thermodynamic state from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    output ThermodynamicState state;
  algorithm
    state := setState_psX(p,s,fill(0,0));
  end setState_ps;

  replaceable function setState_dT
    "Return thermodynamic state from d and T"
    extends Modelica.Icons.Function;
    input Density d "density";
    input Temperature T "Temperature";
    output ThermodynamicState state;
```

```
algorithm
  state := setState_dTX(d,T,fill(0,0));
end setState_dT;

replaceable function density_ph "Return density from p and h"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEnthalpy h "Specific enthalpy";
  output Density d "Density";
algorithm
  d := density_phX(p, h, fill(0,0));
end density_ph;

replaceable function temperature_ph
  "Return temperature from p and h"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEnthalpy h "Specific enthalpy";
  output Temperature T "Temperature";
algorithm
  T := temperature_phX(p, h, fill(0,0));
end temperature_ph;

replaceable function pressure_dT "Return pressure from d and T"
  extends Modelica.Icons.Function;
  input Density d "Density";
  input Temperature T "Temperature";
  output AbsolutePressure p "Pressure";
algorithm
  p := pressure(setState_dTX(d, T, fill(0,0)));
end pressure_dT;

replaceable function specificEnthalpy_dT
  "Return specific enthalpy from d and T"
  extends Modelica.Icons.Function;
  input Density d "Density";
  input Temperature T "Temperature";
  output SpecificEnthalpy h "specific enthalpy";
algorithm
  h := specificEnthalpy(setState_dTX(d, T, fill(0,0)));
end specificEnthalpy_dT;

replaceable function specificEnthalpy_ps
  "Return specific enthalpy from p and s"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEntropy s "Specific entropy";
  output SpecificEnthalpy h "specific enthalpy";
algorithm
  h := specificEnthalpy_psX(p,s,fill(0,0));
end specificEnthalpy_ps;

replaceable function temperature_ps
  "Return temperature from p and s"
  extends Modelica.Icons.Function;
  input AbsolutePressure p "Pressure";
  input SpecificEntropy s "Specific entropy";
  output Temperature T "Temperature";
algorithm
  T := temperature_psX(p,s,fill(0,0));
end temperature_ps;
```

```
  replaceable function density_ps "Return density from p and s"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input SpecificEntropy s "Specific entropy";
    output Density d "Density";
  algorithm
    d := density_psX(p, s, fill(0,0));
  end density_ps;

  replaceable function specificEnthalpy_pT
    "Return specific enthalpy from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    output SpecificEnthalpy h "specific enthalpy";
  algorithm
    h := specificEnthalpy_pTX(p, T, fill(0,0));
  end specificEnthalpy_pT;

  replaceable function density_pT "Return density from p and T"
    extends Modelica.Icons.Function;
    input AbsolutePressure p "Pressure";
    input Temperature T "Temperature";
    output Density d "Density";
  algorithm
    d := density(setState_pTX(p, T, fill(0,0)));
  end density_pT;
end PartialPureSubstance;

partial function Modelica.SIunits.Conversions.ConversionIcon
  "Base icon for conversion functions"
algorithm

end ConversionIcon;

connector Modelica.Blocks.Interfaces.RealInput =
                      input RealSignal "'input Real' as connector";

connector Modelica.Blocks.Interfaces.RealSignal
  "Real port (both input/output possible)"
  replaceable type SignalType = Real;

  extends SignalType;

end RealSignal;

connector Modelica.Blocks.Interfaces.RealOutput =
                       output RealSignal "'output Real' as connector";

block Modelica.Blocks.Math.Add "Output the sum of the two inputs"
  extends Interfaces.SI2SO;
  parameter Real k1=+1 "Gain of upper input";
  parameter Real k2=+1 "Gain of lower input";

equation
  y = k1*u1 + k2*u2;
end Add;

partial block Modelica.Blocks.Interfaces.SI2SO
  "2 Single Input / 1 Single Output continuous control block"
```

```
  extends BlockIcon;

  RealInput u1 "Connector of Real input signal 1";
  RealInput u2 "Connector of Real input signal 2";
  RealOutput y "Connector of Real output signal";

end SI2SO;

partial block Modelica.Blocks.Interfaces.BlockIcon
  "Basic graphical layout of input/output block"
equation

end BlockIcon;

block Modelica.Blocks.Math.Gain
  "Output the product of a gain value with the input signal"

  parameter Real k=1 "Gain value multiplied with input signal";
public
  Interfaces.RealInput u "Input signal connector";
  Interfaces.RealOutput y "Output signal connector";
equation
  y = k*u;
end Gain;

block Modelica.Blocks.Continuous.Integrator
  "Output the integral of the input signal"
  import Modelica.Blocks.Types.Init;
  parameter Real k=1 "Integrator gain";

  /* InitialState is the default, because it was the default in
Modelica 2.2
     and therefore this setting is backward compatible
  */
  parameter Init.Temp initType=Modelica.Blocks.Types.Init.InitialState
    "Type of initialization (InitialState and InitialOutput are
identical)";
  parameter Real y_start=0 "Initial or guess value of output (=
state)";
  extends Interfaces.SISO(y(start=y_start));

initial equation
  if initType == Init.SteadyState then
     der(y) = 0;
  elseif initType == Init.InitialState or
         initType == Init.InitialOutput then
    y = y_start;
  end if;
equation
  der(y) = k*u;
end Integrator;

partial block Modelica.Blocks.Interfaces.SISO
  "Single Input Single Output continuous control block"
  extends BlockIcon;

  RealInput u "Connector of Real input signal";
  RealOutput y "Connector of Real output signal";
end SISO;

block Modelica.Blocks.Continuous.Derivative
```

```
  "Approximated derivative block"
  import Modelica.Blocks.Types.Init;
  parameter Real k=1 "Gains";
  parameter SIunits.Time T(min=Modelica.Constants.small) = 0.01
    "Time constants (T>0 required; T=0 is ideal derivative block)";
  parameter Init.Temp initType=Modelica.Blocks.Types.Init.NoInit
    "Type of initialization";
  parameter Real x_start=0 "Initial or guess value of state";
  parameter Real y_start=0 "Initial value of output (= state)";
  extends Interfaces.SISO;

  output Real x(start=x_start) "State of block";

protected
  parameter Boolean zeroGain = abs(k) < Modelica.Constants.eps;
initial equation
  if initType == Init.SteadyState then
    der(x) = 0;
  elseif initType == Init.InitialState then
    x = x_start;
  elseif initType == Init.InitialOutput then
    if zeroGain then
       x = u;
    else
       y = y_start;
    end if;
  end if;
equation
  der(x) = if zeroGain then 0 else (u - x)/T;
  y = if zeroGain then 0 else (k/T)*(u - x);
end Derivative;

block Modelica.Blocks.Math.Add3 "Output the sum of the three inputs"
  extends Interfaces.BlockIcon;

  parameter Real k1=+1 "Gain of upper input";
  parameter Real k2=+1 "Gain of middle input";
  parameter Real k3=+1 "Gain of lower input";
  input Interfaces.RealInput u1 "Connector 1 of Real input signals";
  input Interfaces.RealInput u2 "Connector 2 of Real input signals";
  input Interfaces.RealInput u3 "Connector 3 of Real input signals";
  output Interfaces.RealOutput y "Connector of Real output signals";

equation
  y = k1*u1 + k2*u2 + k3*u3;
end Add3;

block Modelica.Blocks.Nonlinear.Limiter "Limit the range of a signal"
  parameter Real uMax=1 "Upper limits of input signals";
  parameter Real uMin= -uMax "Lower limits of input signals";
  parameter Boolean limitsAtInit = true
    "= false, if limits are ignored during initializiation (i.e.,
y=u)";
  extends Interfaces.SISO;

equation
  assert(uMax >= uMin, "Limiter: Limits must be consistent. However,
uMax (=" + String(uMax) +
                       ") < uMin (=" + String(uMin) + ")");
  if initial() and not limitsAtInit then
     y = u;
```

```
     assert(u >= uMin - 0.01*abs(uMin) and
            u <= uMax + 0.01*abs(uMax),
           "Limiter: During initialization the limits have been
ignored.\n"+
           "However, the result is that the input u is not within the
required limits:\n"+
           "  u = " + String(u) + ", uMin = " + String(uMin) + ", uMax
= " + String(uMax));
  else
     y = smooth(0,if u > uMax then uMax else if u < uMin then uMin
else u);
  end if;
end Limiter;

block Modelica.Blocks.Sources.Constant
  "Generate constant signal of type Real"
  parameter Real k=1 "Constant output value";
  extends Interfaces.SO;

equation
  y = k;
end Constant;

partial block Modelica.Blocks.Interfaces.SO
  "Single Output continuous control block"
  extends BlockIcon;

  RealOutput y "Connector of Real output signal";
end SO;

partial block Modelica.Blocks.Interfaces.SVcontrol
  "Single-Variable continuous controller"
  extends BlockIcon;

  RealInput u_s "Connector of setpoint input signal";
  RealInput u_m "Connector of measurement input signal";
  RealOutput y "Connector of actuator output signal";
end SVcontrol;

model Modelica.Blocks.Tables.CombiTable1D
  "Table look-up in one dimension (matrix/file) with n inputs and n
outputs "
  import Modelica.Blocks.Types;
  parameter Boolean tableOnFile=false
    "true, if table is defined on file or in function usertab";
  parameter Real table[:, :]=fill(0.0,0,2) "table matrix (grid = first
column)";
  parameter String tableName="NoName"
    "table name on file or in function usertab (see docu)";
  parameter String fileName="NoName" "file where matrix is stored";
  parameter Integer columns[:]=2:size(table, 2)
    "columns of table to be interpolated";
  parameter Blocks.Types.Smoothness.Temp
smoothness=Types.Smoothness.LinearSegments
    "smoothness of table interpolation";
  extends Modelica.Blocks.Interfaces.MIMOs(final n=size(columns, 1));
protected
  Real tableID;
equation
  if tableOnFile then
```

```
    assert(tableName<>"NoName", "tableOnFile = true and no table name
given");
  end if;
  if not tableOnFile then
    assert(size(table,1) > 0 and size(table,2) > 0, "tableOnFile =
false and parameter table is an empty matrix");
  end if;

  for i in 1:n loop
    y[i] = if not tableOnFile and size(table,1)==1 then
              table[1, columns[i]] else dymTableIpo1(tableID,
columns[i], u[i]);
  end for;
  when initial() then
    tableID=dymTableInit(1.0, smoothness, if tableOnFile then
tableName else "NoName", if tableOnFile then fileName else "NoName",
table, 0.0);
  end when;
end CombiTable1D;

partial block Modelica.Blocks.Interfaces.MIMOs
  "Multiple Input Multiple Output continuous control block with same
number of inputs and outputs"

  extends BlockIcon;
  parameter Integer n=1 "Number of inputs (= number of outputs)";
  RealInput u[n] "Connector of Real input signals";
  RealOutput y[n] "Connector of Real output signals";
end MIMOs;

model Modelica.Blocks.Tables.CombiTable2D
  "Table look-up in two dimensions (matrix/file) "

  import Modelica.Blocks.Types;
  extends Modelica.Blocks.Interfaces.SI2SO;

  parameter Boolean tableOnFile=false
    "true, if table is defined on file or in function usertab";
  parameter Real table[:, :]=fill(0.0,0,2)
    "table matrix (grid u1 = first column, grid u2 = first row)";
  parameter String tableName="NoName"
    "table name on file or in function usertab (see docu)";
  parameter String fileName="NoName" "file where matrix is stored";
  parameter Blocks.Types.Smoothness.Temp
smoothness=Types.Smoothness.LinearSegments
    "smoothness of table interpolation";
protected
  Real tableID;
equation
  if tableOnFile then
    assert(tableName<>"NoName", "tableOnFile = true and no table name
given");
  end if;
  if not tableOnFile then
    assert(size(table,1) > 0 and size(table,2) > 0, "tableOnFile =
false and parameter table is an empty matrix");
  end if;

  y = dymTableIpo2(tableID, u1, u2);
  when initial() then
```

```
    tableID=dymTableInit(2.0, smoothness, if tableOnFile then
tableName else "NoName", if tableOnFile then fileName else "NoName",
table, 0.0);
  end when;
end CombiTable2D;

block Modelica.Blocks.Sources.RealExpression
  "Set output signal to a time varying Real expression"
  Blocks.Interfaces.RealOutput y=0.0 "Value of Real output";


end RealExpression;
```

Anexo B: Conversión de TMY a ParaTrough

En este anexo se detallan los pasos a seguir para convertir un archivo TMY procedente del SAM en un *record* de *ParaTrough* utilizable por los demás elementos de la librería. Los formatos de origen del TMY pueden ser o TMY2 o INTL.

1. Descarga e instalación del SAM

Si no se tiene aún descargado el software SAM, se debe visitar su página web [4] y descargarlo. Está disponible para sistemas operativos Windows, OS y Linux.

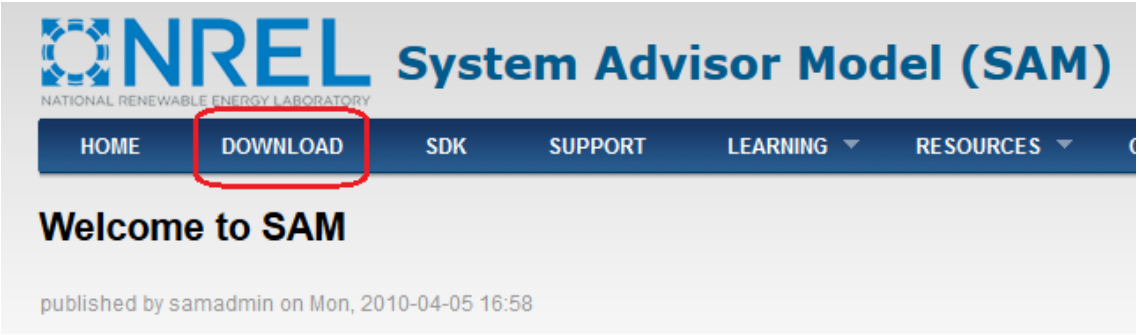


Figura B.1: Descarga del SAM en la página web del NREL

2. Acceder al software SAM y a los ficheros TMY

Se debe acceder al software SAM una vez se haya descargado. Se inicia un nuevo proyecto de cualquier tecnología CSP.

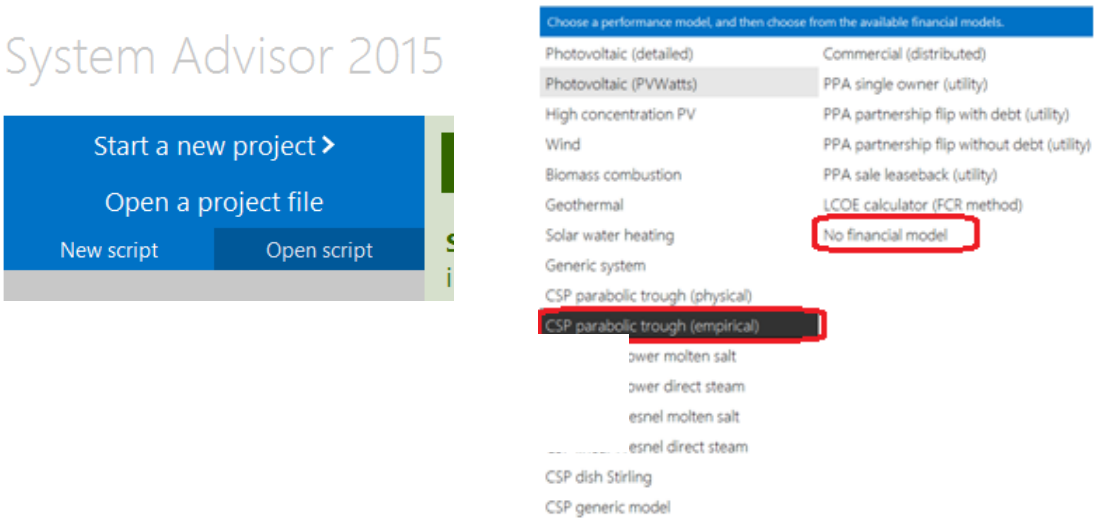
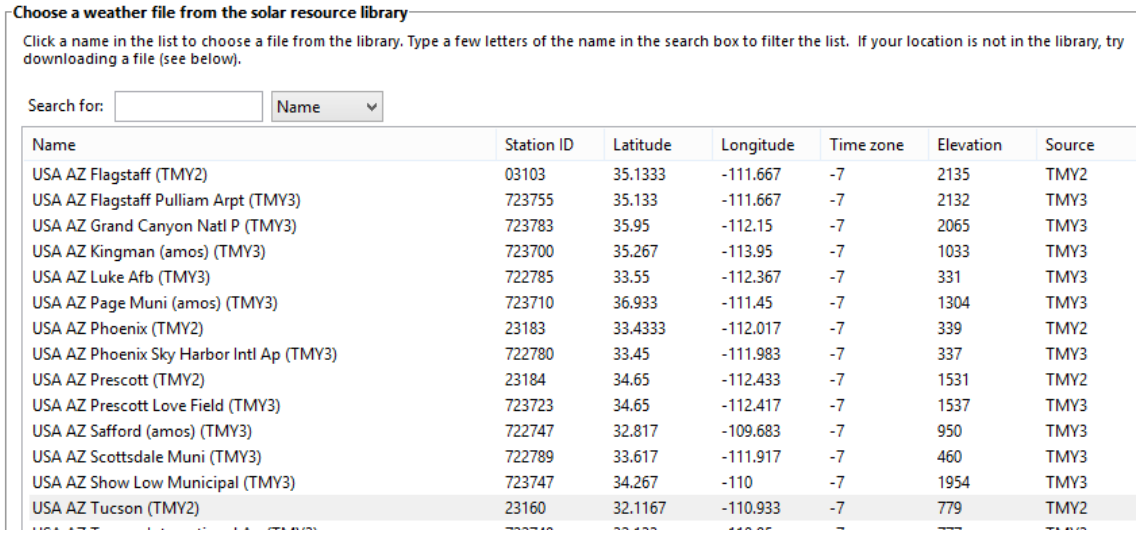


Figura B.2: Iniciando un proyecto CSP de colector parabólico sin modelo financiero en SAM

3. Acceder a los datos TMY de las localizaciones de SAM

Una vez lanzado el proyecto, la primera página que aparece es la de localizaciones y recursos. En ella aparecen en forma de listado los 1619 diferentes archivos TMY que posee la librería de SAM.



Name	Station ID	Latitude	Longitude	Time zone	Elevation	Source
USA AZ Flagstaff (TMY2)	03103	35.1333	-111.667	-7	2135	TMY2
USA AZ Flagstaff Pulliam Arpt (TMY3)	723755	35.133	-111.667	-7	2132	TMY3
USA AZ Grand Canyon Natl P (TMY3)	723783	35.95	-112.15	-7	2065	TMY3
USA AZ Kingman (amos) (TMY3)	723700	35.267	-113.95	-7	1033	TMY3
USA AZ Luke Afb (TMY3)	722785	33.55	-112.367	-7	331	TMY3
USA AZ Page Muni (amos) (TMY3)	723710	36.933	-111.45	-7	1304	TMY3
USA AZ Phoenix (TMY2)	23183	33.4333	-112.017	-7	339	TMY2
USA AZ Phoenix Sky Harbor Intl Ap (TMY3)	722780	33.45	-111.983	-7	337	TMY3
USA AZ Prescott (TMY2)	23184	34.65	-112.433	-7	1531	TMY2
USA AZ Prescott Love Field (TMY3)	723723	34.65	-112.417	-7	1537	TMY3
USA AZ Safford (amos) (TMY3)	722747	32.817	-109.683	-7	950	TMY3
USA AZ Scottsdale Muni (TMY3)	722789	33.617	-111.917	-7	460	TMY3
USA AZ Show Low Municipal (TMY3)	723747	34.267	-110	-7	1954	TMY3
USA AZ Tucson (TMY2)	23160	32.1167	-110.933	-7	779	TMY2

Figura B.3: Listado de los TMY de localizaciones en SAM

En la parte inferior derecha de la tabla que se refleja en la Figura B.3, hay un botón para acceder a los archivos en formato plano *.csv (separado por comas)*. Acceda.

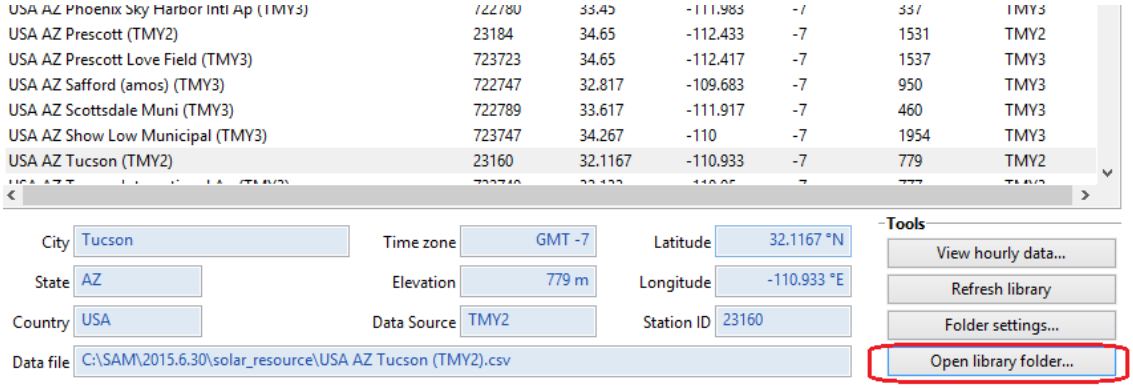


Figura B.4: Botón de acceso los archivos *csv* de los TMY

Cuando se pulsa el botón se abre la carpeta donde se encuentran estos archivos.

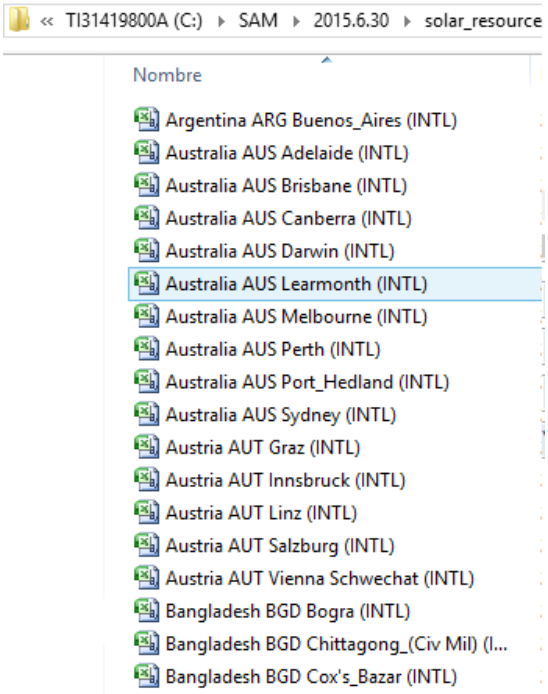


Figura B.5: Carpeta con los archivos TMY en formato *csv*

4. Copiar los datos al conversor TMY2 o conversor INTL

Se abre el archivo TMY del SAM que desee y copie los datos desde la columna 4 hasta el final.

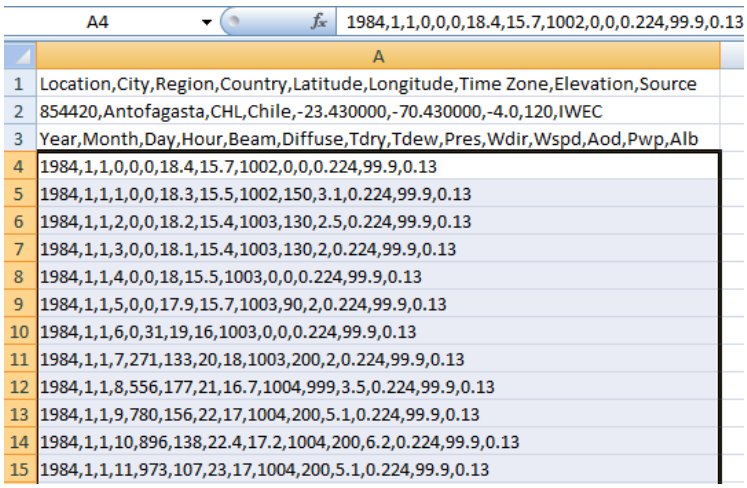
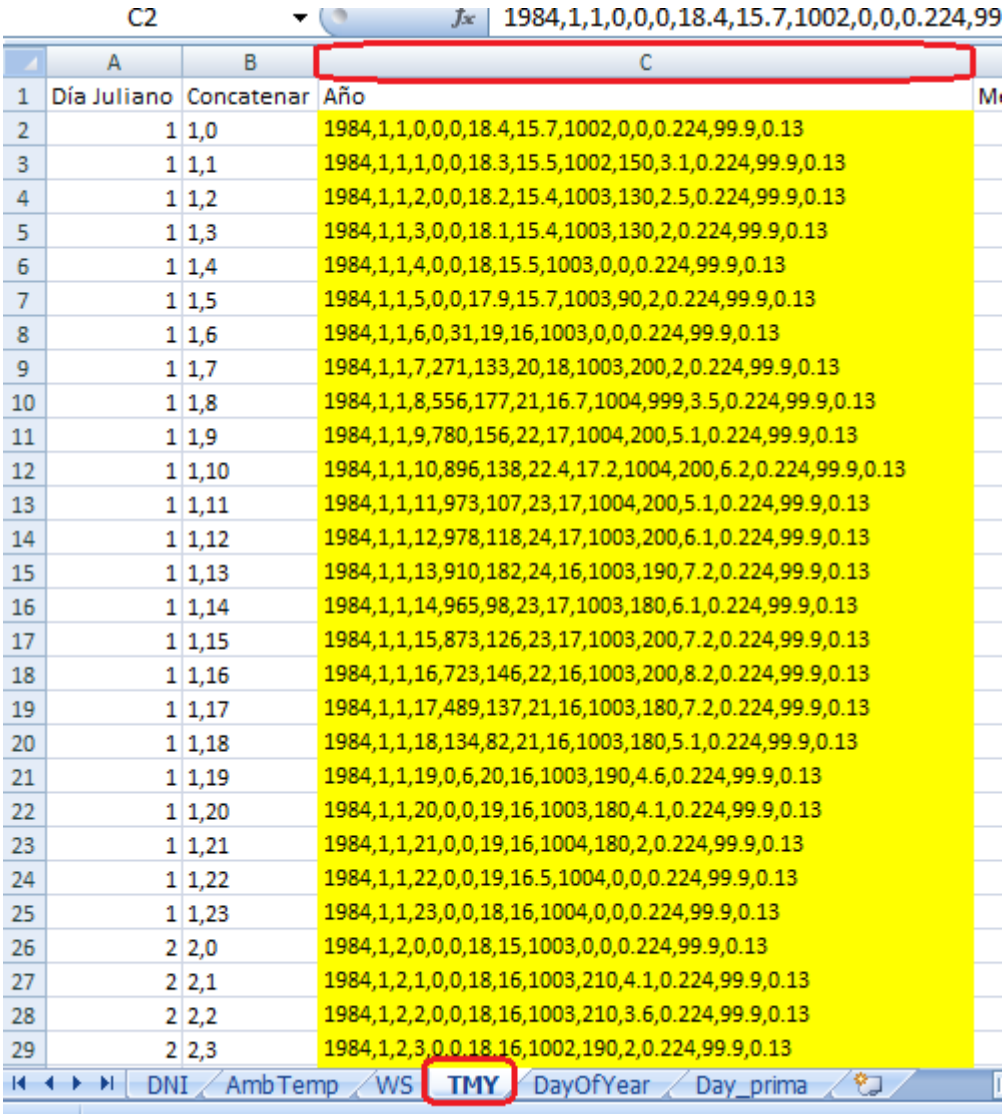


Figura B.6: Copiado de los datos del TMY de Antofagasta (Chile)

A continuación se abre el archivo *INTLconverter.xlsx* o el archivo *TMY2converter.xlsx*, dependiendo si los datos del SAM están en formato INTL o TMY2, respectivamente. Estos dos archivos son suministrados en el CD-rom anexo al presente TFM. En el caso del ejemplo de la Figura B.6, los datos de Antofagasta (Chile) están en INTL, por lo que se abre el *INTLconverter.xlsx.*

Se pegan los valores anteriormente copiados en la columna C (marcada en amarillo) de la pestaña TMY.



C2 — 1984,1,1,0,0,0,18.4,15.7,1002,0,0,0.224,99

	A	B	C
1	Día Juliano	Concatenar	Año
2	1	1,0	1984,1,1,0,0,0,18.4,15.7,1002,0,0,0.224,99.9,0.13
3	1	1,1	1984,1,1,1,0,0,18.3,15.5,1002,150,3.1,0.224,99.9,0.13
4	1	1,2	1984,1,1,2,0,0,18.2,15.4,1003,130,2.5,0.224,99.9,0.13
5	1	1,3	1984,1,1,3,0,0,18.1,15.4,1003,130,2,0.224,99.9,0.13
6	1	1,4	1984,1,1,4,0,0,18,15.5,1003,0,0,0.224,99.9,0.13
7	1	1,5	1984,1,1,5,0,0,17.9,15.7,1003,90,2,0.224,99.9,0.13
8	1	1,6	1984,1,1,6,0,31,19,16,1003,0,0,0.224,99.9,0.13
9	1	1,7	1984,1,1,7,271,133,20,18,1003,200,2,0.224,99.9,0.13
10	1	1,8	1984,1,1,8,556,177,21,16.7,1004,999,3.5,0.224,99.9,0.13
11	1	1,9	1984,1,1,9,780,156,22,17,1004,200,5.1,0.224,99.9,0.13
12	1	1,10	1984,1,1,10,896,138,22.4,17.2,1004,200,6.2,0.224,99.9,0.13
13	1	1,11	1984,1,1,11,973,107,23,17,1004,200,5.1,0.224,99.9,0.13
14	1	1,12	1984,1,1,12,978,118,24,17,1003,200,6.1,0.224,99.9,0.13
15	1	1,13	1984,1,1,13,910,182,24,16,1003,190,7.2,0.224,99.9,0.13
16	1	1,14	1984,1,1,14,965,98,23,17,1003,180,6.1,0.224,99.9,0.13
17	1	1,15	1984,1,1,15,873,126,23,17,1003,200,7.2,0.224,99.9,0.13
18	1	1,16	1984,1,1,16,723,146,22,16,1003,200,8.2,0.224,99.9,0.13
19	1	1,17	1984,1,1,17,489,137,21,16,1003,180,7.2,0.224,99.9,0.13
20	1	1,18	1984,1,1,18,134,82,21,16,1003,180,5.1,0.224,99.9,0.13
21	1	1,19	1984,1,1,19,0,6,20,16,1003,190,4.6,0.224,99.9,0.13
22	1	1,20	1984,1,1,20,0,0,19,16,1003,180,4.1,0.224,99.9,0.13
23	1	1,21	1984,1,1,21,0,0,19,16,1004,180,2,0.224,99.9,0.13
24	1	1,22	1984,1,1,22,0,0,19,16.5,1004,0,0,0.224,99.9,0.13
25	1	1,23	1984,1,1,23,0,0,18,16,1004,0,0,0.224,99.9,0.13
26	2	2,0	1984,1,2,0,0,0,18,15,1003,0,0,0.224,99.9,0.13
27	2	2,1	1984,1,2,1,0,0,18,16,1003,210,4.1,0.224,99.9,0.13
28	2	2,2	1984,1,2,2,0,0,18,16,1003,210,3.6,0.224,99.9,0.13
29	2	2,3	1984,1,2,3,0,0,18,16,1002,190,2,0.224,99.9,0.13

DNI | AmbTemp | WS | TMY | DayOfYear | Day_prima

Figura B.7: Pegado de los datos de Antofagasta (Chile) en el conversor *INTLConverter.xlsx*

5. Adecuación de los datos

Se seleccionan todos los datos de la columna C, desde la fila 2 hasta el final y se dividen los datos separados por comas en columnas, utilizando para ello la aplicación empotrada que suministra Excel.

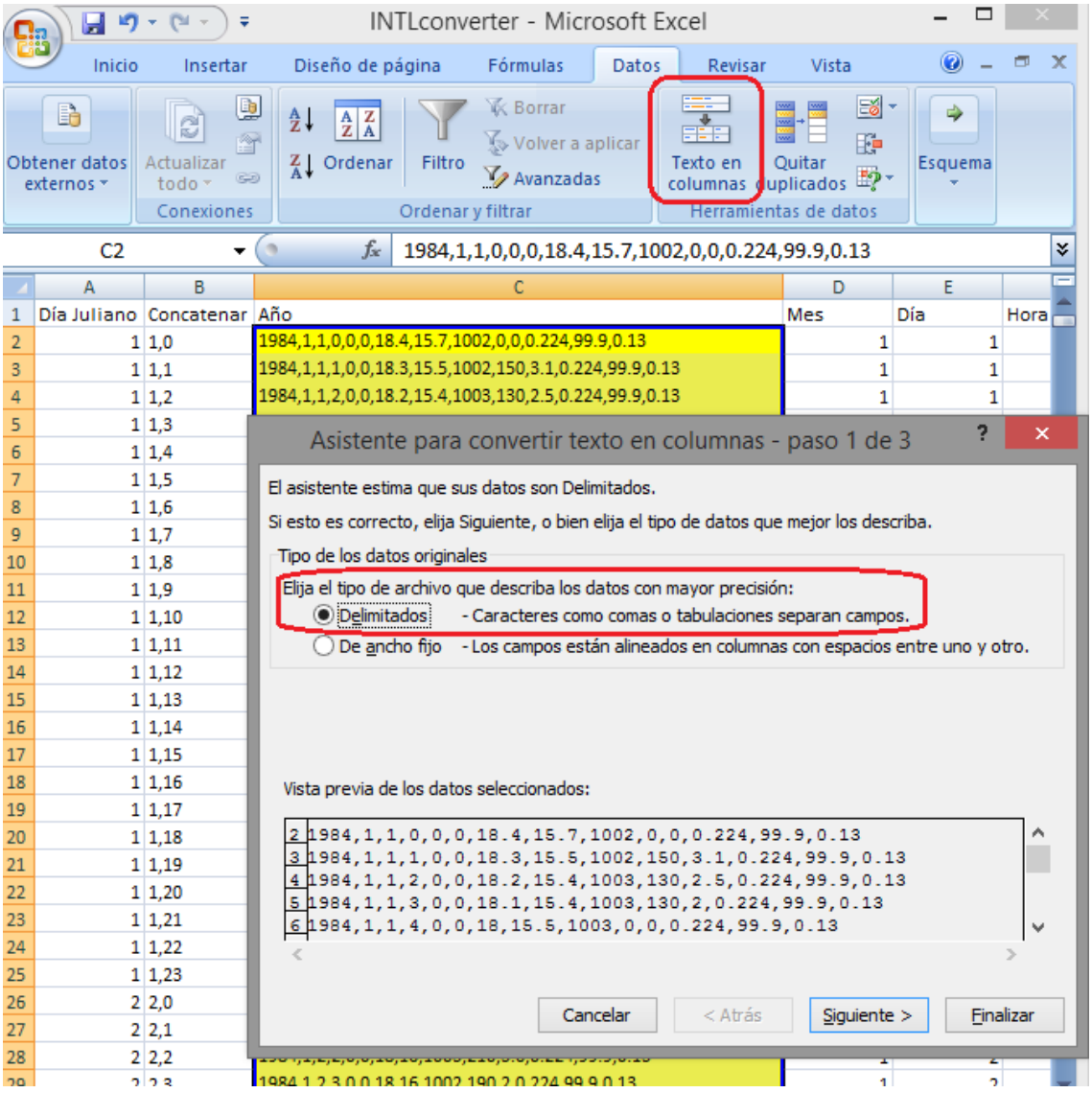


Figura B.8: Distribución del texto plano separado por comas en columnas

En el siguiente paso del asistente de Excel, se escoge como elemento separador la coma.

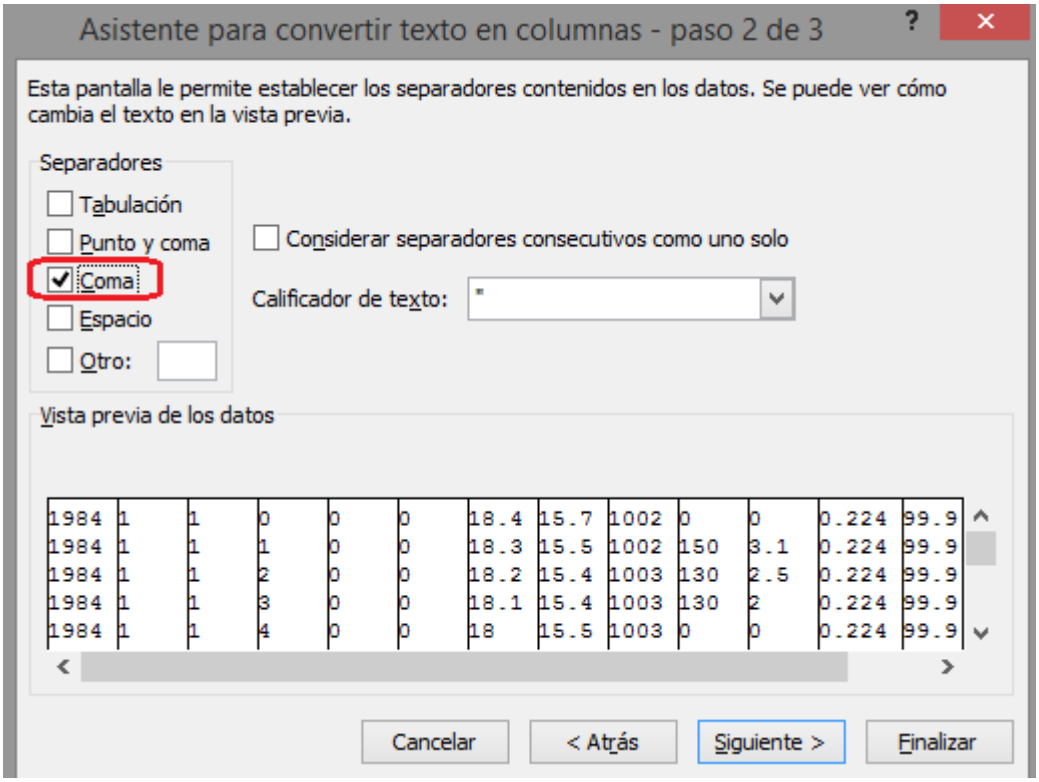


Figura B.9: Selección de la coma como elemento separador del texto plano

Cuando se finaliza el proceso, los datos que estaban separados por comas y dispuestos en una misma columna, se distribuyen entre las demás columnas que previamente están especificadas para albergar los datos correctos.

Año	Mes	Día	Hora	DNI	DHI	Tdry	Tdew	Pres	Wdir	WS	Aod	Pwp	Alb
1984	1	1	0	0	0	18,4	15,7	1002	0	0	0,224	99,9	0,13
1984	1	1	1	0	0	18,3	15,5	1002	150	3,1	0,224	99,9	0,13
1984	1	1	2	0	0	18,2	15,4	1003	130	2,5	0,224	99,9	0,13
1984	1	1	3	0	0	18,1	15,4	1003	130	2	0,224	99,9	0,13
1984	1	1	4	0	0	18	15,5	1003	0	0	0,224	99,9	0,13

Figura B.10: Distribución de los datos en las columnas previamente definidas

Se observa en la Figura B.10 que los datos ya quedan ordenados en sus respectivas columnas. Hay que indicar que el punto es el separador decimal en los archivos *.csv*. Si en la configuración regional de Microsoft Office el separador decimal es la coma (usual para usuarios centroeuropeos), basta con reemplazar todos los puntos (“.”) por comas (“,”) de la pestaña TMY para obtener los datos en el formato adecuado.

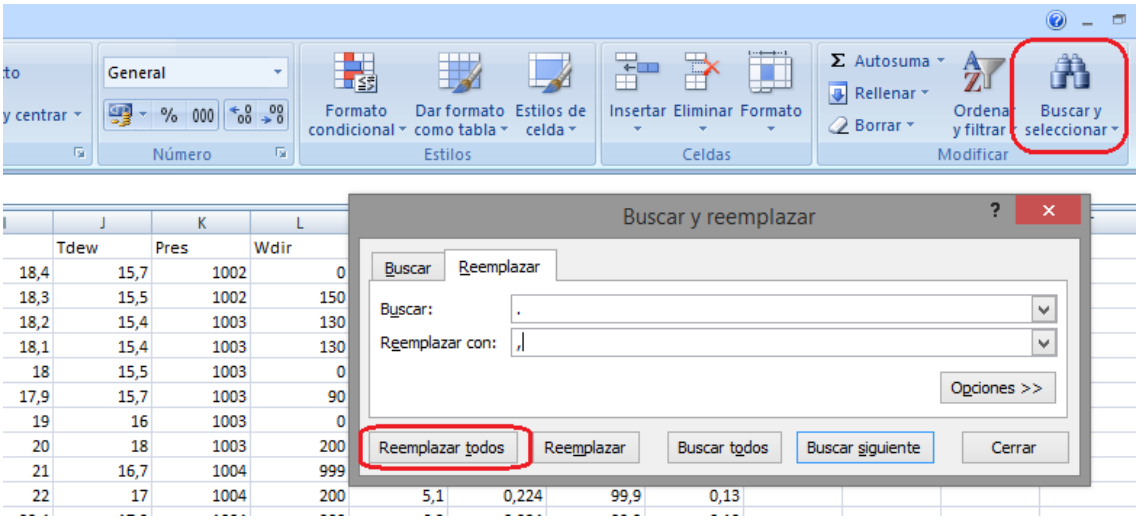


Figura B.11: Reemplazo de punto por coma como separador decimal

6. Carga de datos en *ParaTrough*

Una vez realizada la adecuación de datos descrita en el apartado 5 de este anexo, se crean mediante fórmulas preestablecidas en la aplicación las pestañas DNI, AmbTemp y WS.

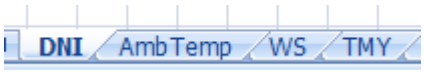


Figura B.12: Pestañas de datos legibles por *ParaTrough*

En estas pestañas los datos procedentes del SAM ya tienen un formato legible por *ParaTrough* ya que se distribuyen de la misma forma que lo que marca la Tabla 5.2.

ParaTrough también lee datos en formato *.csv* por lo que para incluir estos datos con el formato adecuado, es necesario guardar cada una de las pestañas como un archivo de extensión *.csv* separado por comas.

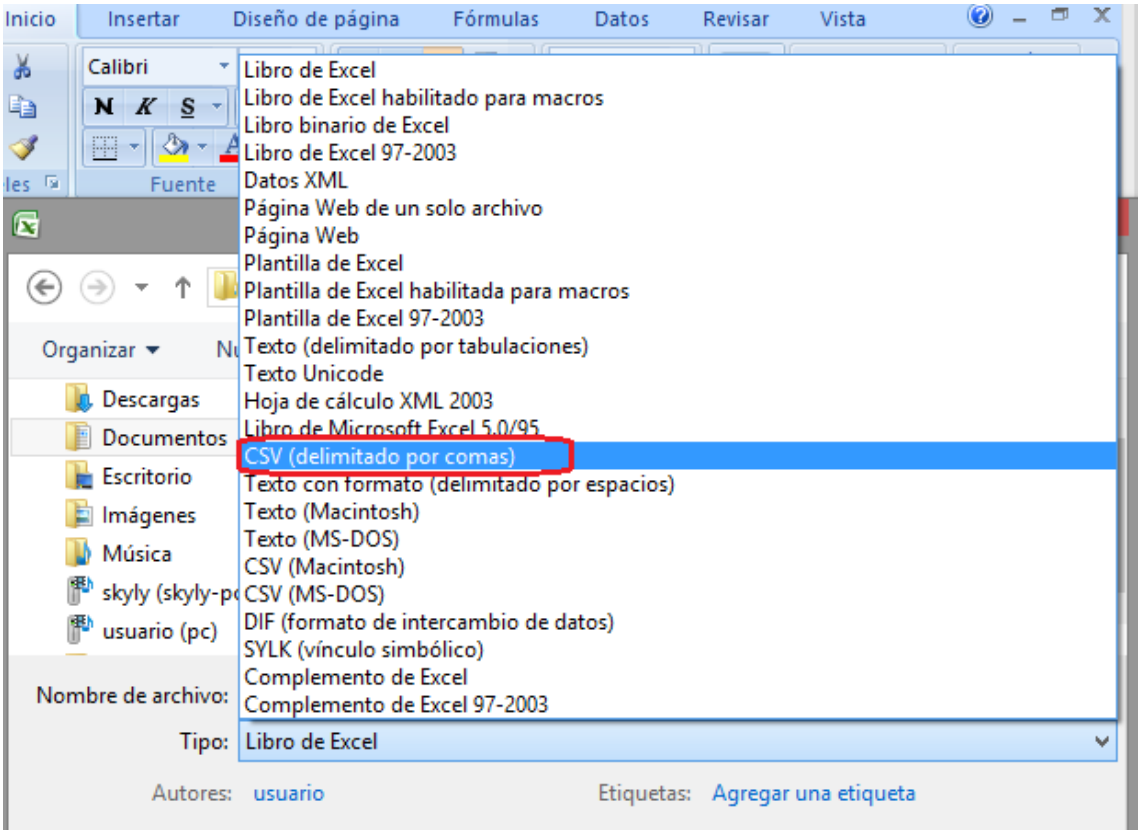


Figura B.13: Guardar como archivo CSV (delimitado por comas)

Se guarda este archivo en el directorio que se desee y con el nombre que se desee. A este nombre y en este directorio tiene que buscar después *ParaTrough* para tomar los datos y cargarlos a la librería.

Se abre *Dymola* y se carga la librería *ParaTrough*. En el entorno gráfico del modelo *A_SolarResource.Tables.Generic2DDataTable* se hace doble click sobre el sub-modelo *CombiTable2D*.

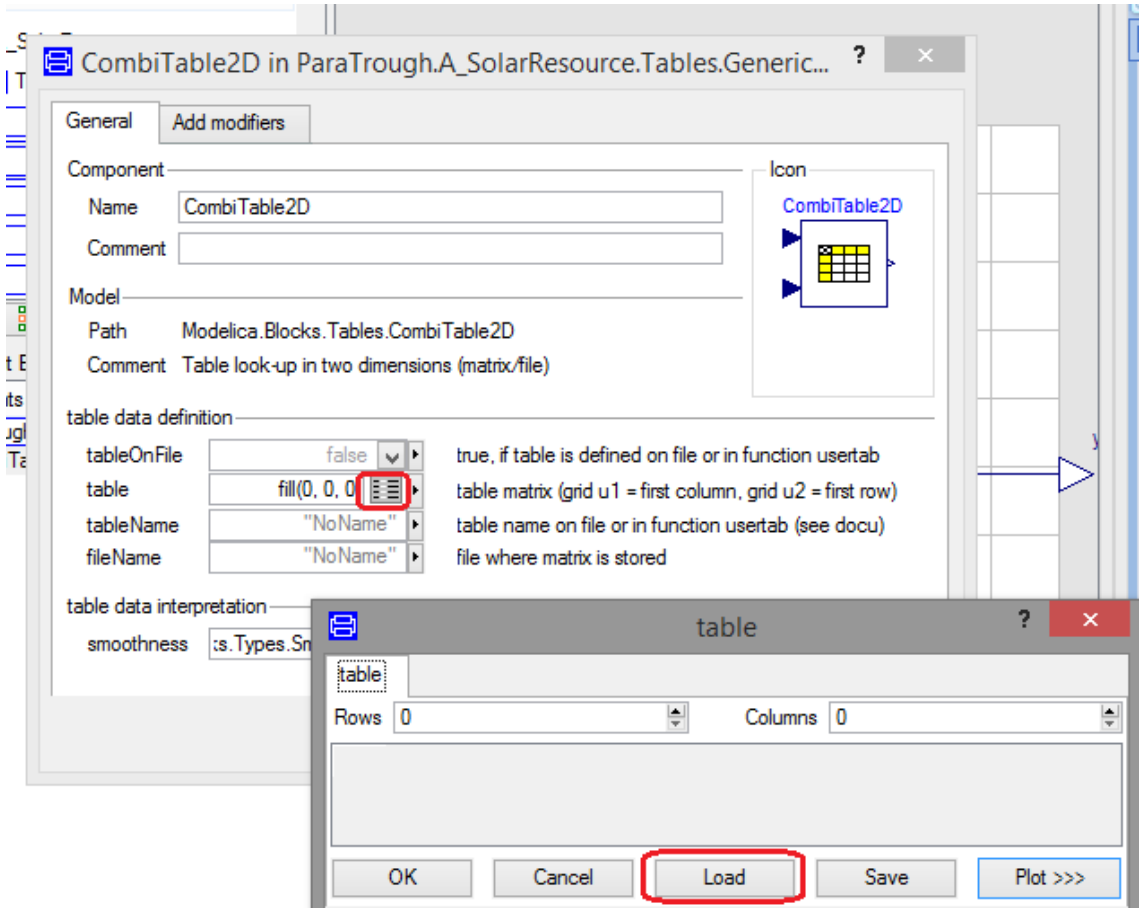


Figura B.14: Cuadro desplegable del sub-modelo *CombiTable2D* del modelo *Generic2DDataTable*

Se abrirá un cuadro desplegable tal como se muestra en la Figura B.14. Se pulsa sobre el recuadro a la derecha del campo *table* y saldrá otro cuadro desplegable. En éste, se pulsa el botón *Load* para cargar los datos deseados, que son los que anteriormente se han guardado en archivo *.csv* desde el conversor adecuado. Tras un breve periodo de procesamiento, se carga una tabla de 24 filas y 366 columnas de acuerdo al formato de la Tabla 5.2. De esta manera la tabla se introduce en *ParaTrough* en el lenguaje *Modelica*.

Se copia el código de la tabla y se introduce en el *record* correspondiente de los datos meteorológicos (previamente creado, véase apartado 5.4.3.1).

```
record Antofagasta_Chile
extends data(
    latitude=-23.43,
    longitude=-70.43,
    meridian=-60,
    DNI=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
        27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
        51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,
        75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,
        99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,
        117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,
        135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,
        153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,
        171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,
        189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,
        207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,
        225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,
        243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,
        261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,
        279,280,281,282,283,284,285,286,287,288,289,290,291,292,293,294,295,296,
        297,298,299,300,301,302,303,304,305,306,307,308,309,310,311,312,313,314,
```

Figura B.15: Introducción de la tabla de datos de DNI en el *record Antofagasta_Chile*

El mismo procedimiento se tiene que hacer para introducir los datos de temperatura ambiente (*AmbTemp*) y velocidad del viento (*WindSpeed*) a partir de las pestañas correspondientes del conversor.

Con este procedimiento descrito en este Anexo B: Conversión de TMY a ParaTrough, se tiene la posibilidad de utilizar en *ParaTrough* los 1619 archivos TMY que contiene SAM.