Guia para la programación de modelos en SiManFor

Detalles necesarios para la introducción exitosa de modelos en la plataforma

Felipe Bravo, Celia Herrero, Cristóbal Ordóñez

12 de noviembre de 2017

Este manual de SiManFor pretende facilitar a los modelizadores la programación de modelos en la plataforma. SiManFor es una aplicación Web que permite la simulación de alternativas de manejo forestal sostenible y está compuesto por distintos módulos(gestión de inventarios, simulación, predicción y proyección, sistemas de consultas, salidas de simulaciones y sistema de seguridad) y admite diferentes roles para los usuarios: administrador y responsable de la gestión del programa, los desarrolladores o programadores, los modelizadores o autores de modelos forestales y los usuarios finales. El principal objetivo de SiManFor es la creación de una comunidad de modelizadores y usuarios que fortalezca el uso de los modelos en la gestión forestal sostenible.

Índice de contenidos

1. Introducción

SiManFor es una herramienta que permite ejecutar modelos de crecimiento sobre inventarios forestales para distintas especies forestales. Con ellos se pueden ejecutar "escenarios selvícolas" que además de simular crecimiento permiten incluir intervenciones selvícolas, análisis de mortalidad y regeneración, factores de competencia, etc. Los usuarios con permisos de modelizador pueden programar estos modelos, para lo que es necesario conocer los detalles técnicos de los mismos y la forma en que son cargados y ejecutados por la aplicación. En este manual se detallan los conceptos y tareas necesarios para realizar esta labor.

2. Detalles técnicos de SiManFor

2.1. Lenguaje de programación

La plataforma esta desarrollada en entorno Visual.NET, y el lenguaje de programación empleado es C#. Ademas es el empleado en la codificacion de los modelos. Este lenguaje es muy simple y permite un aprendizaje rápido en caso de que no se haya usado antes.

Es recomendable consultar algún manual de C# básico y los detalles técnicos del propio lenguaje cuando se encuentren dificultades, si bien en este documento intentaremos plasmar los conocimientos básicos que se requieren para codificar cualquier modelo.

2.2. Estructura de los modelos

Los modelos de crecimiento no son más que clases (estructuras) que agrupan un conjunto de funciones y procedimientos, que son las que contienen los algoritmos que hay que ejecutar en cada fase. Existe una función o procedimiento que contiene las ecuaciones o algoritmos que se deben aplicar en cada una de las fases indicadas anteriormente. Es importante saber que el motor de SiManFor es el que se encarga de ejecutar cada una de ellas en el momento oportuno, y que su posición dentro del código no es relevante. Es decir, aunque se recomienda un orden concreto de aparición y definición de cada función o procedimiento, no es estrictamente necesario mantenerlo. En SiManFor se pueden programar dos tipos de modelo diferentes: modelos de masa y modelos de árbol individual. Para crear y modificar modelos se puede utilzar la herramienta gratuita de MS Visual Studio, aunque es posible instalar un editor de textos ligero que permite resaltar el código como Notepad++ (descarga gratuita en `http://notepad-plus-plus.org/`) o jEdit (descarga gratuita en `http://jedit.org/index.php?page=download`) que resalta las palabras reservadas del ***script***, lo que facilita la edición.

2.3. Fundamentos de C#para SiManFor

Aunque C# es un lenguaje de programación bastante intuitivo y está bien estructurado, puede resultar complicado para el usuario que se acerca a él por primera vez. Afortuunadamente es posible encontrar multitud de información online, como la página oficial de Microsoft (MSDN Library en Español `https://msdn.microsoft.com/es-es/library/kx37x362.aspx`), que incluye los principios del lenguaje, su sintaxis y ejemplos de cómo programar de forma general en este lenguaje. Tambien es posible encontrar guias y manuales de programación como por ejemplo el siguiente escrito en castellano: (`www.nachocabanes.com`)

2.3.1. Biblioteca de clases

Asociado al lenguaje de programación, existe un conjunto de clases que se pueden usar para definir el tipo de las variables que se usen: números enteros o reales, cadenas de texto, vectores y matrices, etc. Además, existe una clase denominada Math que agrupa todas las funciones matemáticas y constantes numéricas comunes: número pi y número e, funciones de logaritmos, medias aritméticas, trigonométricas, etc. Al final de este documento existe una relación de enlaces desde los que se pueden consultar los detalles de las bibliotecas de clases y las funciones

que contiene la clase Math. En estos enlaces también se pueden consultar ejemplos prácticos para entender el funcionamiento y uso de los mismos.

Los siguientes enlaces muestran las secciones de documentación de la biblioteca estándar de clases y la documentación de Math (también accesible desde el primer enlace), respectivamente. Las clases indicadas en dicha documentación pueden usarse en los modelos, aunque se deben tener en cuenta las limitaciones de seguridad impuestas por el motor de ejecución de modelos. Biblioteca estándar de clases de .NET Framework MSDN Library (Español `http://msdn.microsoft.com/es-es/library/ms229335.aspx` Referencia de la clase Math MSDN Library (Español `http://msdn.microsoft.com/es-es/library/system.math.aspx`)

2.3.2. Tipos de variable

En C# las variable pueden ser de distintos tipos. Antes de utilizar una variable es necesario **declararla**, indicando qué tipo de valores va a almacenar. Ademas es posible **inicializarla**, asignandole un valor inicial. Indicamos el código para declarar e inicializar distintos tipos de variable:

```
sbyte     sbVariable = 0  variable for integer in range [-128, 127]
byte      bVariable  = 0  range [0, 255]
short     sVariable  = 0  range  [-32768, 32767]
ushort    usVariable = 0  range  [0, 65535]
int       iVariable  = 0  range  [-2147483648, 2147483647]
uint      uiVariable = 0  range  [0, 4294967295]
long      lVariable  = 0  range  [-9223372036854775808, 9223372036854775807]
ulong     ulVariable = 0  range  [0, 18446744073709551615]

float     fVariable  = 0  real [-1,5·10^-45, 3.4*10^38] and 7 decimals
double    dVariable  = 0  [5,0*10^-324, 1,7*10^308] and 15-16 decimals
decimal   deVariable = 0  [1,0*10^-28, 7,9*10^28] and 28-29 decimals

string    ulVariable = 0  variable for text

bool      ulVariable = 0  variable for binary

int[]     iaVariable = {0,0,0,0}  array with four integer numbers
float[][] faVariable = [4][3]     matrix for real numbers with 4 rows and 3 columns
```

2.3.3. Operadores

Podemos utilizar tres tipos de operadores en C#: aritméticos, que son los más intuitivos; relacionales, que permiten realizar comparaciones entre variables. y lógicos, que permiten realizar operaciones entre variables booleanas.

Operadores aritméticos

operador	operación
+	suma
-	resta
*	multiplicación
/	división
%	resto de la división

Operadores relacionales

operador	operación
<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual a
!=	distinto de

Operadores booleanos

operador	operación
&&	Y
||	O
!	No

2.3.4. Funciones

En C# lo habitual es programar de forma modular, con una función principal, **Main**, o cuerpo del programa, y llamadas a funciones subordinadas. En SiManFor hay un código que no se puede modificar y una serie fija de funciones que puede "rellenar" el modelizador, pero cuyas variables de entrada y tipos están ya fijados.

Al igual que las variables, las funciones hay que declararlas. Es necesario decir si serán visibles en todo el programa (*public*) o no (*private*), si va a devolver un valor, en cuyo caso hay que indicar de qué tipo es o no (indicándolo con *void*), y también hay que indicar si sobreescribimos la función por defecto (que está vacia), en cuyo caso lo indicaremos con *override*. En los programas de SiManFor estos valores estan fijados y debemos conocer de qué tipo es cada función, pero no podemos modificarlo.

2.3.5. Estructuras de control

No es habitual que en SiManFor sea necesario utilizar estructuras complicadas, pero si que es necesario utilizar bucles para recorrer los árboles de una parcela o poneer condiciones para realizar, o no, un cálculo determinado. Veamos pues que estructuras son útiles para nuestros propósitos.

Condicionales: if ... else Esta estructura permite hacer una comprobación y en caso de que el resultaado sea positivo ejecutar una sentencia o ejecutar otra en caso de que no se cumpla la condición; En el ejemplo siguiente en el que comprobamos si la variable altura dominante tiene un valor y en función del resultado se ejecuta una u otra sentencia:

```
  if (plot.H_Dominante.HasValue)
     tree.Altura = plot.H_Dominante.Value + Random();
  else
     tree.Altura = 15 + Random();
```

La condición debe ir siempre entre paréntesis y las sentencias siempre terminadas en punto y coma. Si se desea ejecutar más de una sentencia, estas se pueden poner entre llaves.

Condicionales múltiples: switch ... case Para poder obtener multiples valores a partir de la evaluación de una variable utilizaremos un bucle switch:

```
switch (expression)
{
  case value1: sentence1;
    break;
  case value2: sentence2;
    sentence2b;
    break;
  ...
  case valueN: sentenceN;
    break;
  default:
    otherSentence;
break;
}
```

Bucles while Cuando el número de veces que se va a repetir el bucle es desconocido a priori, se utiliza un bucle while, con la condición evaluada al principio si existe la posibilidad de que no se ejecute ninguna vez, o al final si se desea ejecutarlo al menos una vez.

```
int n = 1;
while (n < 6)
{
  Console.WriteLine(n);
  n = n + 1;
}
```

cuando la condición va al final:

```
do
   sentences;
while (condition)
```

Bucles for Cuando conocemos el número de veces que queremos repetir un bucle utilizamos una sentencia **for**

```
for (InitialValue; RepetitionCondition; Increment)
   Sentence;
for (counter=1; counter<=10; counter++)
   Console.Write("{0} ", counter);
```

Si queremos incluir una condición para salir de un bucle for es posible con la sentencia **continue** dentro de una condición **if**:

```
for (counter=1; counter<=10; counter++)
{
  if (counter==5)
  continue;
  Console.Write("{0} ", counter);
}
```

2.3.6. Entornos, Espacios de nombres, Clases y objetos

Al principio del archivo de código es necesario incluir unas pocas líneas que van a indicar al motor de SiManFor qué entornos genéricos del lenguaje de programación y de la propia plataforma es necesario cargar. Estos entornos son iguales en los dos tipos de modelo, como se puede apreciar a continuación en la definición de cada uno de ellos.

C# es un lenguaje orientado a objetos, y como tal está formado por clases que a su vez están compuestas por objetos. Además, para evitar duplicidades o incoerencias en los nombres cuando se comparte código, es habitual que todos los códigos se incluyan dentro de un espacio de nombres que les permite independizarlo de otros. En SiManFor cada modelo está definido en el espacio de nombres **EngineTest** y dentro de la clase **template** cuyos objetos son las distintas funciones que necesita un modelo. Las funciones que no se incluyan tomaran el valor por defecto, que es una función vacia, es decir, que no alterarán el inventario.

Clase para modelo de masa Una clase **template** para un modelo de masa se definiría así:

```
using System;
using System.Collections.Generic;
using Simanfor.Core.EngineModels;
using Simanfor.Entities.Enumerations;
namespace EngineTest
{
    public class MassModelTemplate : MassModelBase
    {

        public override void Initialize(Parcela plot)
        {

        }

        public override void ApplyModel(Parcela oldPlot, Parcela newPlot, int years)
        {

        }

        public override void ApplyCutDown(Parcela oldPlot, Parcela newPlot, CutDownType cutDownType, TrimType trimType, float value)
        {

        }
    }
}
```

Clase para modelo de árbol individual Una clase template para un modelo de árbol individual se definiría así:

```
using System;
using System.Collections.Generic;
using Simanfor.Core.EngineModels;
namespace EngineTest
{

    public class Template : ModelBase
    {
        public override void CalculateInitialInventory(Parcela plot)
        {

        }
```

```
        public override double Survives(double years, Parcela plot, PieMayor tree)
        {
            return 0.0F;
        }

        public override void Grow(double years, Parcela plot, PieMayor oldTree, PieMayor newTree)
        {

        }

        public override double? AddTree(double years, Parcela plot)
        {
            return 0.0F;
        }

        public override Distribution[] NewTreeDistribution(double years, Parcela plot, double AreaBasimetricaIncorporada)
        {
            return distribution;
        }

        public override void ProcessPlot(double years, Parcela plot, PieMayor[] trees)
        {

        }
    }
}
```

2.3.7. Uso de variables en SiManFor

Las variables que se pueden emplear en un modelo en SiManFor son de dos tipos: las definidas por el modelizador o las dependientes de la base de datos almacenada en SiManFor. Para el caso de variables definidas por el modelizador es necesario indicar el "tipo" de variable; por ejemplo para una variable de tipo **numero real** cuyo nombre sea **diametroMinimo** y un valor inicial **0** la instrucción sería:

```
double diametroMinimo = 0;
```

Si las variables dependen de los inventarios almacenados en SiManFor (que se verán detalladas más adelante) no será necesario declararlas para poder utilizarlas, pero hay que seguir escrupulosamente los convencionalismos asociados a la programación orientada a objetos.

Para ello hay que fijarse en las propiedades asociadas al objeto (variable) y tener en cuenta de qué tabla de datos depende: **Parcela** o **PieMayor**. Además, dependiendo de la función con la que se esté trabajando, la forma de acceder a la base de datos será diferente. En el encabezado de cada función se indica de qué forma se accede a los datos, y no se puede modificar. Si por ejemplo queremos acceder a la altura dominante que es una variable de masa dentro de la función *CalculateInitialInventory* y asignarle su valor a una variable temporal *H_Dom*, deberemos escribir el código siguiente:

```
public override void CalculateInitialInventory(Parcela plot)
// The function  declares  -plot- as input variable from the class -Parcela-
  {
    double H_Dom = plot.H_DOMINANTE.Value;
    // The variable temporal H_Dom gets the value of H_DOMINANTE
  }
```

La función *CalculateInitialInventory* accede directamente a los datos de la clase *Parcela* y el nombre que le asigna es *plot*, por lo cual se antepone al nombre de la variable separado por un punto. A continuación se indica la propiedad de la variable que queremos utilizar, el valor en este caso, por lo que lo indicamos con *Value*, también separado por un punto.

Si queremos utilizar o cambiar variables de árbol en una función como la anterior (que no declara un objeto de la base de datos *PieMayor*) podemos acceder a ellas con un bucle que recorre todos los pies mayores que están en el objeto que sí se ha declarado: *plot*. Esto lo podemos hacer de la siguiente forma:

```
public override void CalculateInitialInventory(Parcela plot)
{
 foreach (PieMayor tree in plot.PiesMayores)
  {
   tree.ALTURA = 27; // for every -tree- inside -plot- the variable ALTURA gets the value 27
  }
}
```

Si la función elegida tiene declarada la clase *PieMayor*, la forma de referirse a la variable es más sencilla:

```
public override double Survives(double years, Parcela plot, PieMayor tree)
{
  double treeSurvival = 1.0F;
  tree.ALTURA = plot.H_DOMINANTE.Value +  Random();
  // asignamos a la variable de arbol ALTURA el valor de la altura dominante
  return treeSurvival;
}
```

Si queremos saber si un variable tiene un valor almacenado deberemos acceder a la propiedad *HasValue*, que es de tipo lógico y devuelve **TRUE** cuando tiene un valor y **FALSE** en caso contrario. Un ejemplo de uso puede ser el siguiente:

```
public override double Survives(double years, Parcela plot, PieMayor tree)
{
if !(tree.ALTURA.HasValue)
   tree.ALTURA = plot.H_DOMINANTE.Value +  Random();
}
```

Los inventarios con los que trabajan los modelos son creados internamente por la aplicación durante la ejecución del mismo y pueden proceder de dos fuentes distintas:

1. Inventario original, tras aplicar los criterios de filtrado que el usuario haya seleccionado.
2. Inventario resultante de aplicar un modelo o una corta previamente.

Hay que recordar que en ambos casos el escenario va a utilizar una base de datos temporal, que será una copia reducida del SDM, o el resultado de la aplicación del modelo o la clara programada. Como se indicó al enumerar las variables disponibles en la aplicación, es posible utilizar variables extra con nombre genérico para poder utilizar variables que no se tubieron en cuenta en el momento de diseñar el SDM. Por ejemplo, si queremos utilizar una variable de biomasa (biomasa del fuste), podemos calcular su valor en el modelo, y almacenarlo en una de las variables adicionales (ej. VAR_1). Sabemos que cuando en el output vemos VAR_1, su valor hará referencia a la biomasa del fuste. Si queremos facilitar la lectura al usuario final también podremos modificar el output de forma que se cambie la etiqueta de la columna y en lugar de VAR_1 aparezca *BiomasaFuste*.

3. Escenarios en SiManFor: Fundamentos

Lo primero que necesitamos es conocer la lógica de funcionamiento de la plataforma. Para obtener más detalles se aconseja consultar el manual de usuario de SiManFor. La ejecución de los modelos pasa por varias fases en las que se indica a la plataforma qué árboles (o qué variable de parcela, si el modelo es de masa) debe procesar para obtener un nuevo inventario con los resultados. En los diagramas 1 y 2 se puede ver la estructura de cada tipo de modelo y las funciones asociadas a cada fase (parte derecha).

4. Variables en SiManFor

En SiManFor se ha diseñado un modelo de datos (SDM - SiManFor Data Model) que se almacena en una base de datos accesible on-line y gestionada con MS-SQL. La subida de inventarios se realiza a través de la plataforma web en la sección inventarios, mediante archivos de excel con un formato fijo y cuyo modelo se puede encontrar en la sección de ayuda de SiManFor. (Ver manual de usuario para obtener más detalles)

Estas variables poseen un modelo de datos similar al "SiManFor Data Model" (SDM), que es el formato en el que se "suben" los datos a la plataforma, pero distinto en cuanto a las entidades (variables) con las que puede trabajar. Esta versión del SDM es bastante más reducida, dado que para la ejecución de un modelo sólo se utilizan dos tablas (una con información a nivel de árbol y otra a nivel de parcela) en las que se agrupan varias de las tablas originales.

La mayor parte de los modelos son capaces de funcionar con una versión reducida del SDM, agrupadas en dos únicas tablas cuya estructura se indica a continuación. Los nombres deben respetarse en la programación, siguiendo las indicaciones adecuadas para referirse a ellas (Ver apartado 5.9). En la tabla 1 se muestran las variables de parcela,

Cuadro 1: **Variables de parcela**

$ID\_INVENTARIO$ identificador	$D\_DOMINANTE$ diámetro dominante en cm	Variables adicionales:
$ESTADILLO$ identificador de parcela	D_MAX diámetro máximo en cm	VAR_1
ID identificador	D_MIN diámetro mínimo en cm	VAR_2
$A\_BASIMETRICA$ en m^2/ha	H_MEDIA altura media en m	VAR_3
$H\_DOMINANTE$ en m	DM_COPA diámetro medio de copa en m	VAR_4
N_PIES en $arboles/parcela$	DG_COPA diámetro cuadrático medio de copa en m	VAR_5
N_PIESHA en $arboles/ha$	$I\_REINEKE$ índice de densidad de rodal	VAR_6
$EDAD$ en años	I_HART índice de Hart	VAR_7
D_MEDIO diámetro medio en cm	FCC fracción de cabida cubierta en %	VAR_8
$D\_CUADRATICO$ diámetro cuadrático medio en cm	SI índice de sitio en m	VAR_9

y en la tabla 2 las variables de árbol.

Cuadro 2: **Variables de árbol**

ID identificador	INCLINACION inclinación	ANCHO_CM_1 ancho de copa en m
ID_INVENTARIO identificador	REC rectitud del fuste	ANCHO_CM_2
ESTADILLO identificador de parcela	RAM ramosidad del fuste	RADIO_C_1 radio de copa en m
ID_PARCELA identificador de parcela interno	CONICIDAD conicidad del fuste	RADIO_C_2
ARBOL identificador de árbol	PUDRICION nivel de pudrición	RADIO_C_3
NUMEROINDIVIDUOS numero de arboles a los	CLAS_PIE clasificación madera en pie	RADIO_C_4
que representa el registro (para clases)	CLASE_SOCIOLOGICA clase sociológica	LCW máxima anchura de copa en m
ESPECIE según código IFN	VCC volumen en cm^3	C_MORFICO coeficiente mórfico
DIAMETRO_1 diámetro 1 en cm	VSC volumen sin corteza en cm^3	C_FORMA cociente de forma
DIAMETRO_2 diámetro 2 en cm	IAVC Incremento Anual de volumen con corteza	PERC_DURAMEN %duramen
CALIDAD Calidad	VLE volumen de leñas	
FORMA Forma de cubicación	BAL área basimétrica de los arboles más	
ALTURA altura total en m	gruesos que el individuo en m^2/ha	Variables adicionales:
PARAMESP parámetros especiales	CR índice de copa viva	VAR_1
OBSERVACIONES observaciones	SECCION_COPA_MAXIMA sección copa máx.	VAR_2
DAP valor medio de diámetro 1 y 2	EDAD_BASE edad del árbol en la base	VAR_3
CORTEZA_1 espesor en mm	DIAMETRO_MIN Diámetro mínimo en cm	VAR_4
CORTEZA_2 espesor en mm	DIAMETRO_4 Diámetro a 4 metros en cm	VAR_5
CORTEZA espesor medio en mm	FCV fracción de copa viva	VAR_6
CIRCUNFERENCIA en cm	ALTURA_BC altura a la base de la copa en m	VAR_7
EXPAN factor de expansión a la hectárea	ALTURA_MAC altura al máximo ancho de la copa en m	VAR_8
ESBELTEZ relación altura/diámetro (m/cm)	ALTURA_RM altura a la primera rama viva en m	VAR_9
SEC_NORMAL sección normal en m^2	ALTURA_TOC altura del tocón	VAR_10

En cada conjunto de datos se han incluido otras 10 variables, denominadas VAR_1, VAR_2, ... VAR_10. Estas pueden ser empleadas por los modelizadores en caso de que su modelo incluya variables no definidas en la base de datos. Estas variables no se pueden cambiar de nombre, pero se puede modificar el output para que el usuario final pueda identificarlas (se detallará cómo en el manual de uso de *plantillas de resultados*).

5. Programación de modelos de árbol individual

5.1. Estructura de la programación de cada proceso

Para incluir un modelo de árbol individual es aconsejable utilizar la plantilla alojada en la sección de ayuda de la plataforma. Además se puede descargar un ejemplo de modelo completo que puede ayudar a la programación de nuevos modelos. Es importante tener en cuenta que el flujo de ejecución de un modelo de árbol individual es como indica la figura 1. El orden de esta imagen es el que regula el funcionamiento del escenario, independientemente del orden en el que aparezca en la plantilla del modelo.

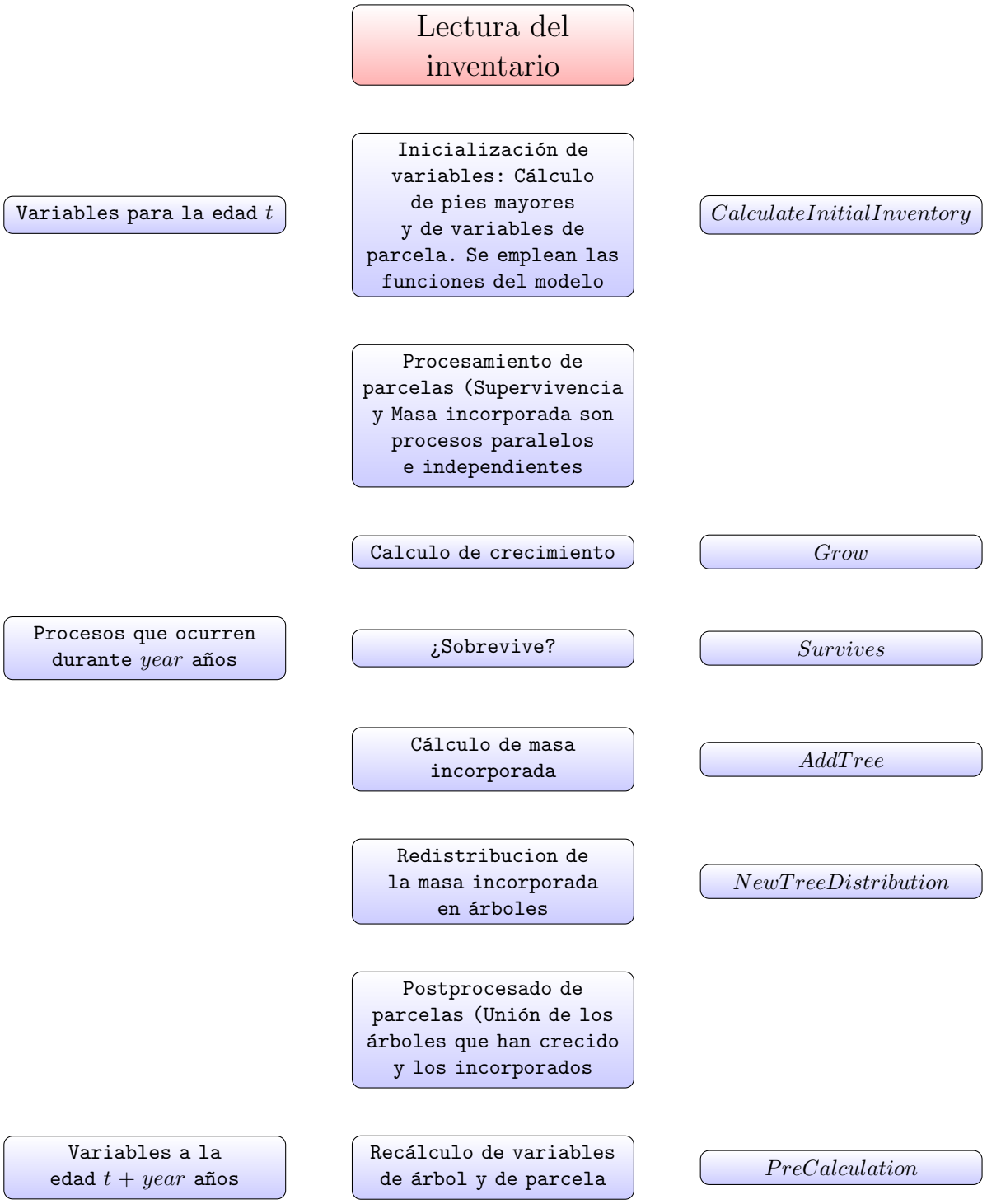


Figura 1: Diagrama de flujo de un modelo de árbol individual

Veamos las fases de cálculo que puede incluir un modelo de árbol individual en SiManFor:

5.1.1. Inicialización

Permite al modelo calcular las variables no disponibles en el propio inventario pero necesarias para la ejecución del modelo. Se ejecuta una vez por cada pie mayor incluido en el escenario.

A partir de la lectura del inventario hay una función en la que se calculan las variables de inicialización necesarias para la ejecución del modelo *CalculateInitialInventory*. Por ejemplo, se puede calcular la altura a la que la copa es más ancha o a la base de la copa, si se necesitan para el modelo, ya que son variables que no se miden habitualmente en los inventarios forestales. También es posible calcular para todos los individuos variables que, en algunos inventarios se miden únicamente para algunos pies, como la altura total. Para ello se pueden utilizar propiedades de la variable que indican si tiene almacenado algún valor o no.

```
/// <summary>
/// Procedimiento que permite la inicializacion de variables de parcelas necesarias
/// para la ejecucion del modelo. Solo se ejecuta en el primer nodo.
/// Variables que deben permanecer constantes como el indice de sitio -SI- deben calcularse
/// solo en este apartado del modelo
/// </summary>
/// <param name="plot"></param>
public override void CalculateInitialInventory(Parcela plot)
{
/// funcion que permite el acceso ordenado a la base de datos
/// (p.ej. para calcular el BAL mas rapidamente)
  IList<PieMayor> piesOrdenados = base.Sort(plot.PiesMayores, new PieMayorSortingCriteria.DescendingByField("DAP"));
    foreach (PieMayor tree in plot.PiesMayores)
      {
        /// se calculan las variables de arbol en este bucle
      }
    /// se calculan las variables de parcela en este apartado
 }
```

Según la declaración del encabezado es una función pública (public), en la que se sobreescribe el código por defecto (override) y no devuelve ningún valor (void). Todas las modificaciones que ejecuta la función se hacen sobre el inventario temporal.

5.1.2. Funciones de evolución de la masa

A continuación tenemos las funciones que desarrollan el modelo (*Grow*, *AddTree*, *NewTreeDistribution* y *Survives*). En la primera se calcula el crecimiento:

Crecimiento Modifica las propiedades (variables) del árbol nuevo después de la proyección temporal. Se ejecuta una vez por cada árbol vivo del escenario.

```
// <summary>
// Procedimiento que permite modificar las propiedades del arbol durante su crecimiento despu\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setb
// </summary>
// <param name="years"></param>
// <param name="plot"></param>
// <param name="oldTree"></param>
// <param name="newTree"></param>
public override void Grow(double years, Parcela plot, PieMayor oldTree, PieMayor newTree)
{
  // Ecuaciones que permiten programar como cambian las variable en \OT1\textquotedblleft years\OT1\textquotedblright  a\unhbox \voidb@x \bgroup
}
```

Supervivencia Finalmete tenemos una función para calcular la supervivencia, y permite calcular la probabilidad de supervivencia de un árbol transcurrido el un tiempo **years**. Se ejecuta una vez por cada árbol incluido en el escenario:

```
// <summary>
// Funcion que indica si el arbol sobrevive o no despu\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {e\global \mathcha
// </summary>
// <param name="years"></param>
// <param name="plot"></param>
// <param name="tree"></param>
// <returns>Devuelve el porcentaje de \unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {a\global \mathchardef \accent@spac
public override double Survives(double years, Parcela plot, PieMayor tree)
  {
    // Ecuaciones que calculan el \% de arboles que sobreviven tras \OT1\textquotedblleft years\OT1\textquotedblright  annos
    return survive;
  }
```

Esta función devuelve una variable real, mayor que 0 y menor o igual a 1. Con este valor SiManFor calculará el porcentaje de árboles que sobrevive. Cuando ese valor es menor de 1, se duplica el registro completo; al original se

le cambia el valor de la variable EXPAN, asignandole el valor EXPAN*survive, y al registro duplicado EXPAN*(1-survive) y a variable ESTADO el valor "M". Si la supervivencia es 1 no habrá cambios. Es **responsabilidad del modelizador** comprobar que el valor de la supervivencia está en el rango $(0, 1]$, ya que en caso contrario el programa producirá un error.

Masa Incorporada El procedimiento de cálculo de la masa incorporada se realiza con dos funciones. En la primera se calcula la inclusión de nuevos árboles al escenario al alcanzar un tamaño inventariable. Se ejecuta una sola vez.

La masa incorporada se calcula en la siguiente:

```
// <summary>
// Procedimiento que permite a\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {n\global \mathchardef \accent@spacefactor
// </summary>
// <param name="years"></param>
// <param name="plot"></param>
// <returns>Area basimetrica a distribuir o 0 si no hay masa incorporada</returns>
public override double? AddTree(double years, Parcela plot)
{
  double BasalAreaAdded=0.0F;
  // calculation of ''BasalAreaAdded''
  return BasalAreaAdded;
}
```

En la segunda función se utiliza como variable de entrada el resultado de la función anterior, **BasalAreaAdded**, el área basimétrica incorporada:

```
// <summary>
// Expresa como se ha de distribuir la masa incorporada entre los arboles existentes.
// La implementacion por defecto la distribuye de forma uniforme.
// </summary>
// <param name="years"></param>
// <param name="plot"></param>
// <param name="AreaBasimetricaIncorporada"></param>
// <returns></returns>
public override Distribution[] NewTreeDistribution(double years, Parcela plot, double AreaBasimetricaIncorporada)
{
// Hay que definir una matriz (distribution) que pertenece a la clase Distribution
// que tiene 3 propiedades: diametro menor (.diametroMenor), diametro mayor (.diametroMayor)
//  y \unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {a\global \mathchardef \accent@spacefactor \spacefactor }\accent 1
  return distribution;
}
```

Esta función devuelve una matriz, **distribution**, que usa la plataforma internamente para añadir árboles resultado del la masa incorporada, y que permite asignar a cada clase de diámetro el área basimétrica que se añadirá.

5.1.3. Posprocesado de parcelas

Permite al modelo el cálculo de variables derivadas similares a las que se calcularon en el apartado de inicialización, tras los cambios producidos en variables de árbol y parcela en las fases anteriores.

La parte final de un modelo de árbol individual es el recálculo de variables una vez que el árbol ha crecido, y teniendo en cuenta los incorporados y la mortalidad (ProcessPlot).

```
// <summary>
// Procedimiento que realiza los c\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {a\global \mathchardef \accent@spacefa
// </summary>
// <param name="years"></param>
// <param name="plot"></param>
// <param name="trees"></param>
public override void ProcessPlot(double years, Parcela plot, PieMayor[] trees)
{
// Instruccion que permite el acceso ordenado a la base de datos
  IList<PieMayor> piesOrdenados = base.Sort(plot.PiesMayores, new PieMayorSortingCriteria.DescendingByField("DAP"));

  foreach(PieMayor tree in piesOrdenados)
    {
      // se calculan las variables de arbol en este bucle
    }
  // se calculan las variables de parcela en este apartado
}
```

5.2. Ejemplo

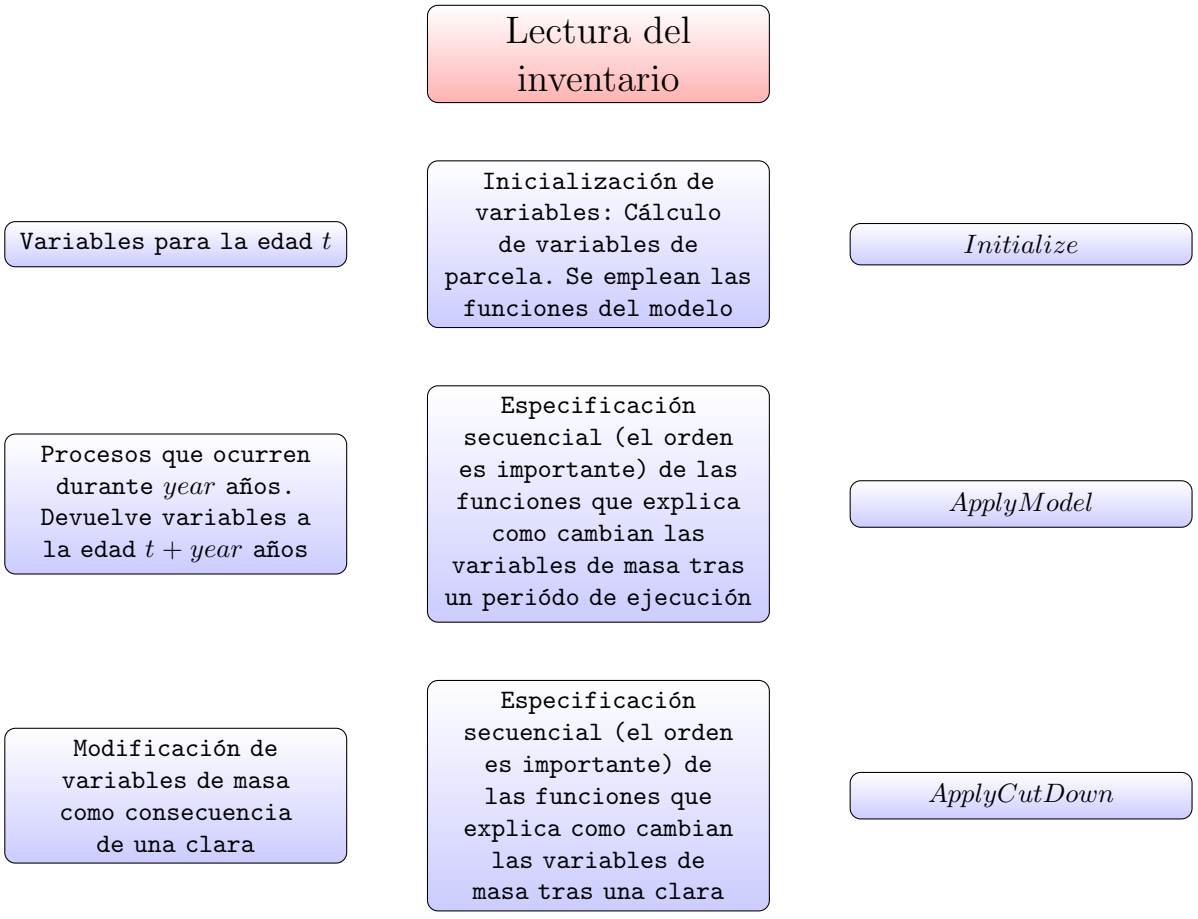
5.2.1. Descripción forestal

5.2.2. Programa por procesos

6. Programación de modelos de masa

6.1. Estructura de la programación de cada proceso

Para incluir un modelo de masa es aconsejable utilizar la plantilla alojada en la sección de ayuda de la plataforma. Además se puede descargar un ejemplo de modelo completo que puede ayudar a la programación de nuevos modelos. Es importante tener en cuenta que el flujo de ejecución de un modelo de masa es como indica la figura 2. El orden de esta imagen es el que regula el funcionamiento del escenario, independientemente del orden en el que aparezca en la plantilla del modelo.



Inicialización de variables En un primer momento es necesario calcular las variables de la masa a partir del inventario. También se calculará la calidad de estación o parámetros necesarios para su estimación. Esta función sólo se utiliza en el momento inicial del escenario.

Evolución de la masa Esta función permite calcular como evolucionan las variables de masa al transcurrir un periódo de tiempo. Es responsabilidad del modelizador realizar los cálculos necesarios para representar el estado de

la masa: densidad en pies por hectárea, área basimétrica y volumen total, así como variables medias de la parcela: diámetro medio cuadrático, altura dominante,...

Variación de la masa al aplicar una clara Esta función es adecuada para especificar la transformación de las variables de masa al realizar una clara. Es necesario realizar los cálculos para las tres formas típicas de especificar el peso de la clara: en porcentaje de número de pies, porcentaje de área basimétrica y porcentaje de volumen. Además SiManFor permite especificar tres tipos de clara: sistemática, claras bajas y claras altas, por lo que deben especificarse en la programación del modelo todas las ecuaciones que se hayan estimado a partir de ensayos de claras realizados bajo esos tipos.

6.2. Ejemplo

6.2.1. Descripción forestal

6.2.2. Programa por procesos

7. Cómo subir un programa a SiManFor

El procedimiento para crear modelos en sm es bastante sencillo, una vez tengamos el código almacenado en una archivo con extensión *cs*. Para ello será suficiente con acceder al menú *Modelos* y clicar en el botón *Nuevo*. A continuación aparece un formulario que nos pide dar nombre y descripción al modelo, y enlazar los archivos necesarios. El único imprescindible es un archivo con el código, en formato de texto plano (normalmente con extensión .cs), pero es posible incluir otros dos, uno con las especificaciones del modelo y otro con la *plantilla de resultados*. Una vez enlazados los archivos basta con clicar en *Crear modelo* y esperar el resultado. A veces hay algún error en la codificación, que será detallado en la web, pero si todo ha salido bien tendremos nuestro modelo en versión *En desarrollo*. Una vez comprobado que todo funciona correctamente podemos pasarlo a versión *En revisión* para que pueda ser testado por los modelizadores que deseemos.

Pasada esta última fase podremos solicitar a los desarrolladores que el modelo pase a estar en versión *Estable*. Para ello (imprescindible que el modelo tenga el archivo con las especificaciones del modelo, que serán de gran ayuda para los usuarios finales.

8. Anexos

8.1. Programa completo de un modelo de árbol individual

Se detalla a continuación el modelo de árbol individual Ibero Pt, modelo de árbol individual para *Pinus pinaster* para el Sistema Ibérico Meridional

```
using System;
using System.Collections.Generic;
using Simanfor.Core.EngineModels;
namespace EngineTest
{
  // <summary>
  // Todas las funciones y procedimientos son opcionales. Si se elimina cualquiera de ellas, se usara un
  // procedimiento o funcion por defecto que no modifica el estado del inventario.
  // Modelo IBERO, 2010, Pinus pinaster (Sistema Iberico)
  // </summary>
  public class Template : ModelBase
  {
    // declaracion de variables publicas
    public PieMayor currentTree;

    // Funciones de perfil utilizadas en el c\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {a\global \mathchardef \accen
    public double r2_conCorteza(double HR)
    {
      double r=(1+1.1034*Math.Exp(-6.0879*HR))*0.5656*currentTree.DAP.Value/200*Math.Pow((1-HR),(0.6330-1.7228*(1-HR)));
```

```
  return Math.Pow(r,2);
}
public double r2_sinCorteza(double HR)
{
  double r=(1+2.4771*Math.Exp(-5.0779*HR))*0.2360*currentTree.DAP.Value/200*Math.Pow(1-HR,0.4733-3.0371*(1-HR));
  return Math.Pow(r,2);
}
// <summary>
// Procedimiento que permite la inicializacion de variables de parcelas necesarias para la ejecucion del modelo
// Solo se ejecuta en el primer nodo.
// Variables que deben permanecer constantes como el indice de sitio deben calcularse solo en este apartado del modelo
// </summary>
// <param name="plot"></param>
public override void CalculateInitialInventory(Parcela plot)
  {
  foreach (PieMayor tree in plot.PiesMayores)
    {
      tree.BAL = 0;
      foreach (PieMayor pm in tree.Parcela.PiesMayores)
        {
          if (pm.DAP>tree.DAP)
          {
            tree.BAL+=pm.SEC_NORMAL.Value*pm.EXPAN.Value/10000;
          }
        }
          if (!tree.ALTURA.HasValue)
          {
                  tree.ALTURA=(13+(32.3287+1.6688*plot.H_DOMINANTE*10-0.1279*plot.D_CUADRATICO*10)*Math.Exp(-11.4522/Math.Sqrt(tree.DAP.Valu
          }
          if (!tree.ALTURA_MAC.HasValue)
          {
            tree.ALTURA_MAC=tree.ALTURA.Value/(1+Math.Exp((double)(-0.0041*tree.ALTURA.Value*10-0.0093*tree.BAL-0.0123*plot.A_BASIMETRICA)));
          }
          if (!tree.ALTURA_BC.HasValue)
          {
                  tree.ALTURA_BC=tree.ALTURA_MAC.Value/
                  (1+Math.Exp((double)(0.0078*plot.A_BASIMETRICA-0.5488*Math.Log(plot.A_BASIMETRICA.Value)-0.0085*tree.BAL)));
          }
          tree.CR=1-tree.ALTURA_BC.Value/tree.ALTURA.Value;
          if (!tree.LCW.HasValue)
          {
                  tree.LCW=(1/10.0F)*(0.1826*tree.DAP.Value*10)*Math.Pow(tree.CR.Value,(0.1594+0.0014*(tree.ALTURA.Value-tree.ALTURA_BC.Valu
          }
          currentTree = tree;
          tree.VCC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_conCorteza); //Integracion --> r2_conCorteza sobre HR en los limit
          tree.VSC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_sinCorteza); //Integracion --> r2_sinCorteza sobre HR en los limit
          currentTree=null;
        }
        plot.SI=Math.Exp(4.016+(Math.Log(plot.H_DOMINANTE.Value)-4.016)*Math.Pow(80/plot.EDAD.Value,-0.5031));
    }
    // <summary>
    // Procedimiento que permite la inicializacion de variables de parcelas necesarias para la ejecucion del modelo
    // </summary>
    // <param name="plot"></param>
    public override void Initialize(Parcela plot)
    {
        plot.SI=Math.Exp(4.016+(Math.Log(plot.H_DOMINANTE.Value)-4.016)*Math.Pow(80/plot.EDAD.Value,-0.5031));
    }
    // <summary>
    // Procedimiento que permite la inicializacion de variables de arbol necesarias para la ejecucion del modelo
    // </summary>
    // <param name="plot"></param>
    // <param name="tree"></param>
    public override void InitializeTree(Parcela plot, PieMayor tree)
    {
      if (!tree.ESTADO.HasValue || String.IsNullOrEmpty(tree.ESTADO.ToString()))
      {
        tree.BAL = 0;
        foreach (PieMayor pm in tree.Parcela.PiesMayores)
        {
          if (!pm.ESTADO.HasValue || String.IsNullOrEmpty(pm.ESTADO.ToString()))
            {
            if (pm.DAP > tree.DAP)
              {
                tree.BAL+=pm.SEC_NORMAL.Value*pm.EXPAN.Value/10000;
              }
            }
          }
        }

        if (!tree.ALTURA.HasValue)
        {
```

```
      tree.ALTURA=(13+(32.3287+1.6688*plot.H_DOMINANTE*10-0.1279*plot.D_CUADRATICO*10)*Math.Exp(-11.4522/Math.Sqrt(tree.DAP.Value*10)))/1
    }
    tree.ALTURA_MAC=tree.ALTURA.Value/(1+Math.Exp((double)(-0.0041*tree.ALTURA.Value*10-0.0093*tree.BAL-0.0123*plot.A_BASIMETRICA)));
    tree.ALTURA_BC=tree.ALTURA_MAC.Value/(1+Math.Exp((double)(0.0078*plot.A_BASIMETRICA-0.5488*Math.Log(plot.A_BASIMETRICA.Value)-0.0085*
    tree.CR=1-tree.ALTURA_BC.Value/tree.ALTURA.Value;
    tree.LCW=(1/10.0F)*(0.1826*tree.DAP.Value*10)*Math.Pow(tree.CR.Value,(0.1594+0.0014*(tree.ALTURA.Value-tree.ALTURA_BC.Value)*10));
}
/// <summary>
/// Funcion que indica si el arbol sobrevive o no despu\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {e\global \
/// </summary>
/// <param name="years"></param>
/// <param name="plot"></param>
/// <param name="tree"></param>
/// <returns>Devuelve el porcentaje de arboles que sobreviven</returns>
public override double Survives(double years, Parcela plot, PieMayor tree)
{
double BA_Survives =1-(1/(1+Math.Exp(2.0968+(4.7358*tree.DAP.Value/plot.D_CUADRATICO.Value)-0.0012*plot.SI.Value*plot.A_BASIMETRICA.Value
  if (BA_Survives>0)
    {
      return BA_Survives;
    }
    return 0.0F;
}
/// <summary>
/// Procedimiento que permite modificar las propiedades del arbol durante su crecimiento despu\unhbox \voidb@x \bgroup \let \unhbox \voidb
/// </summary>
/// <param name="years"></param>
/// <param name="plot"></param>
/// <param name="oldTree"></param>
/// <param name="newTree"></param>
public override void Grow(double years, Parcela plot, PieMayor oldTree, PieMayor newTree)
{
  double DBHG5=Math.Exp(0.2030*Math.Log(oldTree.DAP.Value*10) + 0.4414*Math.Log((oldTree.CR.Value+0.2)/1.2) + 0.8379*Math.Log(plot.SI.Val
  newTree.DAP+=DBHG5/10;
  ///    HTG5=(1/100)*Exp(0.21603+0.40329*Log(DBHG5*10/2)-1.12721*Log(DBH*10)                  +1.18099*Log(HT*100)
  double HTG5=Math.Exp(0.21603+0.40329*Math.Log(DBHG5/2)-1.12721*Math.Log(oldTree.DAP.Value*10)+1.18099*Math.Log(oldTree.ALTURA.Value*100
  newTree.ALTURA+=HTG5/100;
  /// formula de la tesis--corregido error en base de datos    newTree.ALTURA+=(Math.Exp(4.1375+0.3762*Math.Log(DBHG5*10/2)-0.5260*Math.Log
}
/// <summary>
/// Procedimiento que permite a\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {n\global \mathchardef \accent@space
/// </summary>
/// <param name="years"></param>
/// <param name="plot"></param>
/// <returns>Area basimetrica a distribuir o 0 si no hay masa incorporada</returns>
public override double? AddTree(double years, Parcela plot)
{
  double result=1/(1+Math.Exp(12.3424+0.1108*plot.A_BASIMETRICA.Value-0.6154*plot.D_CUADRATICO.Value));
  if (result>=0.38F)
  {
    double BA_Added=6.7389-0.2235*plot.D_CUADRATICO.Value;
    if (BA_Added<0)
      {
            return 0.0F;
      }
      return BA_Added;
    }
    return 0.0F;
}
/// <summary>
/// Expresa como se ha de distribuir la masa incorporada entre los arboles existentes.
/// La implementacion por defecto la distribuye de forma uniforme.
/// </summary>
/// <param name="years"></param>
/// <param name="plot"></param>
/// <param name="AreaBasimetricaIncorporada"></param>
/// <returns></returns>
public override Distribution[] NewTreeDistribution(double years, Parcela plot, double AreaBasimetricaIncorporada)
{
    Distribution[] distribution = new Distribution[3];
    double percentAreaBasimetrica = AreaBasimetricaIncorporada / plot.A_BASIMETRICA.Value;
    distribution[0] = new Distribution();
    distribution[0].diametroMenor = 0.0;
    distribution[0].diametroMayor = 12.5;
    distribution[0].AreaBasimetricaToAdd = 0.0809 * AreaBasimetricaIncorporada;
    distribution[1] = new Distribution();
    distribution[1].diametroMenor = 12.5;
    distribution[1].diametroMayor = 22.5;
    distribution[1].AreaBasimetricaToAdd = 0.3263 * AreaBasimetricaIncorporada;
    distribution[2] = new Distribution();
    distribution[2].diametroMenor = 22.5;
    distribution[2].diametroMayor = double.MaxValue;
```

```
            distribution[2].AreaBasimetricaToAdd = 0.5828 * AreaBasimetricaIncorporada;
            return distribution;
        }
        /// <summary>
        /// Procedimiento que realiza todos los precalculos para preparar el procesamiento de los arboles y parcelas.
        /// </summary>
        /// <param name="years"></param>
        /// <param name="plot"></param>
        /// <param name="trees"></param>
        public override void PreCalculation(double years, Parcela plot, PieMayor[] trees)
        {
        }
        /// <summary>
        /// Procedimiento que realiza los calculos sobre un arbol.
        /// </summary>
        /// <param name="years"></param>
        /// <param name="plot"></param>
        /// <param name="tree"></param>
        public override void ProcessTree(double years, Parcela plot, PieMayor tree)
        {
            currentTree = tree;
            tree.VCC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_conCorteza); //Integracion --> r2_conCorteza sobre HR en los limites
            tree.VSC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_sinCorteza); //Integracion --> d_sinCorteza sobre HR en los limites
            currentTree = null;
                    //double exp = 1/0.8533;
                    //double DIB = Math.Pow((1/2.8708),exp)*Math.Pow(tree.DAP.Value*10,exp);
                    //currentTree = tree;
                //tree.VCC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_conCorteza); //Integracion --> r2_conCorteza sobre HR en los li
                    //tree.VSC=Math.PI*tree.ALTURA.Value*IntegralBySimpson(0,1,0.01,r2_sinCorteza); //Integracion --> r2_sinCorteza sobre HR en lo
                    //currentTree = null;
        }

        /// <summary>
        /// Procedimiento que realiza los calculos sobre una parcela.
        /// </summary>
        /// <param name="years"></param>
        /// <param name="plot"></param>
        /// <param name="trees"></param>
        public override void ProcessPlot(double years, Parcela plot, PieMayor[] trees)
        {
        }
    }
}
```

8.2. Programa completo de un modelo de masa

Se detalla a continuación el modelo de masa de Silves, modelo de árbol individual para *Pinus sylvestris* para el Sistema Ibérico Meridional.

```
using System;
using System.Collections.Generic;
using Simanfor.Core.EngineModels;
using Simanfor.Entities.Enumerations;

// Parte practica del curso de Simanfor como Herramienta Docente (Septiembre 2011)
// ------------------------------------------------------------------------------------
// Modelo para masas de pino silvestre basado en los articulos:
// * del Rio Gaztelurrutia, M; Montero, G.; -Modelo de simulacion de claras en masas de
// Pinus sylvestris L.- monografias inia: Forestal n. 3
// ------------------------------------------------------------------------------------

namespace EngineTest
{
  public class MassModelTemplate : MassModelBase
  {
    public override void Initialize(Parcela plot)
    {
      //Indice de sitio
      double parA, parB, parC, IC;
      parA        = 0.8534446F;
      parB        = -0.27F;
      parC        = 0.439F;
      plot.SI   = parA * plot.H_DOMINANTE.Value / Math.Pow( 1- Math.Exp((double) parB * plot.EDAD.Value/10F) , 1/parC);
      IC        = plot.SI.Value/10F ;
      plot.VAR_1= IC; // almacenamos el indice de calidad en la variable extra VAR_1
      // Volumen inicial
      double parB0, parB1,parB2, parB3;
      parB0        = 1.42706D;
```

```
      parB1        = 0.388317D;
      parB2        = -30.691629D;
      parB3        = 1.034549D;
      plot.VCC  = Math.Exp((double) parB0 + parB1 * IC + parB2 / plot.EDAD.Value + parB3 * Math.Log(plot.A_BASIMETRICA.Value) );
    }


    public override void ApplyModel(Parcela oldPlot, Parcela newPlot, int years)
    {
      newPlot.SI         = oldPlot.SI.Value;
      newPlot.EDAD         = oldPlot.EDAD.Value + years;
      newPlot.VAR_1         = oldPlot.VAR_1.Value;
      // H_DOMINANTE:
      double IC        = oldPlot.SI.Value/10;
      double parA17, parB17, parC17, parA29, parB29, parC29;
      parA17        = 1.9962;
      parB17        = 0.2642;
      parC17        = 0.46;
      parA29        = 3.1827;
      parB29        = 0.3431;
      parC29        = 0.3536;
      double H0_17        = 10 * parA17 * Math.Pow( 1 - Math.Exp((double) -1 * parB17 * newPlot.EDAD.Value / 10 ) , 1/parC17);
      double H0_29        = 10 * parA29 * Math.Pow( 1 - Math.Exp((double) -1 * parB29 * newPlot.EDAD.Value / 10 ) , 1/parC29);
      newPlot.VAR_2        = H0_17;
      newPlot.VAR_3        = H0_29;
      newPlot.H_DOMINANTE        = H0_17 + (H0_29 - H0_17) * (IC - 1.7) / 1.2;
      // AREA_BASIMETRICA:
      double parA0, parA1, parB0, parB1, parA2, parB2, parB3;
      parA0        = 5.103222;
      parB0        = 1.42706;
      parB1        = 0.388317;
      parB2        = -30.691629;
      parB3        = 1.034549;
      newPlot.A_BASIMETRICA        = Math.Pow( oldPlot.A_BASIMETRICA.Value, oldPlot.EDAD.Value / newPlot.EDAD.Value) * Math.Exp(parA0 * (1-oldPlo
      //MORTALIDAD NATURAL
      parA0        = -2.34935;
      parA1        = 0.000000099;
      parA2        = 4.87390;
      newPlot.N_PIESHA        = Math.Pow( Math.Pow(oldPlot.N_PIESHA.Value,parA0) + parA1 * ( Math.Pow( newPlot.EDAD.Value/100 , parA2 ) - Math.Po
      // VOLUMEN
      newPlot.VCC = Math.Exp((double) parB0 + parB1 * IC + parB2 / newPlot.EDAD.Value + parB3 * Math.Log(newPlot.A_BASIMETRICA.Value) );
      newPlot.VAR_10 = Math.Exp((double) parB0 + parB1 * IC + parB2 / newPlot.EDAD.Value  + parB3 * ( Math.Log(oldPlot.A_BASIMETRICA.Value)*oldPl
      //Altura media
      parA0        = -1.155649;
      parA1        = 0.976772;
      newPlot.H_MEDIA        = parA0 + parA1 * newPlot.H_DOMINANTE;
      // D_CUADRATICO:
      double SEC_NORMAL        = newPlot.A_BASIMETRICA.Value * 10000 / newPlot.N_PIESHA.Value    ;
      newPlot.D_CUADRATICO     = 2*Math.Sqrt(SEC_NORMAL/Math.PI)    ;
      //newPlot.D_CUADRATICO        = parA * Math.Pow((double)newPlot.N_PIESHA.Value, parB) * Math.Pow((double)newPlot.H_DOMINANTE.Value, parC);
      // Para que sea de tipo double, guardamos el D_CUADRATICO en VAR_1
      // Nota: usamos PieMayor.EXPAN en lugar de oldTree.EXPAN
      //                newTree.VAR_1 = 43.791 * Math.Pow((double)plot.N_PIESHA.Value, -0.270) * Math.Pow((double)newTree.ALTURA.Value, 0.426);
      //                newTree.DAP = newTree.VAR_1;
      newPlot.VAR_8                = newPlot.N_PIESHA.Value * Math.Pow( 25 / (double) newPlot.D_CUADRATICO.Value, -1.75);
      newPlot.I_REINEKE         = newPlot.VAR_8;
      // Para comprobarlo: SALE DISTINTO, NO PODEMOS SOBREESCRIBIR I_REINEKE EN Parcela
      // newTree.VAR_3 = oldTree.EXPAN.Value * Math.Pow(25 / (double)newTree.VAR_1.Value, -1.75);
    }

  // cutDownType values: ( PercentOfTrees, Volume, Area )
  // trimType values: ( ByTallest, BySmallest, Systematic )
  // value: (% de corta)
    public override void ApplyCutDown(Parcela oldPlot, Parcela newPlot, CutDownType cutDownType, TrimType trimType, float value)
    {
      newPlot.VAR_9 = value;
      //Indice de calidad
      double IC = oldPlot.SI.Value/10;
      // H_DOMINANTE:
      double parA17, parB17, parC17, parA29, parB29, parC29;
      parA17        = 1.9962;
      parB17        = 0.2642;
      parC17        = 0.46;
      parA29        = 3.1827;
      parB29        = 0.3431;
      parC29        = 0.3536;
      double H0_17        = 10 * parA17 * Math.Pow( 1 - Math.Exp((double) -1 * parB17 * newPlot.EDAD.Value / 10 ) , 1/parC17);
      double H0_29        = 10 * parA29 * Math.Pow( 1 - Math.Exp((double) -1 * parB29 * newPlot.EDAD.Value / 10 ) , 1/parC29);
      newPlot.VAR_2        = H0_17;
      newPlot.VAR_3        = H0_29;
      newPlot.H_DOMINANTE        = H0_17 + (H0_29 - H0_17) * (IC - 1.7) / 1.2;
      // parametros de volumen y area basim\unhbox \voidb@x \bgroup \let \unhbox \voidb@x \setbox \@tempboxa \hbox {e\global \mathchardef \accent@s
```

```
      double parA0, parB0, parB1, parB2, parB3;
      parA0        = 5.103222;
      parB0        = 1.42706;
      parB1        = 0.388317;
      parB2        = -30.691629;
      parB3        = 1.034549;
      // parametros de corta
      double parC0,parC1,parC2,SEC_NORMAL,tpuN,tpuBA;
      switch (cutDownType)
      {
      case CutDownType.PercentOfTrees:
        parC0        = 0.531019;
        parC1        = 0.989792;
        parC2        = 0.517850;
        tpuN        = value/100;
        newPlot.N_PIESHA        = (1 - tpuN)*oldPlot.N_PIESHA.Value;
        newPlot.D_CUADRATICO          = parC0 + parC1*oldPlot.D_CUADRATICO + parC2*oldPlot.D_CUADRATICO.Value*Math.Pow(tpuN,2);
        SEC_NORMAL                = Math.PI * Math.Pow(newPlot.D_CUADRATICO.Value/2,2);
        newPlot.A_BASIMETRICA   = SEC_NORMAL * newPlot.N_PIESHA.Value / 10000;
        newPlot.VCC = Math.Exp((double) parB0 + parB1 * IC + parB2 / newPlot.EDAD.Value + parB3 * Math.Log(newPlot.A_BASIMETRICA.Value) );
        break;
      case CutDownType.Volume:
        break;
      case CutDownType.Area:
        parC0        = 0.144915;
        parC1        = 0.969819;
        parC2        = 0.678010;
        tpuBA        = value/100;
        newPlot.A_BASIMETRICA        = (1- tpuBA)*oldPlot.A_BASIMETRICA.Value;
        newPlot.D_CUADRATICO        = Math.Pow(parC0 + parC1*Math.Pow(oldPlot.D_CUADRATICO.Value,0.5) + parC2*(tpuBA) ,2 );
        SEC_NORMAL                = Math.PI * Math.Pow(newPlot.D_CUADRATICO.Value/200,2);
        newPlot.N_PIESHA        = newPlot.A_BASIMETRICA.Value / SEC_NORMAL;
        newPlot.VCC = Math.Exp((double) parB0 + parB1 * IC + parB2 / newPlot.EDAD.Value + parB3 * Math.Log(newPlot.A_BASIMETRICA.Value) );
        break;
      }
    }
}
}
```