کتاب الکترونیک – ویرایش (۱) ۲۰۲۰۰۵۲۸

آموزش مقدماتی

مدلسازی سیستم‌های مهندسی با استفاده از زبان

Modelica

- **بخش اول – نرم افزار SystemModeler**
- **بخش دوم – نرم افزار OpenModelica**
- **بخش سوم – زبان Modelica**

عبدالحمید انصاری نسب مینابی

به نام خدا

کتاب الکترونیکی

آموزش مقدماتی

مدلسازی سیستم‌های مهندسی

با استفاده از زبان

Modelica

عبدالحمید انصاری نسب مینابی

همکاران تهیه کتاب:

نسخه الکترونیکی:

- عبدالحمید انصاری نسب مینابی

ویراست‌نهایی و صفحه‌بندی نسخه کاغذی:

- عبدالحمید انصاری نسب

طرح روی جلد:

- فاطمه دیودار

بخش اول - نرم افزار SystemModeler:

- ترجمه اولیه نرم‌افزار MathModelica: میلاد شفیعی
- ترجمه و ویراست اولیه: عبدالحمید انصاری نسب مینابی
- ویراست مجدد و ترجمه بخشهای اضافه شده پس از تغییر نرم افزار به SystemModeler: عبدالحمید انصاری نسب
- تألیف بخش ۲-۳ "معادلات دیفرانسیل": عبدالحمید انصاری نسب مینابی
- ترجمه و تألیف بخش ۴-۱ "مدلسازی علی و غیر علی": فرهاد محمودی

بخش دوم - نرم افزار OpenModelica:

- ترجمه اولیه: سیده صدیقه موسوی
- ترجمه و ویراست اولیه: عباس حریفی
- ویراست نهایی: عبدالحمید انصاری نسب مینابی
- تألیف بخش ۱۰-۵ "ایجاد مدل پیشرفته مخزن": عبدالحمید انصاری نسب مینابی
- ترجمه و ویراست فصل ۱۳ ویرایشگر ارتباطیOpenModelica: آیدین پناهی

بخش سوم - زبان Modelica:

- ترجمه: مهرزاد انصاری پور
- ویراست نهایی: عبدالحمید انصاری نسب مینابی

برگ مشخصات نسخه کاغذی کتاب:

سرشناسه	آموزش مقدماتی مدلسازی سیستم‌های مهندسی با استفاده از زبان Modelica/ هیئت مترجمین و مولفین عبدالحمید انصاری نسب مینابی... [و دیگران]
مشخصات نشر	تهران: دانشگاه هرمزگان ۱۳۹۳
مشخصات ظاهری	۱۹۴ ص. : مصور، ۱۷×۲۴ س م
شابک	۹۷۸-۶۰۰-۷۲۷۹-۲۶-۷
یادداشت	هیئت مولفین و مترجمین عبدالحمید انصاری نسب مینابی، عباس حریفی، فرهاد محمودی، آیدین پناهی، مهرزاد انصاری‌پور، میلاد شفیعی، سیده صدیقه موسوی
یادداشت	کتابنامه
موضوع	شی گرا (کامپیوتر)
	نرم افزار مودلیکا
	کامپیوتر -- شبیه سازی
	مهندسی -- برنامه های کامپیوتری
	مدل و مدلسازی -- نرم افزار
شناسه افزوده	انصاری نسب مینابی، عبدالحمید، ۱۳۵۹ -
شناسه افزوده	دانشگاه هرمزگان
رده بندی کنگره	۱۳۹۳ ۸آ۹ش/QA۷۶
رده بندی دیویی	۰۰۵/۱۱۷
شماره کتابشناسی ملی	۳۶۱۸۶۶۲۰

نام کتاب	آموزش مقدماتی مدلسازی سیستم‌های مهندسی با استفاده از زبان Modelica
ناشر	دانشگاه هرمزگان
مولفین	عبدالحمید انصاری نسب مینابی، عباس حریفی، فرهاد محمودی، آیدین پناهی، مهرزاد انصاری‌پور، میلاد شفیعی، سیده صدیقه موسوی
طراح جلد	فاطمه دیودار
چاپ و صحافی	افرنگ
نوبت چاپ	اول / ۱۳۹۳
تیراژ	۱۵۰۰ نسخه
قیمت به همراه CD	۱۰۰۰۰۰ ریال
شابک: ۹۷۸-۶۰۰-۷۲۷۹-۲۶-۷	ISBN: 978-600-7279-26-7

پیشگفتار نسخه الکترونیک

با سلام و درود

کتاب " آموزش مقدماتی مدلسازی سیستم‌های مهندسی با استفاده از زبان Modelica"، اولین کتاب آموزش زبان مودلیکا به زبان فارسی است که در سال ۹۳ به چاپ رسید. هدفمان، ایجاد یک کتاب خودخوان کوچک برای دانشجویان و محققان ایرانی بود تا بتوانند خیلی سریع با امکانات مودلیکا آشنا شده و از آن به عنوان یک ابزار قدرتمند در مدلسازی سیستم‌های دینامیک بهره‌مند شوند. از آن روز تا کنون، دوستانی با ارسال ایمیل مرا مورد محبت قرار داده‌اند. متاسفانه در ماه‌های اخیر، ایمیل‌هایی دریافت کرده‌ام مبنی بر نایاب شدن نسخه کاغذی این کتاب در داخل و عدم امکان دسترسی به آن در خارج از کشور، بنابراین بر آن شدم تا کتاب را به صورت نسخه الکترونیکی انتشار دهم و معتقدم که محتوای و مطالب کتاب به خوبی روز اول قابل استفاده است و مطالعه آن را به تمامی محققین، دانشمندان و دانشجویانی که با مدلسازی سیستمهای دینامیک سروکار دارند، پیشنهاد میکنم.

هدف از انتشار الکترونیک این کتاب، ایجاد دسترسی آسان به کتاب برای تمامی علاقه‌مندان و ایجاد بستری برای تصحیح، توسعه و به روز رسانی بهتر کتاب و اضافه کردن مطالب جدید به آن است.

از تمامی خوانندگان عزیز تقاضا میکنم با پشتیبانی از کتاب، ما را در رفع اشکالات کتاب و بهسازی آن و تهیه ویرایش های بعدی یاری رسانند.

لطفاً، نظرات و پیشنهادهای خود را در خصوص مطالب کتاب و مشکلات آن، به گروه واتسپ مودلیکا ارسال نمایید یا از طریق ایمیل Ansari.nasab@gmial.com آنها را برایم ارسال کنید.

در صورتی که مطالب کتاب را مفید یافتید و همچنین توان حمایت از کتاب را دارید، معادل دو یورو را به شماره کارت ۶۰۳۷۶۹۷۴۹۷۹۳۶۲۸۳ بانک صادرات به نام عبدالحمید انصاری نسب مینابی کارت به کارت نمایید. (هشدار: مبلغ واریز شده، حکم کمک بلا عوض را دارد و به هیچ عنوان قابل برگشت، تغییر یا پیگیری نیست و نخواهد بود.)

عبدالحمید انصاری نسب مینابی

خرداد ماه ۱۳۹۹

تمامی حقوق مادی و معنوی این اثر متعلق به عبدالحمید انصاری نسب مینابی می‌باشد.
استفاده از مطالب این کتاب با ذکر منبع، بلامانع است.
فروش این کتاب یا بخشی از آن به هر صورت بدون مجوز کتبی ممنوع می‌باشد.

پیشگفتار

تکنولوژی در همه زمینه‌ها با سرعت زیادی در حال پیشرفت است. هر روز شاهد ساخت ابزارهای قدرتمندتر، روشهای بهتر و امکانات بیشتر هستیم. ابزارهایی که خودشان راه را برای کشف روش‌ها و ابزارهای بهتر هموار می‌کنند. ما در عصر خلق تکنولوژی به کمک تکنولوژی، زندگی می‌کنیم. فلسفه طراحی و محاسبات در دهه‌های اخیر دستخوش تحول زیادی شده است و امروزه برای طراحی یک سیستم لازم نیست نمونه‌های آزمایشگاهی زیادی ساخته شود و برای یافتن بهترین طرح، محاسبات دستی زیادی انجام گیرد؛ همین امر باعث افزایش سرعت طراحی، افزایش کیفیت محصولات و کاهش فوق‌العاده هزینه‌ها گردیده است. این تحول مدیون امکانات نرم‌افزاری ایجاد شده در شبیه‌سازی سیستم‌های مختلف می‌باشد. شبیه‌سازی نرم‌افزاری امکان طراحی سیستمها در حالت بهینه با توجه به شرایط کاری و عمر تجهیز، هزینه ساخت، خط تولید و محاسبات اقتصادی را فراهم نموده است. در دنیای تکنولوژیک امروز برای داشتن کیفیت بالا و قیمت پایین، سرعت و دقت بالا در طراحی و ساخت سیستمهای جدید یا توسعه و بهینه‌سازی سیستمهای موجود قابل رقابت با بازارهای جهانی، ناگزیر باید از قدرتی که نرم‌افزار و سخت‌افزارهای امروزی در اختیار ما قرار داده‌اند، استفاده کرد.

نرم‌افزارهای زیادی برای شبیه‌سازی سیستمهای مختلف مهندسی خلق شده‌اند. نرم‌افزارهایی که مدارات الکترونیک را شبیه‌سازی می‌کنند، محاسبات مربوط به حرکت سیال را انجام می‌دهند یا بارگذاریهای مختلف را در اجزاء ماشینها محاسبه می‌نمایند، سیستمهای انرژی را بهینه می‌نمایند و ... . فهرست این نرم‌افزارها بسیار طولانی است و هر یک قدرت خود را در جنبه‌ای از مهندسی به نمایش می‌گذارند. اساس تمام این نرم‌افزارها، حل معادلات مهندسی سیستمهای مختلف است. این معادلات رفتار قطعات و مواد دنیای واقعی را شبیه‌سازی می‌نمایند. بدیهی است که با شناخت بهتر دنیای اطرافمان هر روز معادلات بهتری کشف می‌گردد که بهتر از معادلات گذشته می‌توانند رفتارها را مدلسازی نمایند و به تبع آنها نرم‌افزارهای بهتری خلق می‌شوند که نتایج واقعی‌تری ارائه می نمایند.

مهندسان، معادلات حاکم بر سیستمهای مربوط به خودشان را به خوبی می‌شناسند، اما برای اغلب آنها پیاده‌سازی و حل این معادلات در کامپیوتر حتی برای سیستمهای کوچک مهندسی کاری زمان‌بر، پر خطا و طاقت‌فرسا است. اغلب مهندسان حتی آنهایی که با زبانهای برنامه‌نویسی و سخت-افزار آشنایی کافی دارند، ترجیح می‌دهند بر سیستمهای مربوط به خودشان متمرکز شوند و بدون درگیر شدن با جزئیات نرم‌افزاری و سخت‌افزاری به شبیه‌سازی آنها بپردازند. وجود نرم‌افزاری که فکر ما را از جزئیات برنامه‌نویسی و حل معادلات رها کند و همچنین کتابخانه بزرگی از مدل قطعات آماده داشته باشد، امکان استفاده از قطعات سیستمهای مختلف را در کنار یکدیگر به راحتی فراهم کند و خلق کردن تجهیزات جدید در آن به سادگی امکانپذیر باشد، آرزوی بزرگی است. برآورده شدن این

آرزو امکان تمرکز مهندسان بر طراحی بهینه را فراهم خواهد نمود. زبان مدلسازی Modelica این آرزو را برآورده می‌سازد.

زبان Modelica برای مدلسازی سیستمهای مهندسی خلق شده است. در این زبان امکان تمرکز بر مدلسازی سیستم واقعی بدون درگیر شدن با جزئیات نرم‌افزاری و سخت‌افزاری مهیا می‌باشد. این زبان با ارائه مفهوم مدل که در واقع می‌تواند هر بخش کوچکی از سیستم شما باشد امکان خلق، شبیه‌سازی و نگهداری سیستم‌های بزرگ را به سادگی برای شما فراهم نموده است. برای مثال یک سیستم آب‌رسانی بزرگ را در نظر بگیرید که شامل انواع لوله‌ها، شیرها، منابع و مصرف‌کننده‌ها است. اگر قرار بود این سیستم را با استفاده از زبانهای برنامه‌نویسی معمول شبیه‌سازی نمایید، لازم بود که معادلات مربوط به هر یک از تجهیزات این سیستم را بنویسید و برای حل کردن همه آنها به صورت همزمان حل کننده مناسب خلق کنید! این کار برای سیستم‌های بزرگ کار بسیار مشکلی است. راه حل ساده‌تر، استفاده از یک نرم‌افزار شبیه‌ساز سیستم‌های آب‌رسانی است، اما فرض کنید می‌خواهید در همین تحلیل فرسایش لوله‌ها را نیز مدل کنید، تا جایی که من اطلاع دارم چنین نرم‌افزاری تا کنون نوشته نشده است. پس چه باید کرد؟ می‌توانید از امکانات زبان Modelica استفاده نمایید. برای شبیه‌سازی این سیستم آب‌رسانی فقط لازم است که یک مدل برای هر نوع قطعه مثلاً لوله، شیر، منبع و مصرف‌کننده ایجاد نمایید یعنی فقط چهار مدل بسازید، برای ساخت مدل هم کافی است که معادلات مربوط به هر قطعه را داشته باشید و آنها را در نرم‌افزار وارد کنید. حتی مدلسازی فرسایش نیز کار ساده‌ای است، کافی است که معادلات مربوط به فرسایش هر قطعه را داشته باشید و آنها را به همراه سایر معادلات در مدل هر قطعه وارد نمایید. حال که چهار مدل را ایجاد نمودید، می‌توانید از آنها به هر تعداد که بخواهید در یک محیط گرافیکی کاربر پسند استفاده نمایید تا کل سیستم آبرسانی را خلق کنید و ابعاد و اندازه‌ها را وارد نموده و سایر کارها را به زبان Modelica بسپارید، این زبان سایر فعالیتهای لازم برای تولید کل معادلات سیستم، استخراج پارامترها، ساده‌سازی، اختصاص حافظه و حل آن را برای شما انجام خواهد داد و پس از انجام محاسبات می‌توانید نتایج حاصل را به صورت دینامیک مشاهده نمایید. حتی می‌توانید از کتابخانه آماده اجزاء استفاده نموده و بدون نوشتن حتی یک خط برنامه، سیستم خود را شبیه‌سازی نمایید. البته اضافه نمودن معادلات فرسایش قطعات را باید خودتان انجام دهید که کار بسیار ساده است. کتابخانه آماده این زبان در حال حاضر شامل مدل تعداد زیادی از قطعات و تجهیزات می‌باشد. با استفاده از این کتابخانه، شبیه‌سازی از این هم ساده‌تر خواهد بود. کتابخانه شامل اجزاء ریاضی، منطقی، الکتریکی، الکترونیکی، مغناطیسی، مکانیکی، ترمودینامیکی، سیالاتی، حرارتی، نئوماتیک و . . . می‌باشد. همچنین در این کتابخانه بیش از ۴۰۰ پارامتر موجود در دستگاه SI با واحد آنها در نظر گرفته شده که ایجاد سیستمهای مختلف را بسیار آسان نموده است. با توجه به این که مفهوم زمان در این زبان پیاده‌سازی شده است، مدلها می‌توانند مانند تجهیزات واقعی رفتار دینامیک داشته

باشند. یکی دیگر از جنبه‌های این زبان، امکان استفاده سیستم‌های مختلف به صورت توأم در مدلسازی است. یک سیستم واقعی مثلاً یک اتومبیل را در نظر بگیرید، این سیستم دارای بخشها و قطعات مختلفی از جمله تجهیزات مکانیکی و تولید قدرت، الکتریکی، سیستمهای ناوبری و کنترلی و سیستمهای هیدرولیک و ارتعاشاتی، آیرودینامیک بدنه و رفتار راننده می‌باشد که همگی در کنار هم فعالیت می‌کنند و برای شبیه‌سازی یک اتومبیل کامل لازم است که رفتار همه بخش‌ها را در کنار هم محاسبه نماییم. در این زبان امکان استفاده از تجهیزات مربوط به حوزه‌های مختلف در کنار یکدیگر وجود دارد، یعنی شما می‌توانید همزمان تمام این اجزاء و سیستمهای مختلف را در مدلسازی استفاده نمایید.

امکان توسعه آسان مدلها، نگهداری و گسترش مدلهای مختلف را تضمین می‌کند. یعنی برای تغییر بخشی از مدل در آینده نیازی به مدلسازی دوباره نیست و کافی است که تغییرات لازم را در مدل قبل اعمال نمایید. فرض کنید شما یک سیستم مهندسی را در این زبان پیاده‌سازی نمودید و مدتها بعد خواستید که بخشی از مدل خود را تغییر بدهید، و رفتار کل سیستم را با توجه به طراحی جدید بررسی نمایید، این کار به سادگی و با انجام تغییرات اندک امکانپذیر است.

شما با داشتن این زبان، یک آزمایشگاه مجازی در کامپیوتر خود دارید.

درباره این کتاب

این کتاب اصول مقدماتی شبیه‌سازی و مدلسازی ساختار یافته به کمک زبان Modelica را به صورت قدم به قدم به شما آموزش خواهد داد. فصلهای این کتاب بر اساس دو محیط شبیه‌سازی، Mathmodelica و OpenModelica ارائه خواهد شد. نرم افزار این محیط‌ها در CD همراه کتاب موجود است. مخاطب این کتاب عمدتاً مهندسان و دانشجویان رشته‌های فنی هستند، در عین حال این کتاب می‌تواند برای محققان و پژوهشگران رشته‌های مختلف که به شبیه‌سازی دینامیک سیستمهای واقعی نیاز دارند، راهگشا باشد.

چرا این کتاب را باید بخوانید؟

خواه یک مهندس طراح در یک شرکت بزرگ باشید، خواه یک دانشجو یا دانش‌آموز بارها پیش آمده است که آرزو کنید، ای کاش یک آزمایشگاه برای ارزیابی طراحی‌ها و ایده‌های خودم در اختیار داشتم، یا بهتر از آن، ای کاش می‌شد یک سیستم واقعی را بدون آزمایش تجربی مورد تجزیه و تحلیل قرار داد، کاش یک آزمایشگاه مجازی در کامپیوتر خود داشتید که می‌شد از آن برای شبیه‌سازی سیستمهای مختلف استفاده نمود، ای کاش می‌شد محاسبات یک سیستم را آسانتر انجام داد، ای کاش می‌شد بدون دردسرها و پیچیدگی‌های زبانهای برنامه‌نویسی، از کامپیوتر برای انجام محاسبات یک سیستم استفاده کرد؛ ای کاش می‌شد سیستمهایی را که به زحمت برنامه‌نویسی نموده‌اید به راحتی توسعه می‌دادید یا از اجزائی که خلق نموده‌اید در محاسبات دیگر استفاده می‌نمودید

و لازم نبود همه چیز را از ابتدا بنویسید، زبان Modelica برای همین کار خلق شده است و پاسخی به این آرزوهاست.

زبان Modelica امکان شبیه‌سازی و مدلسازی آسان هر سیستمی را برای شما فراهم می‌کند. این سیستم می‌تواند شامل قطعات الکتریکی، مکانیکی، مغناطیسی، هیدرولیکی، سیالاتی و ... به صورت توأم باشد. پس هر یک از سیستمهای دنیای واقعی را می‌توانید با این زبان مدلسازی نمایید. این سیستم می‌تواند یک موتور الکتریکی، ربات، اتومبیل، بدن یک موجود زنده و هر سیستم قابل تصوری باشد. امکان شبیه‌سازی سیستمهای گسسته نیز در این زبان فراهم شده است. این نکته که با کامپیوتر می‌توان هر سیستمی را شبیه‌سازی نمود بسیار هیجان‌انگیز است. اگر شما هم می-خواهید مدلسازی سیستمهای واقعی را انجام دهید، این کتاب برای شماست. این ابزار می‌تواند قدرت و سرعت شما را در شبیه‌سازی و انجام محاسبات سیستمهای مختلف به طرز چشمگیری افزایش دهد.

این کتاب برای کیست؟

این کتاب برای کسانی مناسب است که قصد داشته باشند سیستمهای مهندسی را به صورت نرم‌افزاری مدلسازی نمایند. این سیستم می‌تواند به طور همزمان دارای قطعات ریاضی، کنترلی، مکانیکی، الکتریکی، الکترونیکی، سیالاتی و انتقال حرارتی و ... باشد. شما می‌توانید (به شرط آن که معادلات حاکم بر سیستم خود را بدانید) قطعات دلخواه خودتان را ایجاد کنید یا می‌توانید از کتابخانه بزرگ این زبان استفاده نمایید.

مخاطب اصلی این کتاب مهندسان و دانشجویان رشته‌های فنی هستند، اما در عین حال می-تواند مورد استفاده سایر علاقه‌مندان به شبیه‌سازی، محققان و پژوهشگران قرارگیرد. هدف این کتاب آموزش مدلسازی سیستمهای فیزیکی به زبان Modelica است. با استفاده از کتابخانه گسترده‌ای که این زبان در سیستمهای مختلف فراهم نموده است، می‌توانید حتی بدون نوشتن یک خط برنامه سیستمهای زیادی را شبیه‌سازی نمایید. همچنین با امکانات مدلسازی شیءگرای فراهم شده، مدلسازی هر سیستمی که بتوان رفتار آن را به صورت معادلات ریاضی نوشت با این زبان ممکن است. داشتن اطلاعات در زمینه معادلات دیفرانسیل می‌تواند به درک بهتر مدلها کمک کند، اما ضروری نیست.

در این کتاب چه خواهید آموخت؟

- زبان مدلسازی Modelica (یک زبان بسیار قوی در زمینه شبیه سازی دینامیک سیستم‌های مختلف)

- محیط مدلسازی MathModelica (یک محیط گرافیکی بسیار کاربر پسند برای شبیه‌سازی سیستم‌های مختلف)
- محیط مدلسازی OpenModelica (یک محیط Open Source برای مدلسازی با زبان Modelica که می‌تواند به راحتی و بدون هزینه مورد استفاده محققان قرار گیرد)

هدف این کتاب چیست؟

آموزش مدلسازی سیستمهای فیزیکی با استفاده از زبان مدلسازی Modelica اولین هدف این کتاب می‌باشد. هنگامی که این کتاب را مطالعه می‌کنید، به راحتی با این زبان و امکاناتش آشنا خواهید شد و مدلسازی ساختار یافته سیستم‌های مختلف را خواهید آموخت. آموزش این زبان در بخش اول کتاب با استفاده عملی از محیط مدلسازی MathModelica آغاز می‌گردد. این محیط با توجه به دارا بودن رابط کاربر مناسب و قدرت بالا، به عنوان یک فضای مناسب برای مدلسازی مطرح است. این محیط در حال پیشرفت، فضای بسیار مناسب، ساده و قدرتمندی جهت آموزش مدلسازی فراهم نموده است. بخش دوم شما را با محیط OpenModelica آشنا می‌نماید. این محیط Open Source بوده و تلاشهای زیادی برای گسترش آن صورت گرفته است. این محیط با داشتن فضای ساده جهت مدلسازی به زبان Modelica، برای یادگیری این زبان بسیار مناسب است. همچنین با توجه به مجانی بودن ابزارهای این زبان، بستر مناسبی برای مقالات علمی می‌باشد. گستردگی و قدرت این محیط‌ها، در عین سادگی شما را شگفت زده خواهد نمود. اگر توانایی برنامه‌نویسی داشته باشید، خودتان نیز می‌توانید بخشی از این پیشرفت شوید و در توسعه محیط‌های این زبان سهیم گردید.

چگونه از این کتاب استفاده نمایید؟

دو روش را برای استفاده از این کتاب در نظر گرفته‌ایم.

روش اول استفاد از این کتاب به عنوان خودآموز مدلسازی به زبان Modelica است. چیدمان فصول این کتاب برای همین منظور می‌باشد. بنابراین کتاب را از ابتدا شروع نمایید و به ترتیب تا پایان ادامه دهید. برای یادگیری بهتر تمرینات آخر هر فصل را انجام دهید.

روش دوم استفاده از این کتاب به عنوان متن کمک آموزشی یک ترم درس مدلسازی سیستمهای انرژی یا سایر دروس مرتبط است. در این صورت دو تا سه جلسه به آموزش موارد ابتدایی محیط MathModelica و OpenModelica اختصاص دهید و مطالعه بخش اول و دوم را به دانشجویان بسپارید. سایر جلسات را به آموزش بخش سوم اختصاص دهید، همزمان با پیشرفت در بخش سوم از تمرینات عملی مهیا شده استفاده نموده و دانشجویان را تشویق نمایید تا سیستمهای مختلفی را عملاً شبیه‌سازی نمایند.

مقدمه

تلاش برای انجام محاسبات در مسائل مختلف به کمک کامپیوتر اغلب از برنامه‌نویسی شروع می‌شود. اما در زبانهای برنامه‌نویسی سنتی شما باید دانش فراوانی از برنامه‌نویسی داشته باشید و اغلب، زمان لازم برای برنامه‌نویسی بیش از زمانی است که صرف شناخت ساختار مسئله می‌گردد. این فرآیند اغلب از چنان پیچیدگی برخوردار است که با این زبانهای برنامه نویسی کمتر کسی به شبیه‌سازی سیستمهایی با پیچیدگی سیستمهای واقعی فکر می‌کند و حتی اگر کسی یک سیستم را با زبانهای برنامه‌نویسی سنتی مدلسازی نماید، نگهداری برنامه تهیه شده و استفاده از آن برای سایر سیستمها کار بسیار سختی است.

با این حال مفهوم و امکاناتی که مدلسازی در اختیار طراحان، سازندگان، برنامه‌ریزان و استفاده‌کنندگان سیستمهای مختلف قرار می‌دهد، چنان ارزشمند است که تلاشهای بسیاری برای شیبه‌سازی سیسستم‌های مختلف توسط بشر انجام شده است. این تلاشها با اختراع کامپیوتر، با محاسبات فضاپیماها و تسلیحات نظامی شروع شد و این روزها با وجود کامپیوترهای خانگی، مدلسازی راه خود را به پروژه‌های دانشجویی باز کرده است. در هر حال شبیه‌سازی به معنی شناخت یک سیستم و دستیابی به دانش طراحی آن است، با شبیه‌سازی می‌توان شرایط بهینه طراح‌های موجود را یافت و برای بهبود آنها برنامه‌ریزی نمود یا قبل از ساخت طرح‌های جدید به بررسی و ارزیابی آنها پرداخته و تأثیر پارمترهای مختلف را بر کارکرد یا قیمت طرح نهایی به دست آورد. از مزیت‌های مهم مدلسازی امکان تحلیل سیستم با هزینه بسیار کمتر از آزمایش و داشتن اطلاعات فراوان و سریع در خصوص سیستم حتی قبل از ساختن مدل آن می‌باشد. هرچقدر مدل شبیه‌سازی شده به واقعیت نزدیک‌تر باشد، نتایج بهتری از محاسبات به دست خواهد آمد. از طرفی برای نزدیکتر شدن پاسخهای محاسبات به حالت واقعی باید مسائل بیشتری را در مدلسازی در نظر گرفت و در نتیجه مدل از نظر محاسباتی سنگینتر خواهد شد و به همین ترتیب نوشتن کد لازم، نگهداری کد نوشته شده و استفاده مجدد از آن سخت‌تر خواهد شد.

این تلاش با معرفی زبانهای شبیه‌سازی، مسیر مشخصی به خود گرفت. با این زبانها امکان شبیه‌سازی هر سیستم مهندسی به راحتی فراهم می‌گردد. این زبانها با حذف بسیاری از جزئیات برنامه‌نویسی امکان تمرکز بر مدل اصلی را برای مهندسان سیستم فراهم نموده‌اند. با این شرایط لازم نیست شما برنامه‌نویس باشید تا بتوانید یک سیستم را مدلسازی نمایید. فقط کافی است که شما دانش کافی در مورد سیستم داشته باشید. سایر فعالیت‌های لازم برای شبیه‌سازی را زبان‌های شبیه‌سازی انجام خواهند داد. زبان Modelica از جمله این زبان‌هاست که در سال ۱۹۹۶ خلق گردید و از همان

ابتدا امکان مدلسازی سیستمهایی با فیزیک مختلف مثل سیستمهای الکتریکی و مکانیکی را به صورت توأم فراهم نمود. ساختار باز این زبان امکان گسترشش را برای بسیاری از پژوهشگران فراهم نموده است و در حال حاضر کتابخانه‌های مختلفی برای سیستم‌های مختلف نوشته شده است که می‌توانند همزمان برای شبیه‌سازی سیستم‌های مختلف مورد استفاده قرار گیرند. در کنار داشتن کتابخانه گسترده‌ای از قطعات مختلف، امکان نوشتن قطعات جدید را نیز به راحتی فراهم نموده است.

ساختار مدلسازی که ما در این کتاب از آن استفاده خواهیم کرد، تقسیم سیستم‌های بزرگ مهندسی به اجزاء کوچکی است که بلوک نامیده می‌شود. هر بلوک هدف مشخصی را در سیستم برآورده می‌کند و می‌تواند جایگزین یک فرآیند ساده شود. معادلات حاکم بر این بلوکها شامل معادلات جبری دیفرانسیلی است و این معادلات از نوشتن قوانین فیزیکی حاکم بر سیستم مانند معادلات بقای جرم، انرژی، جریان الکتریکی و جاذبه و ... برای بلوکها به دست می‌آید. با حل کل این معادلات می‌توان سیستم را تحلیل نمود.

عبدالحمید انصاری نسب

بهار ۹۲

فهرست مطالب

بخش اول آشنایی با نرم افزار SYSTEMMODELER..............................۱

فصل ۱ محیط شبیه سازی..۱

۱-۱) ساختار نرم افزار SYSTEMMODELER ۱

۱-۲) نصب نرم افزار ۱

۱-۳) فعال سازی نرم افزار ۳

۱-۴) ساختار نرم افزار SYSTEMMODELER ۴

فصل ۲ اولین مدلسازی..۸

۲-۱) مدل HELLO WORLD ۸

۲-۲) برای مدل یک آیکون بسازید ۱۳

۲-۳) معادلات دیفرانسیل ۱۴

فصل ۳ سیستمهای فیزیکی مختلف به صورت توأم..................................۱۹

۳-۱) موتورDC ۱۹

۳-۲) محور صلب و محور انعطاف پذیر ۲۷

۳-۳) سیستم کنترل ۳۱

۳-۴) تحلیل میزان حساسیت ۳۵

فصل ۴ مدار الکتریکی ساده بر پایه قطعات.. ۳۹

۴-۱) مدلسازی سببی و غیر سببی ۳۹

۴-۲) مدلسازی جریان سیگنال ۴۰
۴-۳) مدار برپایه جریان سیگنال (سببی) ۴۱
۴-۴) مدار بر اساس قطعات (مدلسازی غیرسببی) ۴۴

فصل ۵ قطعات کاربر – آونگ مرکب ۴۸

۵-۱) آونگ ۴۸
۵-۲) مدل آونگ مرکب ۵۱

فصل ۶ توابع خارجی و سیگنال CHIRP ۵۴

۶-۱) تابع CHIRP ۵۴
۶-۲) مدلسازی ۵۴

فصل ۷ شبیه سازی مخازن ذخیره ۶۰

۷-۱) مدل ساده مخزن ۶۰
۷-۲) مدل مخزن بر اساس قطعات ۶۲
۷-۲-۱) درگاه ها ۶۵
۷-۲-۲) ایجاد تابع محدودیت برای شیرکنترل ۶۶
۷-۲-۳) اجزای مخزن ۶۶
۷-۲-۴) کنترل کننده ۶۹
۷-۲-۵) سیستم مخزن کوچک ۷۰
۷-۳) مخزن با کنترل کننده PID پیوسته ۷۱
۷-۴) سیستم با سه مخزن ۷۳

فصل ۸ آونگ معکوس ۷۵

۸-۱) آونگ معکوس ۷۵

فصل ۹ توصیه هایی در مدلسازی.. ۷۷

۹-۱) مقادیر اولیه ۷۷

۹-۱-۱) خاصیت START.. ۷۷

۹-۱-۲) مقادیر حدس.. ۷۷

۹-۱-۳) بخش معادلات و الگوریتم‌های مقدار دهی اولیه.. ۷۸

۹-۲) اتفاقات ۷۹

۹-۳) کتابخانه MULTIBODY ۸۰

۹-۳-۱) مقدار اولیه.. ۸۰

۹-۳-۲) زاویه و موقعیت شیء.. ۸۱

۹-۳-۳) پویا نمایی.. ۸۰

۹-۳-۴) استفاده از شکلهای CAD.. ۸۲

۹-۴) پیشنهادات عمومی ۸۳

بخش دوم محیط OPENMODELICA.. ۸۵

فصل ۱۰ نرم افزار OPENMODELICA.. ۸۵

۱۰-۱) OMNOTEBOOK و DRMODELICA ۸۶

۱۰-۲) ویرایشگر متن با برنامه نویسی LITERATE ۸۶

۱۰-۲-۱) ویرایشگر متن OMNOTEBOOK وMATHEMATICA.. ۸۶

۱۰-۳) سیستم آموزشی DRMODELICA ۸۸

۱۰-۳-۱) سلولها.. ۹۲

۱۰-۳-۲) مکان نماها.. ۹۳

۱۰-۴) ایجاد مدل مخزن ۹۳

۱۰-۵) ایجاد مدل پیشرفته مخزن ۹۷

۱۰-۵-۱) شیرکنترل ۹۷

۱۰-۶) آشنایی با OMShell ۱۰۰

۱۰-۷) بخش Interactive session همراه با مثال ۱۰۱

۱۰-۷-۱) شروع کار با Interactive session ۱۰۱

۱۰-۷-۲) امتحان کردن تابع BubbleSort ۱۰۱

۱۰-۷-۳) کتابخانه Modelica و مدل DCMotor ۱۰۳

۱۰-۷-۴) تابع val ۱۰۶

۱۰-۷-۵) مدلهای BouncingBall و Switch ۱۰۶

۱۰-۷-۶) پاک کردن تمامی مدلها ۱۰۹

۱۰-۷-۷) مدل VanDerPol و رسم پارامتری ۱۰۹

۱۰-۷-۸) برنامه نویسی با دستور IF و حلقههای FOR-LOOP و WHILE-LOOP ۱۱۱

۱۰-۷-۹) متغیرها، توابع و انواع متغیرها ۱۱۲

۱۰-۷-۱۰) دریافت اطلاعات در خصوص علت خطاها ۱۱۳

۱۰-۷-۱۱) سایر قالب های خروجی شبیه سازی ۱۱۳

۱۰-۷-۱۲) استفاده از توابع خارجی ۱۱۳

۱۰-۷-۱۳) فراخوانی Model Query و Manipulation API ۱۱۵

۱۰-۷-۱۴) خروج از OpenModelica ۱۱۷

۱۰-۸) تولید XML مدل ۱۱۷

۱۰-۹) خروجی مدل در قالب Matlab ۱۱۸

۱۰-۱۰) دستوراتی برای Interactive Session Handler ۱۱۸

فصل ۱۱ گرافیک ۱۲۰

۱۱-۱) رسم نمودار ساده دوبعدی ۱۲۱

۱۱-۲) همه توابع رسم و گزینه های آنها ۱۲۲

۱۱-۳) بزرگنمایی ۱۲۳

۱۱-۴) رسم نمودار در حین شبیه سازی ۱۲۳

فصل ۱۲ استفاده از ویرایشگر گرافیکی مدل ۱۲۴

۱-۱۲) ایجاد یک پروژه جدید ۱۲۴

فصل ۱۳ ویرایشگر ارتباطی OPENMODELICA ۱۳۰

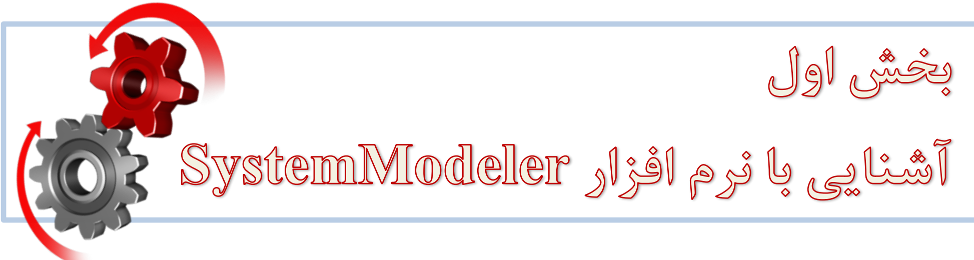
۱-۱۳) درباره OMEDIT ۱۳۰
۲-۱۳) چگونه OMEDIT را شروع کنیم؟ ۱۳۰
۳-۱۳) مدل مقدماتی در OMEDIT ۱۳۱
۱-۳-۱۳) ساخت فایل جدید ۱۳۱
۲-۳-۱۳) اضافه کردن اجزای مدل‌ها ۱۳۲
۳-۳-۱۳) برقراری ارتباط بین دو مدل ۱۳۲
۴-۳-۱۳) شبیه سازی مدل ۱۳۳
۵-۳-۱۳) رسم متغیرها از مدل‌های شبیه سازی شده ۱۳۴
۴-۱۳) پنجره ها ۱۳۵
۱-۴-۱۳) پنجره کتابخانه ۱۳۵
۲-۴-۱۳) پنجره طراحی ۱۳۵
۳-۴-۱۳) پنجره رسم ۱۳۶
۴-۴-۱۳) پنجره پیغام ۱۳۶
۵-۴-۱۳) پنجره مستندات ۱۳۶
۵-۱۳) محاوره ۱۳۷
۱-۵-۱۳) محاوره جدید ۱۳۷
۲-۵-۱۳) محاوره شبیه سازی ۱۳۷
۳-۵-۱۳) محاوره خواص مدل ۱۳۷
۴-۵-۱۳) پنجره خواص مدل ۱۳۸

بخش سوم آموزش زبان MODELICA ۱۳۹

فصل ۱۴ مروری کوتاه بر MODELICA ۱۳۹

۱-۱۴) شروع ۱۴۰
۲-۱۴) متغیرها ۱۴۲
۳-۱۴) توضیحات ۱۴۴
۴-۱۴) ثابتها ۱۴۵
۵-۱۴) مدلسازی شیءگرای ریاضی ۱۴۶
۶-۱۴) کلاس‌ها و نمونه‌ها ۱۴۷
۱۴-۶-۱) ایجاد نمونه‌ها ۱۴۸
۱۴-۶-۲) مقدار دهی اولیه ۱۴۹
۱۴-۶-۳) کلاسهای محدود ۱۵۰
۱۴-۶-۴) استفاده دوباره از کلاس‌های اصلاح شده ۱۵۰
۱۴-۶-۵) کلاس‌های داخلی ۱۵۲
۷-۱۴) توارث ۱۵۲
۸-۱۴) کلاس‌های عمومی ۱۵۳
۱۴-۸-۱) اشیاء به عنوان پارامترهای قابل جایگزینی کلاس ۱۵۳
۱۴-۸-۲) انواع به عنوان پارامترهای قابل جایگزینی کلاس ۱۵۴
۹-۱۴) معادلات ۱۵۵
۱۴-۹-۱) ساختار معادلات تکراری ۱۵۷
۱۰-۱۴) مدلسازی فیزیکی غیرسببی ۱۵۸
۱۴-۱۰-۱) مدلسازی فیزیکی در مقابل مدلسازی سببی ۱۵۸
۱۱-۱۴) قابلیت مدلسازی جزء به جزء ۱۶۰
۱۴-۱۱-۱) اجزاء ۱۶۱
۱۴-۱۱-۲) دیاگرام اتصالات ۱۶۲
۱۴-۱۱-۳) درگاه ها و کلاس‌های اتصال دهنده ۱۶۳
۱۴-۱۱-۴) اتصالات ۱۶۴
۱۲-۱۴) کلاس‌های جزیی مشخصات عمومی را بیان می‌کنند ۱۶۵
۱۴-۱۲-۱) استفاده مجدد از کلاس‌های جزیی ۱۶۶
۱۳-۱۴) کتابخانه قطعات الکتریکی ۱۶۷
۱۴-۱۳-۱) مقاومت ۱۶۷

۱۴-۱۳-۲) خازن ۱۶۷
۱۴-۱۳-۳) القاگر(سلف) ۱۶۸
۱۴-۱۳-۴) منبع ولتاژ ۱۶۸
۱۴-۱۳-۵) زمین ۱۶۹
۱۴-۱۴) مدل مدار ساده ۱۷۰
۱۴-۱۵) آرایه‌ها ۱۷۱
۱۴-۱۶) ساختار الگوریتمی ۱۷۴
۱۴-۱۶-۱) الگوریتم‌ها ۱۷۴
۱۴-۱۶-۲) دستورات ۱۷۵
۱۴-۱۶-۳) توابع ۱۷۶
۱۴-۱۶-۴) توابع خارجی ۱۷۸
۱۴-۱۶-۵) نگاه به الگوریتم‌ها به عنوان تابع ۱۷۹
۱۴-۱۷) مدل کردن ترکیبی ۱۸۰
۱۴-۱۸) بسته‌ها ۱۸۴
۱۴-۱۹) پیاد سازی واجرای MODELICA ۱۸۵
۱۴-۱۹-۱) ترجمه دستی مدل مدار ساده ۱۸۷
۱۴-۱۹-۲) تبدیل به فضای حالت ۱۸۹
۱۴-۱۹-۳) روش حل ۱۹۱



فصل ۱ محیط شبیه سازی

۱-۱) ساختار نرم افزار SystemModeler

این نرم‌افزار با داشتن یک رابط کاربری گرافیکی بسیار قوی و کارآمد، یکی از بهترین نرم‌افزارهای موجود برای یادگیری مدلسازی به زبان Modelica است.

این بخش شامل مثالهایی است که شما را برای شروع مدلسازی به کمک نرم‌افزار SystemModeler آماده می‌کند. این مثالها دارای توضیحات قدم به قدم و مشروح در خصوص روش ایجاد و شبیه‌سازی مدلهای مختلف می‌باشد. در هر بخش، تمریناتی برای درک بهتر این نرم‌افزار در نظر گرفته شده است. در صورتی که شما از ویرایش پیشرفته SystemModeler استفاده می‌کنید، امکان برقراری ارتباط با نرم‌افزار ریاضی Mathematica نیز فراهم شده است. با این روش استفاده از همه امکانات برنامه بسیار قوی Mathematica در کنار قدرت شبیه‌سازی SystemModeler امکانپذیر می‌گردد.

برای یادگیری بهتر توصیه می‌گردد تمام فصول را به ترتیب ارائه آنها دنبال کنید و مثالهای داده شده را خودتان پیاده‌سازی نموده و نتایج شبیه‌سازی را مشاهده نمایید. اگر قصد ندارید مثالها را خودتان تایپ کنید، مثالهای این فصل، پس از نصب نرم‌افزار SystemModeler در کادر Library Browser در کتابخانه IntroductoryExamples در دسترس شما می‌باشد.

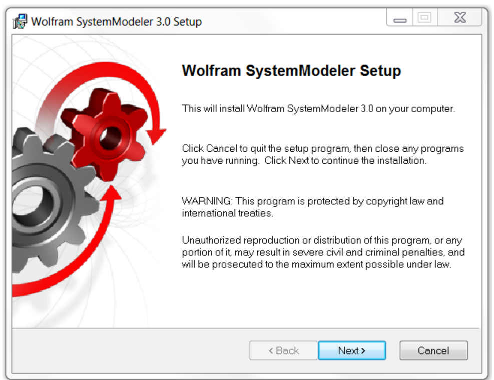
در این فصل اطلاعات اولیه مورد نیاز در خصوص نصب نرم‌افزار و همچنین محیط نرم‌افزار System Modeler ارائه خواهد شد.

۱-۲) نصب نرم افزار

نرم‌افزار SystemDesigner در CD ضمیمه این کتاب موجود است. همچنین می‌توانید جدیدترین ویرایش این نرم‌افزار را از آدرس زیر دانبار گذاری نمایید:

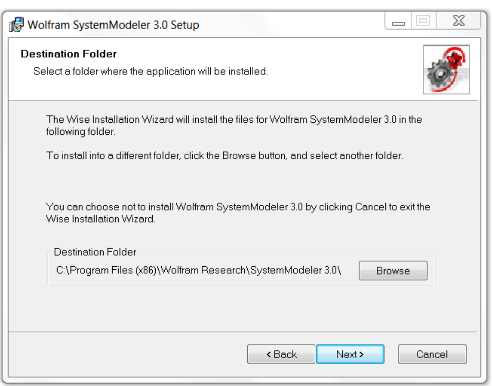
http://www.wolfram.com/system-modeler/

فایل قابل نصب این نرم‌افزار با نام SystemModeler.exe را اجرا نمایید. نصب این نرم‌افزار بسیار ساده است. شکل ۱-۱ نمایان خواهد شد:



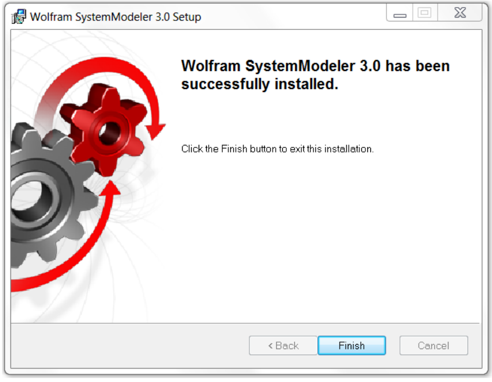
شکل ۱-۱.شروع نصب نرم افزار.

در شکل ۱-۲ دکمه Next> را کلیک کنید. کادر مشخص کننده محل نصب به شکل زیر نمایش داده خواهد شد:



شکل ۱-۲. قدم دوم در نصب نرم افزار.

پیشنهاد می‌گردد در هنگام نصب همه پیش‌فرض‌های نرم‌افزار را بپذیرید. برای ادامه نصب در کادرهای نمایان شده بعدی نیز دکمه Next> را کلیک کنید. پس از پایان نصب شکل ۱-۳ به عنوان موفقیت نصب نرم‌افزار نمایش داده خواهد شد:

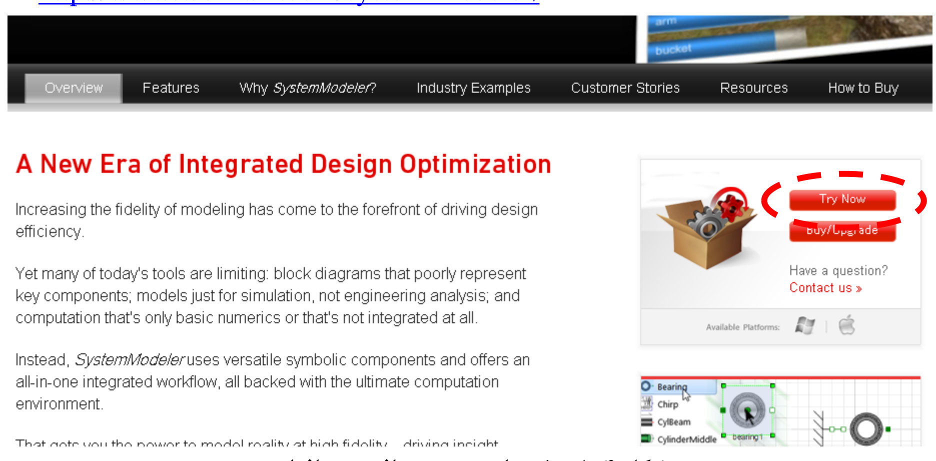


شکل ۳-۱. نصب موفق.

۳-۱) فعال سازی نرم افزار

جهت فعالسازی این نرم‌افزار لازم است جدیدترین ویرایش نرم افزار را از آدرس زیر درخواست نمایید:

http://www.wolfram.com/system-modeler/



شکل ۴-۱. صفحه اینترنتی دریافت نرم افزار

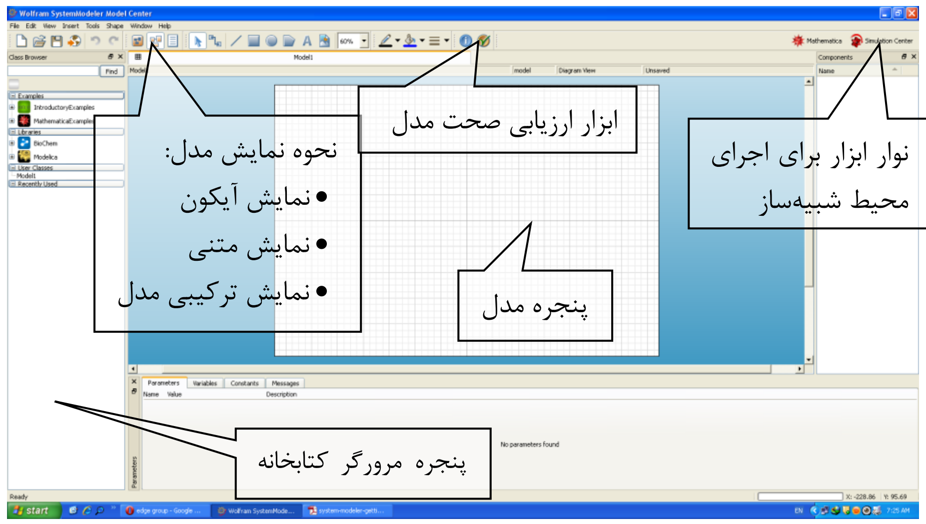
پس از انجام مراحل درخواست که شامل عضویت شرکت Wolfram است و پر کردن فرم درخواست می‌باشد، کد فعالسازی و تارنمای دریافت نرم افزار به ایمیل شما ارسال خواهد شد. نرم افزار را اجرا کنید، شکل ۵-۱ را خواهید دید:

شکل ۵-۱. کادر فعالسازی نرم افزار.

در این شکل کد فعالسازی را وارد نمایید و کلید Activate را فشار دهید. پس از چند لحظه تماس نرم‌افزار با شبکه، فعالسازی انجام خواهد شد. توجه فرمایید که کد فعالسازی فقط برای یک کامپیوتر معتبر است.

۱-۴) ساختار نرم افزار SystemModeler

روند شبیه‌سازی در این نرم افزار شامل دو مرحله می‌باشد، ابتدا مدل را در محیط Modeling Center طراحی نموده، سپس مدل را در محیط Simulation Center شبیه‌سازی خواهید نمود. محیط Modeling Center شامل بخشهایی است که در شکل ۶-۱ مشاهده می‌نمایید.



شکل ۶-۱. محیط نرم افزار.

پس از طراحی مدل در این محیط برای انجام شبیه‌سازی لازم است محیط SimulationCenter را اجرا نمایید. انتقال مدل طراحی شده به طور خودکار انجام خواهد شد. کافی است روی آیکون در منو ابزار نرم افزار Model Center کلیک کنید. در صورتی که مدلی را در بخش SystemDesigner طراحی کرده باشید قبل از اجرای این نرم افزار مدل شما ترجمه شده و برای طراحی آماده می‌گردد.

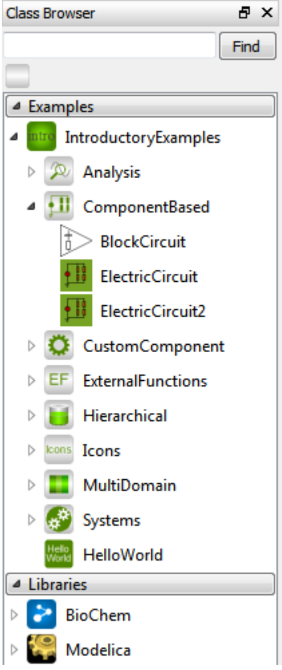
بخشهای مختلف صفحه اصلی این محیط را در شکل ۷-۱ مشاهده می‌نمایید.

شکل ۷-۱. محیط نرم افزار Simulation Center.

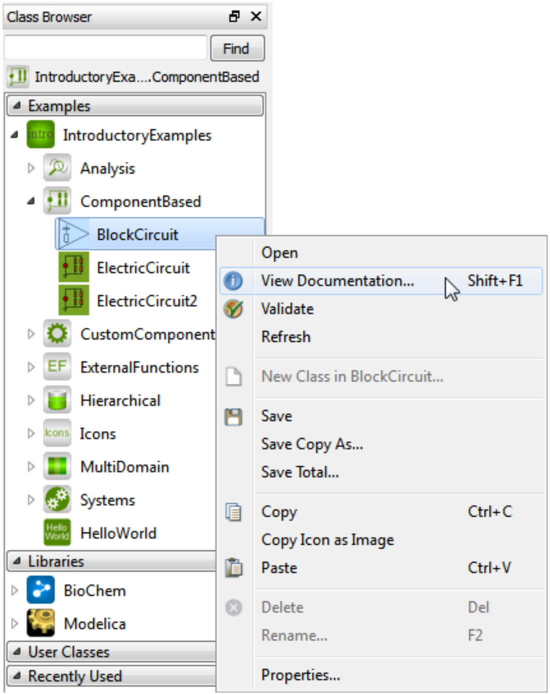
برای مشاهده اغلب مثالهای این کتاب می‌توانید از پنجره سمت چپ (Library Browser) کتابخانه Introductory Examples را انتخاب نمایید.

دو بار تلیک کردن روی نام هر کتابخانه (IntroductoryExamples) باعث باز شدن آن کتابخانه می‌گردد. همچنین می‌-توانید علامت + را به - تبدیل کرده تا کتابخانه باز شود. مشاهده می‌کنید که این کتابخانه شامل چندین کتابخانه دیگراست، روی یک کتابخانه به دلخواه (برای مثال کتابخانه اولComponentbased) دو بار تلیک کنید مثالهای این کتابخانه باز می‌شود در نهایت با دو بارتلیک کردن روی هر کدام از مثالها، مثال مورد نظر در پنجره ویرایش مدل نمایش داده می‌شود.

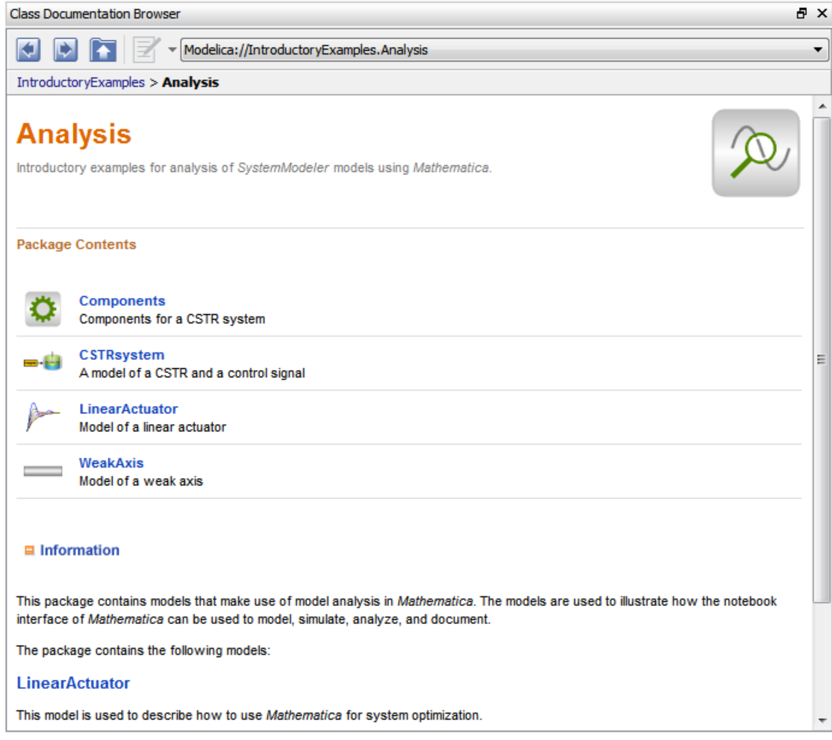
علاوه بر مثالهای این کتاب، اطلاعات جامعی درباره هر کتابخانه و مدلهای موجود در کتابخانه و همچنین مثالهای مقدماتی با تلیک راست روی هر کتابخانه یا مدل و انتخاب گزینه ViewDocumentation از منو باز شده در دسترس شماست.



شکل ۸-۱.ساختار مثال های مقدماتی در Library browser.



شکل ۹-۱.نمایش اطلاعات یک مدل.



شکل ۱۰-۱. نمایش اطلاعات مدل‌ها.

فصل ۲ اولین مدلسازی

برای شروع آموزش نرم‌افزار Modelica اساسی‌ترین قدم پیاده‌سازی یک معادله دیفرانسیل در این نرم افزار است. معادله دیفرانسیل پایه‌ای‌ترین معادله ریاضی برای شبیه‌سازی یک مدل فیزیکی دینامیک است.دراین مثال یک معادله دیفرانسیل در نرم‌افزار پیاده‌سازی و شبیه‌سازی می‌گردد. همچنین روش ساختن یک آیکون برای این مدل شرح داده می‌شود.

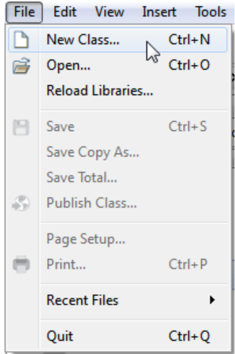
۲-۱) مدل Hello World

یک سنت قدیمی و رایج در آموزش هر زبان برنامه‌نویسی، نوشتن برنامه‌ای برای نمایش عبارت Hello World می‌باشد. با توجه به این که زبان Modelica (زبان استفاده شده در نرم‌افزار SystemModeler) برای حل معادلات خلق شده است، چاپ یک عبارت، کار معقولی به نظر نمی-رسد؛ درعوض به عنوان اولین مثال (مدل Hello World) یک معادله دیفرانسیل ساده را حل خواهیم کرد:

$$\dot{x} = -x \qquad (۲-۱)$$

متغیر x در این معادله یک متغیر دینامیکی است (یک متغیر حالت) که مقدارش در هر زمان می‌تواند تغییر کند. مشتق زمانی x ($\dot{x}$) در modelica به صورت der(x) نوشته می‌شود. در زبان modelica مدل ساختاری است که معادلات مربوط به یک قطعه یا سیستم واقعی را نگه می‌دارد. برای نوشتن این مثال نیز یک مدل ایجاد می‌کنیم. همه ساختارهای این زبان از نوع کلاس (ساختار استاندارد برنامه نویسی) هستند. پس در ابتدا یک کلاس از نوع مدل ایجاد می‌کنیم.

مرحله اول را با اجرای نرم‌افزار System Designer (محیط ویرایشگر SystemModeler) و ایجاد یک مدل جدید در این نرم افزار شروع خواهیم کرد. این مدل درون هیچ کتابخانه‌ای قرار نگرفته و مستقل است. برای ایجاد مدل مسیر ...File >> New Class را مطابق شکل ۱-۲ دنبال کنید:



شکل ۱-۲. انتخاب کردن New Class ازمنوی File.

پس از انتخاب گزینه New Class، پنجره این دستور باز خواهد شد (شکل بالا). از کشو Restriction، گزینه Model را به عنوان نوع کلاس انتخاب کنید. در این پنجره برای مدل، نام و شرح (اختیاری) مشخص خواهیم کرد. در این مثال نام مدل را HelloWorld می‌گذاریم و در بخش شرح A differential equation را می‌نویسیم.

شکل ۲-۲.مشخص کردن نام و تعریف برای مدل.

هنگامی که کلید OK را تلیک کنید، مدل HelloWorld ایجاد خواهد شد و در Library Browser (کادر سمت چپ) نشان داده می‌شود. حال مدل در پنجره اصلی باز شده است. بر روی آیکون modelica text view در میله ابزار تلیک کنید یا از میانبر ctrl+3 استفاده کنید تا پنجره اصلی به حالت متنی تغییر کند.

شکل ۳-۲.آیکون modelica text view در میله ابزار ویرایشگر مدل

نمایش متنی مدل به شکل زیر خواهد بود

```
model HelloWorld "A differential equation"
  ¤;
end HelloWorld;
```

نماد ¤ موجود در سطر دوم شامل اطلاعاتی درباره نمایش گرافیکی مدل می‌باشد و هر وقت که شما مدل را در نمایش گرافیکی ویرایش کنید به طور خودکار تغییر خواهد کرد. در واقع این بخش

شرح اجزاء گرافیکی مدل و وضعیت قرارگیری آنها را نگهداری می‌کند. توجه کنید توصیفی که در کادر محاوره‌ای به عنوان شرح مدل وارد کرده بودیم به مدل اضافه شده است. حالا فقط لازم است متغیر و معادله را به مدل اضافه کنیم. در پنجره اصلی (حالت متنی)، متغیر و معادله را وارد می-کنیم.

```
model HelloWorld "A differential equation"
	Real x(start=1);
equation
	der(x)=-x;
	¤;
end HelloWorld;
```

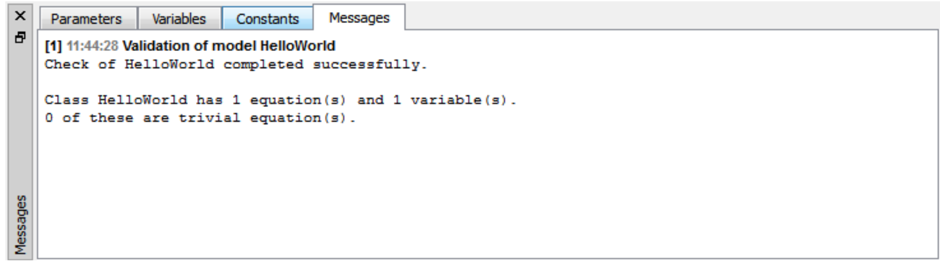
توجه کنید که وقتی متغیر را تعریف می‌کنیم مقدار اولیه آن را با استفاده از عبارت start=1 برابر یک قرار می‌دهیم، این عبارت باید بدون فاصله بعد از تعریف متغیر در پرانتز قرار گیرد. مدل Hello World آماده است.

"در زبان Modelica حروف بزرگ و کوچک با هم متفاوت هستند"

قبل از شبیه‌سازی مدل، می‌توانید صحت آن را با تلیک روی آیکون validate class در میله ابزار بررسی کنید.

شکل ۲-۴.آیکون گزینه validate class درمیله ابزار ویرایشگر مدل.

با فعال کردن validate class، گزارش وضعیت مدل شما در قسمت پایین پنجره اصلی در بخش پیام‌ها (Messages) تولید خواهد شد.



شکل ۲-۵.گزارش ارزیابی مدل.

اگر همه معادلات مدل صحیح باشد، گزارشی شبیه گزارش بالا مشاهده خواهید کرد. به عبارت completed successfully توجه نمایید. همچنین در این بخش تعداد متغیرها و معادلات این مدل نمایش داده شده است. این اطلاعات در سیستمهای پیچیده و عیب‌یابی مدلها به شما کمک فراوانی خواهد نمود.

برای شروع شبیه‌سازی نرم‌افزار Simulation Center (محیط شبیه‌سازی SystemModeler) را اجرا می‌کنیم. روی گزینه Simulation Center در میله ابزار تلیک کنید.

شکل ۶-۲.گزینه simulation center در میله ابزار ویرایشگر مدل.

شبیه‌ساز اجرا خواهد شد و مدل Hello World به طور خودکار ترجمه و جهت اجرا آماده می‌گردد. برای شبیه سازی مدل Hello World بعد از ترجمه مدل یک آزمایش تولید می‌گردد (به ترجمه قابل اجرای مدل که آماده شبیه‌سازی است آزمایش گفته می‌شود)، در محیط شبیه‌ساز در کادر سمت چپ (Experiment Browser) پارامترهای قابل تنظیم آزمایش دیده می‌شود. نمونه این کادر در شکل ۸-۲ نمایش داده شده است.

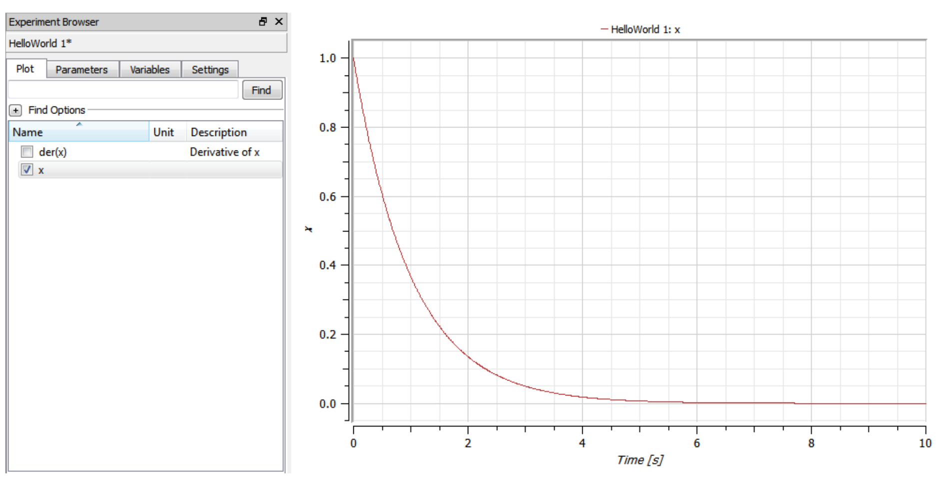
در Experiment Browser شما قادر به مشاهده و تنظیم پارامترهای شبیه‌سازی، مقادیر پارامترها و مقادیر اولیه متغیرها هستید، در این مرحله مقادیر پیش فرض را همان طور که هست رها می‌کنیم. سپس برای شروع شبیه‌سازی روی آیکون Simulation تلیک می‌کنیم یا از میانبر ctrl+R استفاده می‌کنیم.

شکل ۷-۲.آیکون simulation در میله ابزار مرکز شبیه سازی.

بعد از کامل شدن شبیه‌سازی، پنجره نمایش نمودار Experiment Browser نمایان می‌شود. برای نمایش مقدار x جعبه کوچک کنار متغیر x در کادر سمت چپ Experiment Browser را علامت بزنید. نمودار x را خواهید دید. به همین راحتی اولین معادله دیفرانسیل را که می توانست نمودار رفتار یک قطعه باشد را شبیه‌سازی کردید و پاسخ آن را برای ۱۰ ثانیه رسم نمودید.

Experiment Browser
HelloWorld 1*
Plot | Parameters | Variables | Settings
Basic
Start time: 0.0
Stop time: 10.0
Interval length:
Output intervals: Automatic
Solver: DASSL
Tolerance: 1e-6
Advanced
Options
Check min/max attributes
Stop at steady state
Synchronize with real-time
Input
Output

شکل ۸-۲.کادر تنظیمات آزمایش مدل **hello world** در مرکز شبیه سازی.



شکل ۹-۲.نمودارمتغیر **x** از مدل **hello world** در مرکز شبیه سازی.

۲-۲) برای مدل یک آیکون بسازید

حالا برای ساختن یک آیکون اختصاصی برای این مدل خودمان به نرم‌افزار ویراشگر مدل برمی-گردیم. در محیط ویرایشگر، آیکون icon view در میله ابزار را تلیک کنید یا از ctrl+1 میانبر استفاده نمایید.

شکل ۱۰-۲.گزینه icon view در میله ابزار از ویرایشگر مدل.

برای ترسیم آیکون از ابزار ترسیم موجود در میله ابزار ویرایشگر استفاده می‌شود (شکل ۱۱-۲).

شکل ۱۱-۲.ابزار های ترسیم در میله ابزار در ویرایشگر مدل.

گزینه دوم (آیکون مستطیل) را انتخاب کنید تا فعال شود. یک مستطیل رسم کنید که کل پنجره نمایش آیکون را بپوشاند. روی مستطیل ۲بار تلیک کنید تا پنجره ویژگی‌های مستطیل باز شود.

شکل ۱۲-۲.کادر مشخصات مستطیل.

در بخش خصوصیات سطح (Fill properties)، رنگ داخل مستطیل را از سفید (white) به خاکستری (grey) و طرح (pattern) داخلی آن را به solid تغییر دهید، روی کلید OK تلیک کنید. حال یک مستطیل با سطح خاکستری داریم. بعد کلید Esc روی صفحه کلید را بزنید تا همه

چیز در پنجره از انتخاب خارج گردد. در نهایت برای نوشتن متن، روی آیکون A (text tool) تلیک کنید و کل فضای مستطیل را انتخاب نمایید (در واقع محدوده نوشتن متن را مشخص کنید). پس از انتخاب محدوده نوشتن متن، کادر محاوره‌ای ویژگی‌های متن (text properties) باز می‌شود. متن Hello World را در قسمت text وارد کنید. می‌توانید هر ویژگی دیگر مربوط به متن نوشته شده را در این بخش تغییر دهید. مناسب است در خصوص لزوم عدم انتخاب هیچ چیزی قبل از اضافه کردن متن در اینجا تأکید گردد و علت آن توضیح داده شود. اگر چیزی انتخاب شده باشد، مثلاً مستطیل رسم شده، ممکن است به جای اضافه شدن متن، مستطیل را جابه‌جا نماییم. جابه‌جا کردن اجزاء در حالی که هر یک از ابزارهای ترسیم انتخاب شده است امکان‌پذیر می‌باشد.

شکل ۱۳-۲.آیکون مدل .hello world

می‌توانید آیکون مدل Hello Word را در Library browser مشاهده کنید، در هر جایی که از این مدل استفاده کرده باشید، آن را با آیکون بالا خواهید دید.

تمرین

معادله مدل را تغییر دهید به عنوان مثال یک متغیر اضافه کنید و نتیجه را بررسی نمایید.

۳-۲) معادلات دیفرانسیل

این نرم‌افزار برای شبیه‌سازی خلق شده است اما چون قوانین حاکم بر دینامیک سیستمهای فیزیکی به شکل معادلات دیفرانسیل هستند، نرم‌افزارهای شبیه‌سازی توانایی بالایی در حل معادلات دیفرانسیل دارند. در این بخش به پاره‌ای از این توانایی‌ها اشاره خواهیم نمود. اگر با معادلات دیفرانسیل آشنایی ندارید می‌توانید مطالعه این بخش را برای آینده نگه دارید و بدون این که به آموخته‌های شبیه‌سازی شما لطمه‌ای وارد شود، کتاب را ادامه بدهید.

برای مثال فرض کنید می‌خواهیم معادله دیفرانسیل درجه ۲ زیر را حل نماییم:

$$\ddot{x} - \dot{x} + x = 1 \qquad (۲-۲)$$

$$\dot{x}(0) = 5$$

نکته: برای محاسبه مشتق دوم یا بالاتر پارامترها، نمی‌توان از مشتق به صورت تو در تو استفاده نمود. در واقع der(x) را به عنوان یک تابع از x فرض نکنید، آن را معادل متغیر دیگری در نظر بگیرید که مقدارش در هر لحظه برابر مشتق اول x است. به عنوان مثال برای محاسبه مشتق دوم x ($\ddot{x}$)، نمی‌توان آن را به شکل der(der(x)) نوشت! لازم است متغیر جدیدی مانند y تعریف نمایید که برابر مشتق اول x است و از آن مشتق بگیرید. یعنی:

```
y=der(x);
der(y)-y+x=1;
```

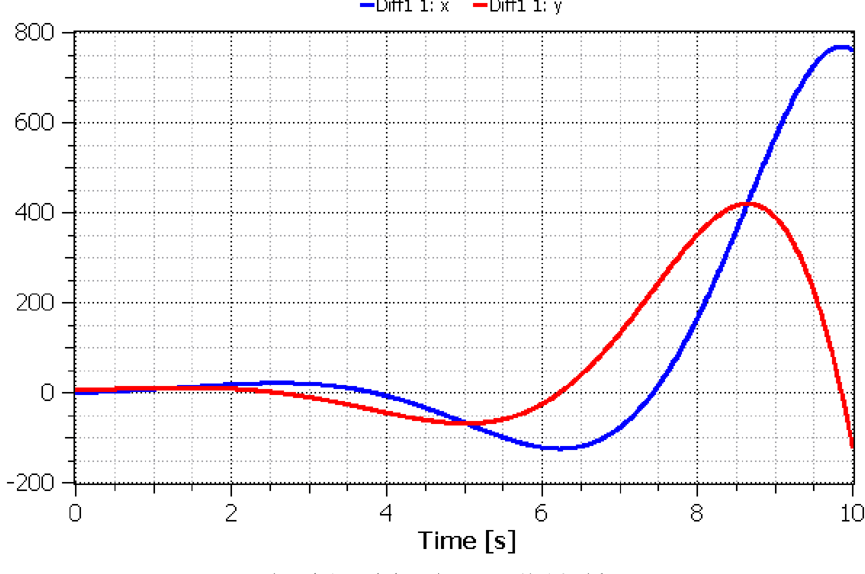
در ضمن با توجه به این که مقدار اولیه مشتق x برابر ۵ است، در هنگام تعریف متغیر y مقدار اولیه‌اش را برابر ۵ در نظر خواهیم گرفت. یعنی:

```
Real y(start=5);
```

لازم به ذکر است اگر برای متغیری مقدار اولیه تعریف نشود، به صورت خودکار مقدار اولیه آن برابر صفر قرار خواهد گرفت. معادلات لازم جهت حل معادله دیفرانسیل بالا به شکل زیر خواهد بود:

```
model Diff1  "second order differential equation"
        ¤;
        Real y(start=5),x;
equation
        y=der(x);
        der(y) - y + x=1;
end Diff1;
```

نمودار شبیه‌سازی معادله دیفرانسیل بالا را رسم نمایید.



شکل ۱۴-۲.نتیجه حل معادله دیفرانسیل.

معادله دیفرانسیل زیر را در نظر بگیرید؛ آیا می‌توانید کد لازم برای محاسبه این معادله دیفرانسیل را بنویسید؟

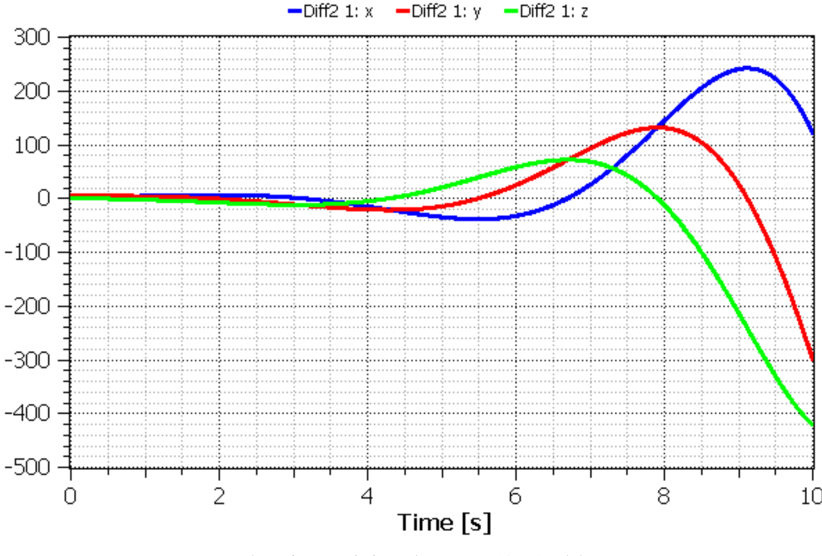
$$\ddot{x} - \sin(\dot{x}) + x = 1 \qquad (۲-۳)$$

$$\dot{x}(0) = 5$$

کد معادله بالا به شکل زیر است:

```
model Diff2  "second order non-linear differential equation"
        ¤;
        Real z,y(start=5),x;
equation
        z=der(y);
        y=der(x);
        der(z) - sin(y) + x=1;
end Diff2;
```

اگر با تلیک روی منو ابزار شبیه‌سازی Simulation Center مدل جدید به صورت خودکار در شبیه-ساز باز نشد، لازم است مدلهای پیشین را در شبیه‌ساز با استفاده از منو فایل گزینه Close ببندید و مدل جدید را با تلیک دوباره روی منو ابزار شبیه‌ساز به محیط شبیه‌سازی انتقال بدهید.



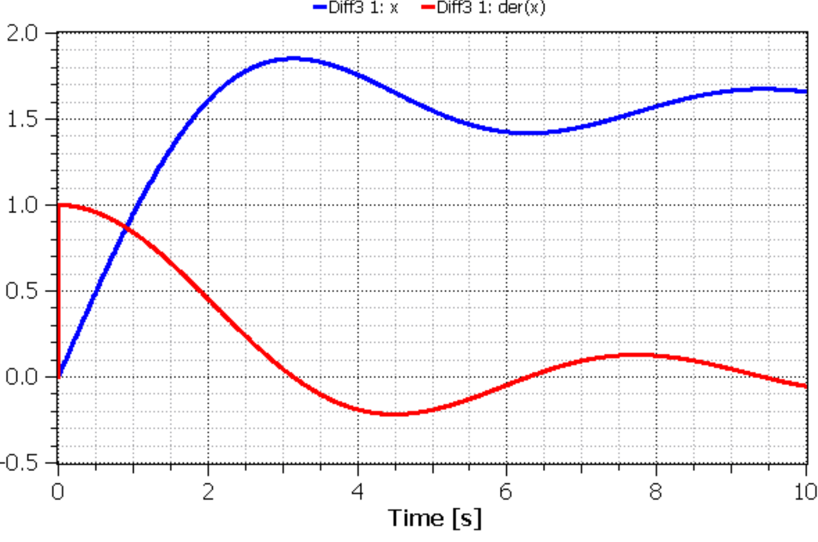
شکل ۱۵-۲.نتیجه حل معادله دیفرانسیل.

اگر در معادله دیفرانسیل، متغیر زمان نیز داده شده باشد، به جای آن از متغیر time استفاده خواهیم کرد. متغیر time در این نرم‌افزار از پیش تعریف شده می‌باشد بنابراین تغییر یا تعریف آن مجاز نمی‌باشد. برای مثال معادله دیفرانسیل زیر را در نظر بگیرید:

$$\dot{x} = \frac{\sin(t)}{t} \qquad (۲-۴)$$

برنامه حل این معادله به صورت زیر خواهد بود، به مخرج کسر سمت راست توجه نمایید، یک عدد بسیار کوچک به مخرج اضافه شده است، این عدد برای جلوگیری از ایجاد نقطه تکین تعریف نشده در زمان صفر در نظر گرفته شده است. هنگام شروع شبیه‌سازی مقدار متغیر time برابر صفر خواهد بود (مگر آنکه شما زمان شروع شبیه‌سازی را تغییر بدهید) بنابراین با توجه به این که تقسیم بر صفر تعریف نشده است، یک عدد بسیار کوچک (نسبت به حد بالای متغیر time) را به مخرج اضافه نموده‌ایم تا این مشکل حل گردد. استفاده از این عدد فقط برای رفع مشکل تقسیم بر صفر می‌باشد و تأثیر مشهودی در پاسخ معادله ایجاد نخواهد کرد. شما همچنین می‌توانستید زمان شروع شبیه‌سازی را به عددی بزرگتر از صفر تغییر بدهید تا این مشکل ایجاد نگردد. مثلاً زمان شروع شبیه‌سازی را برابر 0.0001 قرار دهید تا ضمن حل معادله و رفع مشکل تقسیم بر صفر جواب معقولی داشته باشیم.

```
model Diff3 "second order non-linear differential equation"
    ¤;
    Real x;
equation
    der(x) = sin(time)/(time + 1e-20);
end Diff3;
```



شکل ۱۶-۲.نتیجه حل معادله دیفرانسیل زمانی

فصل ۳ سیستمهای فیزیکی مختلف به صورت توأم

در این فصل روش پیاده‌سازی مدل یک موتور الکتریکی در SystemModeler به صورت قدم به قدم نشان داده خواهد شد و توانایی شبیه‌سازی محیط‌های مختلف فیزیکی به صورت توأم مورد بررسی قرار خواهد گرفت. همچنین خواهیم آموخت با استفاده از محیط شبیه‌سازی مدل‌هایی را که در محیط ویرایشگر ایجاد نموده‌اید را تنظیم کرده و از دیدگاه متغیرهای مختلف بررسی کنید.

۳-۱) موتورDC

موتور الکتریکی شامل قطعات الکتریکی و مکانیکی است. یک مدل دینامیکی ساده موتور DC قابل کنترل نیاز به اجزاء زیر دارد:

۱- منبع ولتاژ متغیر ۲- مقاومت الکتریکی ۳- القاگر ۴- المان نیروی محرکه الکتریکی که ارتباط بین انرژی الکتریکی و مکانیکی تولید شده به وسیله میدان مغناطیسی در موتور DC را برقرار می-سازد. اثر محور موتور با یک جرم چرخان یا اینرسی نمایش داده می‌شود.

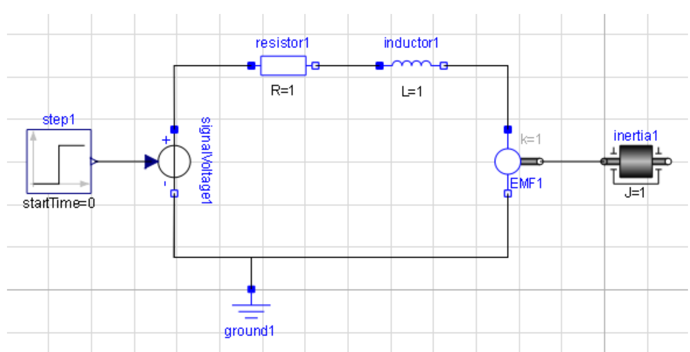
مدل همه اجزاء لازم برای شبیه‌سازی این موتور و همچنین بسیاری از تجهیزات دیگر در کتابخانه modelica وجود دارد و نیازی به معادله نویسی مجدد ندارد. برای ایجاد این مدل فقط کافی است که این اجزاء را در کتابخانه پیدا کنیم، آنها را به محیط مدل خود بکشیم، اتصالات بین آنها را برقرارکنیم و پارامترهای هر کدام را تنظیم کنیم.

تمام این اجزاء را می‌توانید در کتابخانه استاندارد modelica موجود در نرم‌افزار SystemModeler بیابید. جستجو در قسمت find (کادر سمت چپ بالا) در Library Browser می‌تواند در پیدا کردن اجزاء مختلف به شما کمک زیادی بکند. با نگه داشتن و کشیدن این اجزاء از کتابخانه استاندارد modelica به ویرایشگر مدل می‌توان مدل موتور DC را مطابق زیر ایجاد نمود.

برای ساختن مدل شکل ۱-۳، باید مدل جدیدی ایجاد کنید. اجزاء به کار رفته در این مدل را از Library Browser پیدا کرده و آنها را به پنجره ویرایشگر مدل بیاورید و در نهایت، اجزاء را با استفاده از ابزار اتصالconnection line tool به یکدیگر متصل نمایید.

با خلق یک مدل جدید به نام DCMotor شروع می‌کنیم.

File >> New class



شکل ۱-۳. نمایش گرافیکی موتور DC در ویرایشگر مدل.

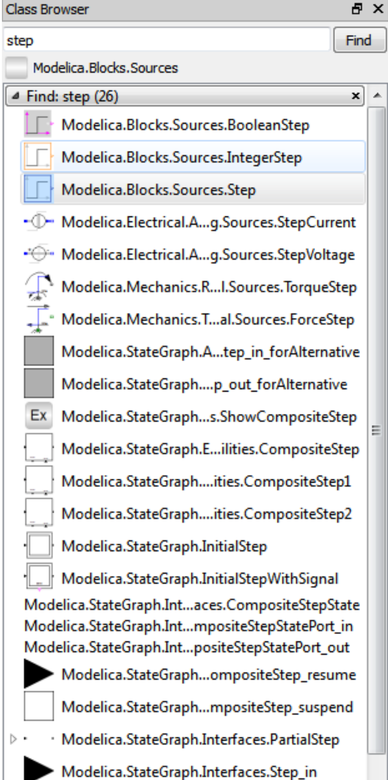
در قسمت description می‌توانید هر چیزی را قرار دهید (مثلاً DC motor dynamic model). تمام عضوهای مورد نیاز برای ساختن این مدل در Library Browser موجود است. گام دوم پیدا کردن عضوهای مورد نیاز در Library Browser و انتقال آنها به پنجره ویرایشگر مدل می‌باشد. هر عضو را می‌توان با ۲ روش پیدا کرد، ما هر دو روش را توضیح خواهیم داد.

برای یافتن منبع پله‌ای، از جستجوگر کتابخانه استفاده خواهیم کرد. واژه step را در جعبه متنی بالای جستجوگر کتابخانه، سمت چپ کلید Find بنویسید و کلید Entar را بزنید یا کلید Find را تلیک کنید. اگر همه چیز خوب پیش برود، شما حداقل با ۲۳ مورد مطابقت اجزاء با واژه step رو به رو خواهید شد. عضو مورد نظر ما Modelica.Blocks.Sources.Step می‌باشد، که در زیر با رنگ آبی انتخاب شده است. برای افزودن این قطعه به مدل موتور DC، آن را گرفته و به پنجره ویرایشگر بکشید.

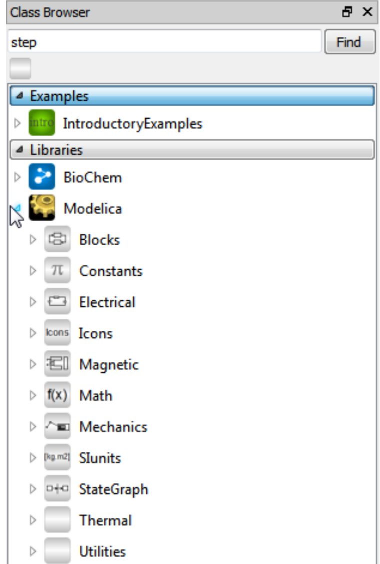
قطعه سیگنال ولتاژ در کتابخانه Modelica.Electrical.Analog.Sources قرار دارد. از آنجا که ما محل دقیق قطعه مورد نظر خود را می‌دانیم، از نمایش درختی کاوشگر کتابخانه، با بازکردن مسیر شاخه‌ها تا رسیدن به قطعه مورد نظر استفاده می‌کنیم.

عضو Signal Voltage در مسیر زیر قرار دارد، پس مسیر زیر را باز می‌کنیم:

Modelica>>Electrical>>Analog>>Sources

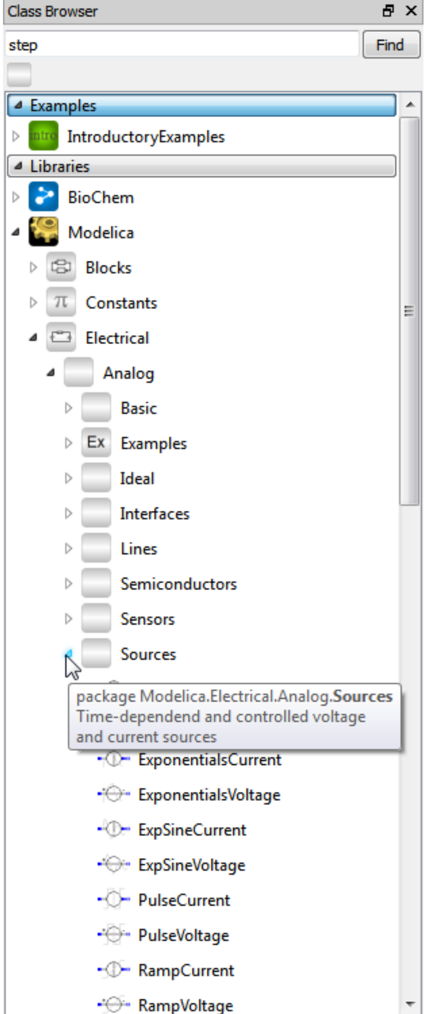


شکل ۲-۳.نتیجه جستجو برای step با استفاده از روش اول.



شکل ۳-۳.باز شدن کتابخانه Modelica در library browser.

همانطور که مشاهده می‌کنید درون کتابخانه modelica تعداد زیادی کتابخانه دیگر وجود دارد. طبق مسیر ذکر شده روی کتابخانه Electrical ۲بار تلیک می‌کنیم، دوباره چندین کتابخانه باز می‌شود که طبق مسیر، کتابخانه Analog و سپس کتابخانه Sources را باز می‌کنیم. پس از باز کردن آخرین کتابخانه، مدل Signal Voltage نمایان می‌شود.

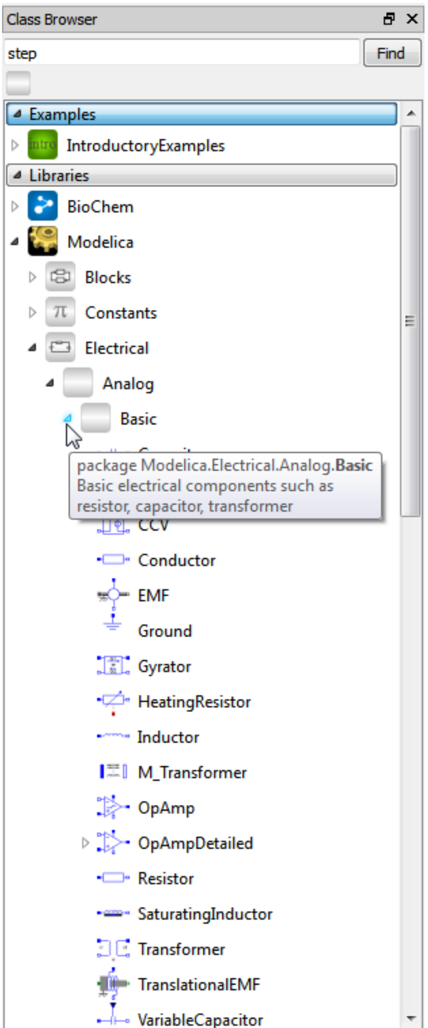


شکل ۴-۳.مسیر Modelica >>Electrical >>Analog >> Sources.

عضو Signal Voltage را به ویرایشگر مدل انتقال دهید تا به DCMotor اضافه شود. تا به حال ۲ جزء از اجزاء لازم را به مدل اضافه کرده‌ایم. سایر اجزاء الکتریکی (Resistor,Inductor,ground وEMF) را می‌توانید در مسیر زیر بیابید:

Modelica >> Electrical >> Analog >> Basic

با توجه به این که قبلاً مسیر کتابخانه‌ها را برای یافتن Signal Voltage باز کرده‌ایم، کافی است کتابخانه Basic را باز کنیم و همه عضوهای Resistor,Inductor,ground وEMF نمایان می-شود.



شکل ۵-۳.کتابخانه های مسیر Modelica >>Electrical >>Analog >> Basics.

وقتی که تمام اجزاء الکتریکی را به مدل DCMotor اضافه کردیم، فقط یک جزء باقی می‌ماند، قطعه اینرسی که لازم است به مدل اضافه شود. این عضو را در مسیر زیر بیابید.

Modelica >> Mechanics >> Rotational

این عضو را می‌توان با جستجو کردن در قسمت Find پیدا کرد یا مستقیماً از نمودار درختی کتابخانه‌ها مسیر مورد نظر را باز کرده تا به عضو inertia برسید.

پس از اضافه کردن جزء اینرسی مدل موتور DC کامل شده است، تنها چیزی که باقی می‌ماند اتصال اجزاء به یکدیگر است. می‌توانید در صورت نیاز اجزاء را چرخانده یا جابه‌جا نمایید.

برای چرخاندن اجزاء با استفاده از موشی روی عضو مورد نظر تلیک راست کنید، در کادری که باز می‌شود در قسمت Shape چهار گزینه زیر وجود دارد، با استفاده از این گزینه‌ها می‌توانید عضو را در هر جهتی که می‌خواهید بچرخانید.

۱. چرخاندن به سمت چپ Rotate Left
۲. چرخاندن به سمت راست Rotate Right
۳. معکوس کردن نسبت به محورعمودی عضو Flip horizontal
۴. معکوس کردن نسبت به محورافقی عضو Flip vertical

برای مثال روی عضو Signal Voltage تلیک راست کنید در قسمت Shape ابتدا گزینه اول را تلیک کنید دوباره این عمل را تکرار کنید ولی این بار گزینه چهارم را انتخاب کنید. مشاهده خواهید کرد که عضو مورد نظر در جهت نشان داده شده مطابق شکل قرار می‌گیرد.

برای جابه‌جا کردن عضو کافی است روی عضو مورد نظر تلیک چپ کرده و نگهدارید. سپس در عین حال که کلید چپ موشی را نگهداشته‌اید، موشی را حرکت دهید، مشاهده می‌کنید که عضو مورد نظر جابه‌جا می‌شود و در مکانی که کلید موشی را رها می‌کنید قرار می‌گیرد.

برای متصل کردن اجزاء باید از ابزار اتصال (connection line tool) استفاده کنید، این ابزار در نوار ابزار قرار دارد.

شکل ۶-۳.ابزار اتصال در میله ابزار در ویرایشگر مدل.

به عنوان مثال برای متصل کردن زمین مدار الکتریکی (ground) به قطب منفی ولتاژ (signal voltage)، ابتدا روی ابزار اتصال (آیکون) تلیک کنید تا فعال شود، مکان‌نمای موشی را روی پایه زمین مدار قرار دهید و تلیک کنید سپس مکان‌نما را روی قطب منفی ولتاژ ببرید، آیکون مکان‌نما تغییر می‌کند، روی آن تلیک کنید، مشاهده می‌کنید که دو عضو به یکدیگر متصل شدند. به همین ترتیب سایر اجزاء را به یکدیگر متصل کنید تا مدل موتور DC مشابه شکل ۱-۳ بشود.

در حالی که شما عضوها را به مدل اضافه و آنها را به یکدیگر متصل می‌کنید، ویرایشگر مدل کدهای modelica معادل مدل‌ها و اتصالات مربوط به آن را تولید می‌کند. برای دیدن ساختار متنی

مدل تولید شده توسط modelica می‌توانید روی گزینه modelica text view در میله ابزار (آیکون) تلیک کنید و کدهای تولید شده را مشاهده کنید.

در این کدها، هر جزء تعریف گردیده و اتصال بین اجزاء در بخش معادلات نمایش داده می‌شود. کد تولید شده توسط ویرایشگر مدل را در این قسمت مشاهده می‌کنید.

قسمت تفسیر ¤ (annotation)، در خصوص مکان و قرارگیری اعضاء، مسیرشان و همچنین مقدار چرخش و هر آنچه که مربوط به خصوصیات عضو در صفحه نمایش است را مشخص کرده است. برای مثال می‌توانید مقدار چرخش ۲۷۰ درجه را برای عضو Signal Voltage مشاهده کنید. برای مشاهده این تفسیرها روی صفحه نمایش متنی کلیک راست کرده و گزینه Expand Annotations را انتخاب نمایید. برای پنهان کردن خطوط تفسیر از کلیک راست و انتخاب گزینه Collapse Annotations استفاده نمایید.

در قسمت معادله (equation) هر یک از اتصالات بین دو عضو با یک معادله اتصال نشان داده می‌شود.

```
model DCMotor
    Modelica.Blocks.Sources.Step step ¤;
    Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1 ¤;
    Modelica.Electrical.Analog.Basic.Resistor resistor1 ¤;
    Modelica.Electrical.Analog.Basic.Inductor inductor1 ¤;
    Modelica.Electrical.Analog.Basic.EMF EMF1 ¤;
    Modelica.Mechanics.Rotational.Inertia inertia1 ¤;
    Modelica.Electrical.Analog.Basic.Ground ground1 ¤;

equation
    connect(EMF1.flange_b,inertia1.flange_a) ¤;
    connect(EMF1.n,signalVoltage1.n) ¤;
    connect(signalVoltage1.n,ground1.p) ¤;
    connect(inductor1.n,EMF1.p) ¤;
    connect(resistor1.n,inductor1.p) ¤;
    connect(signalVoltage1.p,resistor1.p) ¤;
    connect(step1.y,signalVoltage1.v) ¤;
    ¤;
end DCMotor;
```

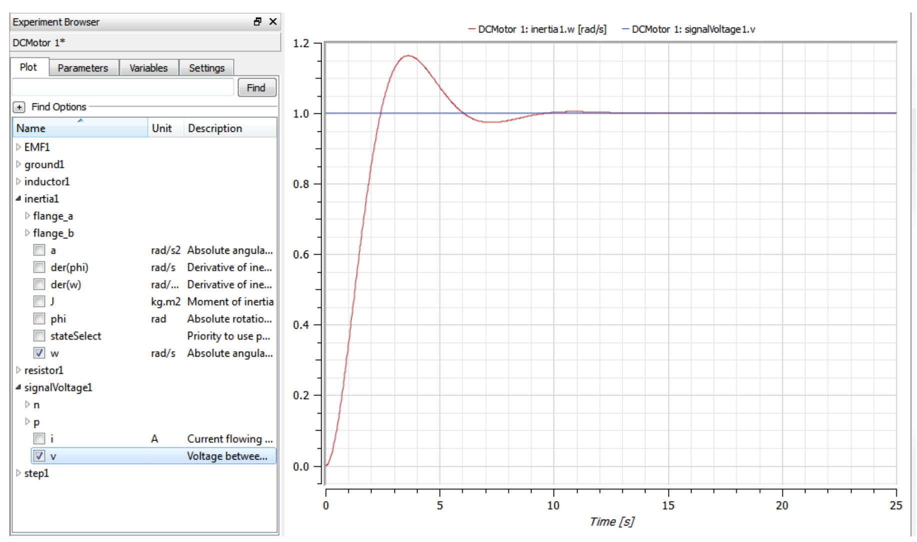
ترتیب تعریف اجزاء و معادلات، به ترتیب اضافه شدن آنها به مدل بستگی دارد. بنابراین دستورالعمل‌ها و معادلات داده شده ممکن است که با آنچه در نمونه بالا نشان داده شده اندکی تفاوت داشته باشد. در نوشته‌های این کتاب برای خوانایی بیشتر همه مشخصات گرافیکی مربوط به مشخصات مکان، چرخش و سایر مشخصات گرافیکی از قسمت تعریفات مدل DCMotor حذف شده است.

حال مدل موتور DC کامل شده و برای شبیه‌سازی آماده است. روی گزینه محیط شبیه‌سازی در ویرایشگر مدل تلیک کنید تا مدل ترجمه شده و آماده شبیه‌سازی گردد. در بخش تنظیمات محیط شبیه‌سازی، زمان شبیه‌سازی را به ۲۵ ثانیه تغییر دهید، برای این کار در کادر time setting از تنظیمات DCMotor experiment در قسمت stop time عدد ۲۵ را وارد کنید.

DCMotor 1*
Plot | Parameters | Variables | Settings
Basic
Start time: 0.0
Stop time: 25
Interval length:
Output intervals: Automatic
Solver: DASSL
Tolerance: 1e-6

شکل ۷-۳. تغییرزمان شبیه سازی به ۲۵ ثانیه.

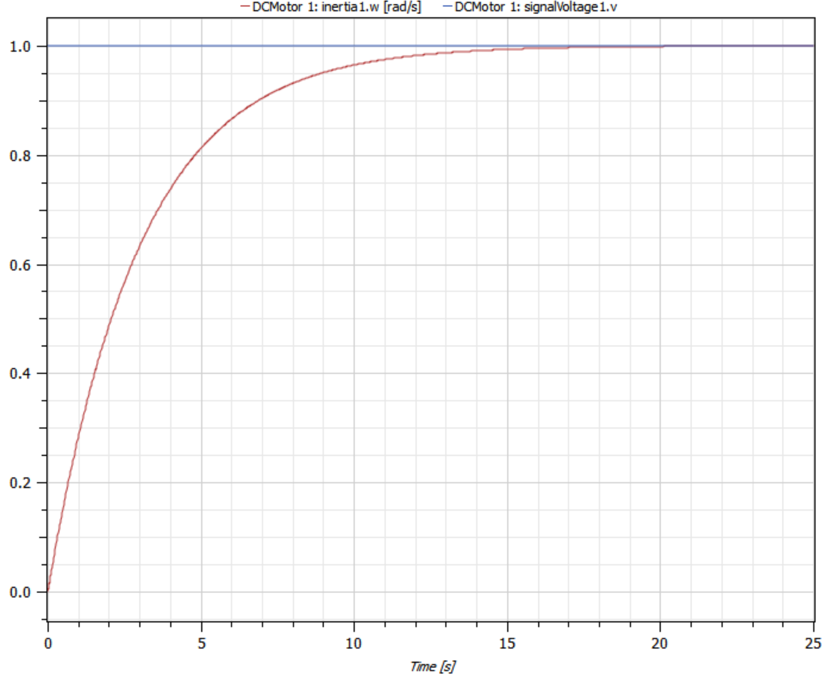
روی گزینه شبیه‌سازی (آیکون) تلیک کنید تا شبیه‌سازی آغاز گردد. پس از کامل شدن شبیه‌سازی متغیرهایی را همانند شکل زیر انتخاب کنید تا نمودار آنها در experiment browser رسم شود.



شکل ۸-۳. نموداراینرسی (متغیر W)و ولتاژ (متغیر V) برای مدل DCMotor با مقادیرپارامتری پیش فرض.

در نهایت، نتایج را خواهید دید. نمودار سرعت دورانی جزء اینرسی و ولتاژ منبع نسبت به زمان رسم شده است. تغییر مقادیر پارامترها برای اصلاح رفتار سیستم کار ساده‌ای است. ما مقدار مقاومت الکتریکی، مقدار ضریب خودالقایی القاگر و مقدار اینرسی را تغییر خواهیم داد تا به جای حالت نوسانی، یک پاسخ پایدارتر و میرا شونده داشته باشیم.

برای تنظیم مقادیر متغیرها در کاوشگر آزمایش به بخش نمایش متغیرها بروید. برای تغییر مقدار یک متغیر روی مقدار فعلی آن ۲بار تلیک کنید. مقدار مقاومت الکتریکی را برابر ۱۰ اهم (10 Ohm)، القاگر را برابر ۰.۱ هانری (0.1 H) و جرم جزء اینرسی را ۰.۳ کیلوگرم (0.3 Kg) قراردهید. مدل را دوباره شبیه‌سازی کنید و تغییرات نتایج سرعت دورانی اینرسی را بررسی نمایید.



شکل ۹-۳.نمودار گشتاور اینرسی و ولتاژ برای مدل DCMotor با مقادیر پارامتری داده شده.

۳-۲) محور صلب و محور انعطاف پذیر

در این بخش با مدل کردن محور صلب شروع می‌کنیم، پاسخ آنرا نسبت به تابع پله با اضافه کردن یک گشتاور پله‌ای بررسی می‌کنیم و نشان خواهیم داد که چگونه می‌توانیم با افزودن انعطاف-پذیری به محور صلب آن را با دقت بیشتری مدل کرد.

با مدلسازی محور صلب شروع می‌کنیم. یک مدل جدید بسازید. اسمی دلخواه برای مدل در نظر بگیرید. اجزاء مدل Step, Torque, Inertia, and IdealGear را می‌توانید در مسیرهای زیر بیابید یا آنها را در قسمت Find جستجو کنید:

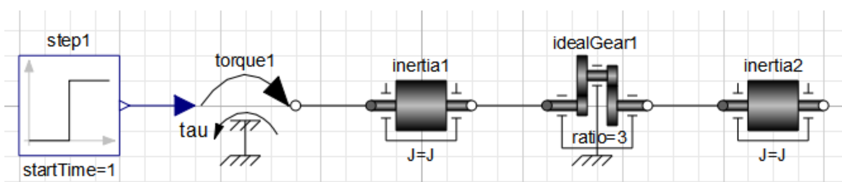
Modelica >> Blocks >>
Sources
Modelica >> Mechanics >> Rotational

اجزاء را به مدل کشیده و اتصالات بین اجزاء را مطابق شکل زیر برقرار کنید.

برای این مثال مدل آماده شده از قبل در بخش مثالهای نمونه، از مسیر زیر در دسترس است:

IntroductoryExamples >> MultiDomain

همه مدلهای این کتاب در کتابخانه مثالهای مقدماتی وجود دارد ولی توجه کنید همه آنها به صورت فقط خواندنی هستند و نمی‌توانید آنها را تغییر دهید. بنابراین دلیل خوبی برای ایجاد مدل از ابتدا دارید. مخصوصاً اگر قصد داشته باشید متغیرهای مدل خود را تغییر دهید لازم است مدلسازی را از ابتدا انجام دهید. با انجام مدلسازی‌ها قدم به قدم با نرم افزار آشنا شده و یادگیری شما عمیق‌تر خواهد شد.



شکل ۱۰-۳. نمایش گرافیکی IntroductoryExampeles>>MultiDomain>>StiffAxis model.

با انتخاب عضو idealGear1 می‌توانیم متغیرهای آن را نیز ویرایش کنیم، متغیرهای این عضو در کادر پایین، در قسمت پارامترها (Parameters) در دسترس و قابل ویرایش می‌باشد.

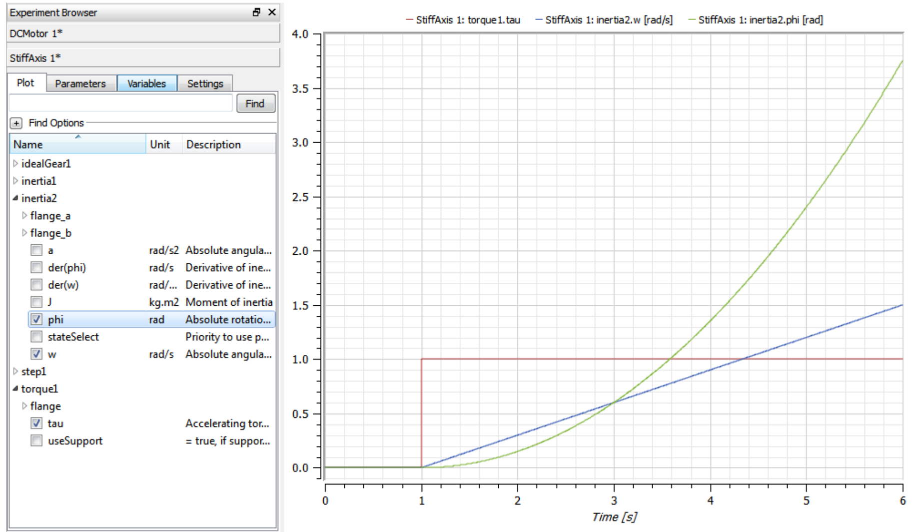
Parameters | Variables | Constants | Messages

Name	Value	Description
ratio	3	Transmission ratio (flange_a.phi/flange_b.phi)
useSupport	false	= true, if support flange enabled, otherwise implicitly grounded

شکل ۱۱-۳. ویرایش مقدار پارامتر نسبت انتقال، چرخدنده ایده ال.

نسبت چرخدنده‌ها را برابر ۳ قرار دهید. این مقدار به این معنی است که زاویه‌ها و سرعت زاویه‌ای ۳ برابر افزایش می‌یابد و گشتاور به نسبت ۳ از یک طرف چرخدنده‌ها به طرف دیگر کاهش می‌یابد.
همچنین زمان شروع پله را با تغییر مقدار پارامتر startTime به ۱ تغییر دهید.

بعد از شبیه‌سازی سیستم به مدت ۶ ثانیه، مشاهده می‌کنید که گشتاور ثابت باعث شتاب زاویه‌ای ثابت می‌گردد. در نتیجه سرعت زاویه‌ای به شکل شیب ثابت و زاویه محور، منحنی درجه دو می‌باشد (شکل ۱۲-۳).

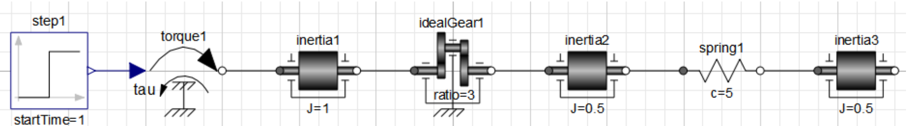


شکل ۱۲-۳. نمودارگشتاور، زاویه دورانی و سرعت زاویه‌ای برای اینرسی۲.

با افزودن انعطاف‌پذیری به محور می‌توان رفتار محور را به رفتار واقعی نزدیکتر کرد. در این صورت گفته می‌شود مدل دقت بیشتری دارد. واژه انعطاف‌پذیر در مقابل واژه صلب انتخاب شده است و منظور محوری است که مانند محور واقعی می‌تواند مقدار کمی پیچش را تحمل کند و در مقابل این پیچش حالت فنری از خود نشان بدهد. این کار با جایگزین کردن مدلی متشکل از دو جرم چرخشی که با یک فنرپیچشی به هم متصل شده‌اند، امکان پذیر می‌شود (شکل ۱۳-۳).

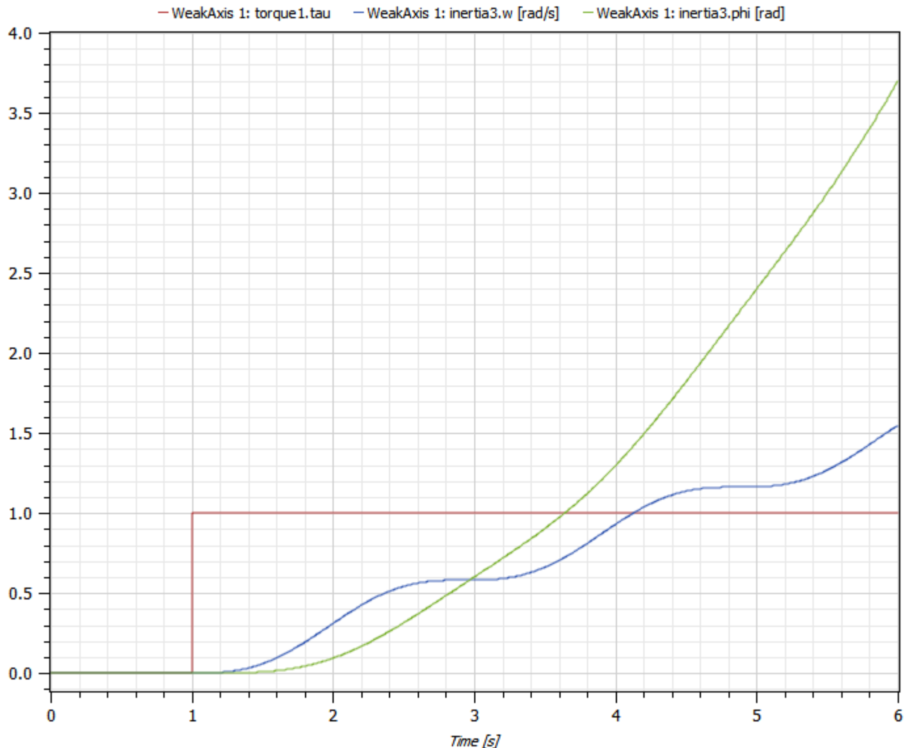
فنر پیچشی درمسیرکتابخانه‌های زیر قرار دارد:

Modelica>>Mechanics>>rotational



شکل ۱۳-۳. نمایش گرافیکی IntroductoryExampeles>>MultiDomain>> WeakAxis Model.

توجه داشته باشید که گشتاور اینرسی ۱و ۲ $0.5\mathrm{Kgm}^2$ می‌باشد (توجه فرمایید مقدار جرم اینرسی نصف شده است) و ثابت فنر شماره ۱ برابر 0.5Nm/rad تنظیم شده است. این سیستم را برای ۶ ثانیه شبیه‌سازی کرده و نتایج را بررسی می‌کنیم. مقایسه نتایج با مدل محور صلب نشان دهنده عملکرد مشابهی است، فقط یک انحنای اضافی در رفتار دیده می‌شود. توجه کنید که گشتاور اینرسی۳ (inertia3) آخرین عضو از محور می‌باشد و inertia2 آخرین عضو نیست، بنابراین سرعت چرخشی و زاویه‌ای عضوinertia3 را رسم می‌کنیم.



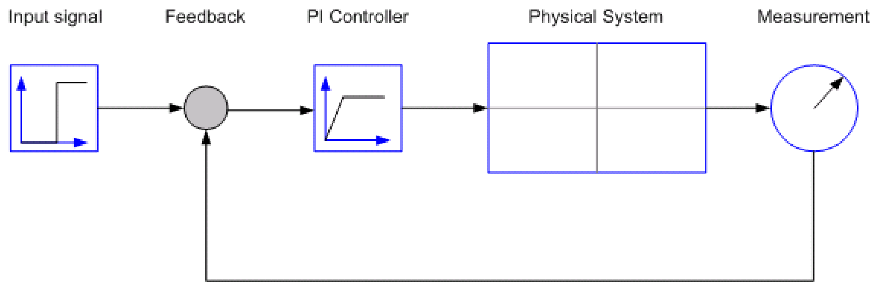
شکل ۱۴-۳. نمودار گشتاور، زاویه جزء اینرسی ۳ و سرعت زاویه ای آن.

تمرین

یک موتور DC ساده با یک فنر پیچشی در محور بیرونی و یک المان اینرسی دیگر بسازید. آن را شبیه‌سازی کرده و نتایج را بررسی کنید. پارامترها را تغییر داده و نتایج را مقایسه کنید. همچنین می‌توانید یک گشتاور به ورودی اضافه کرده و آن را به inertia2 متصل کرده و دوباره سیستم را شبیه‌سازی و تحلیل کنید.

۳-۳) سیستم کنترل

این بخش را با ایجاد یک مکانیزم سروو صلب و سست به پایان می‌رسانیم، این مکانیزم را با استفاده از مدل DCMotor و مدل محورهایی که در بخش گذشته توضیح داده شد، خواهیم ساخت.
ساختار سیستم کنترل در شکل زیر به صورت شماتیک نشان داده شده است. این سیستم از یک سیگنال ورودی، یک حسگر، یک حلقه بازخور[1] و یک کنترل‌کننده تشکیل شده است. سیستم فیزیکی از مدل DCMotor و یک محور تشکیل شده است.



شکل ۱۵-۳. نمایش ساده شده سیستم کنترل.

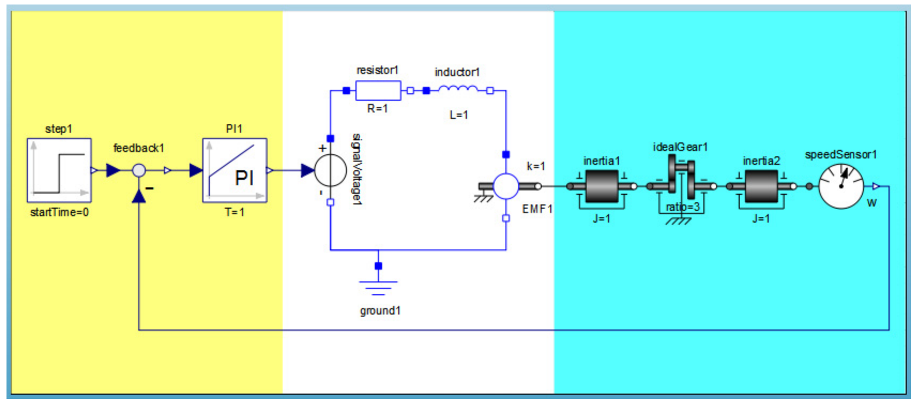
همه بلوک‌ها را همانند شکل بالا به یکدیگر متصل می‌کنیم. انتخاب پیش فرض پارامترهای تنظیم کننده k=1 و T=1 می‌باشد، تابع تبدیل کنترل‌کننده PI برابر است با:

$$G_{PI} = kTs + \frac{1}{Ts} \qquad (۳-۱)$$

با ایجاد سیستم کنترل برای DCMotor و محور صلب که قبلاً ایجاد شدند شروع می‌کنیم.
همانطور که در شکل زیر مشاهده می‌کنید ۳ عضو جدید معرفی شده است: جزء Feedback، کنترل-کننده PI و حسگر سرعت.

---

[1] Feedback

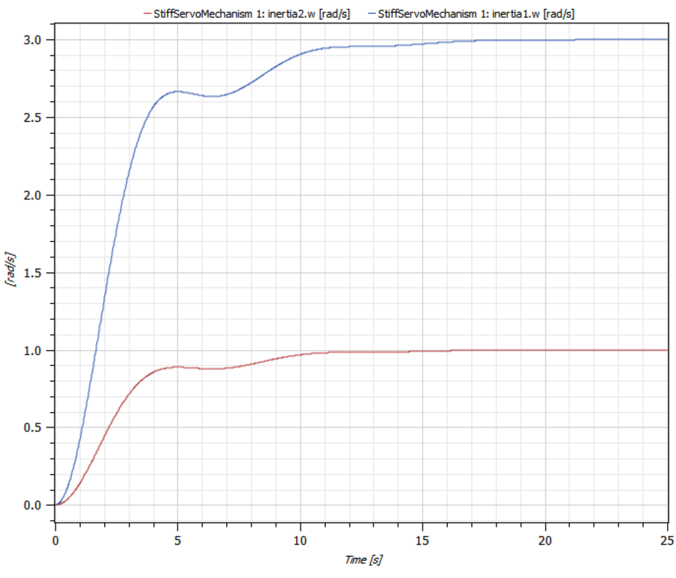


شکل ۱۶-۳. دیاگرام IntroductoryExampeles.MultiDomain.StiffServoMechanism.

این عضوها را می‌توانید در مسیرهای زیر پیدا کنید:

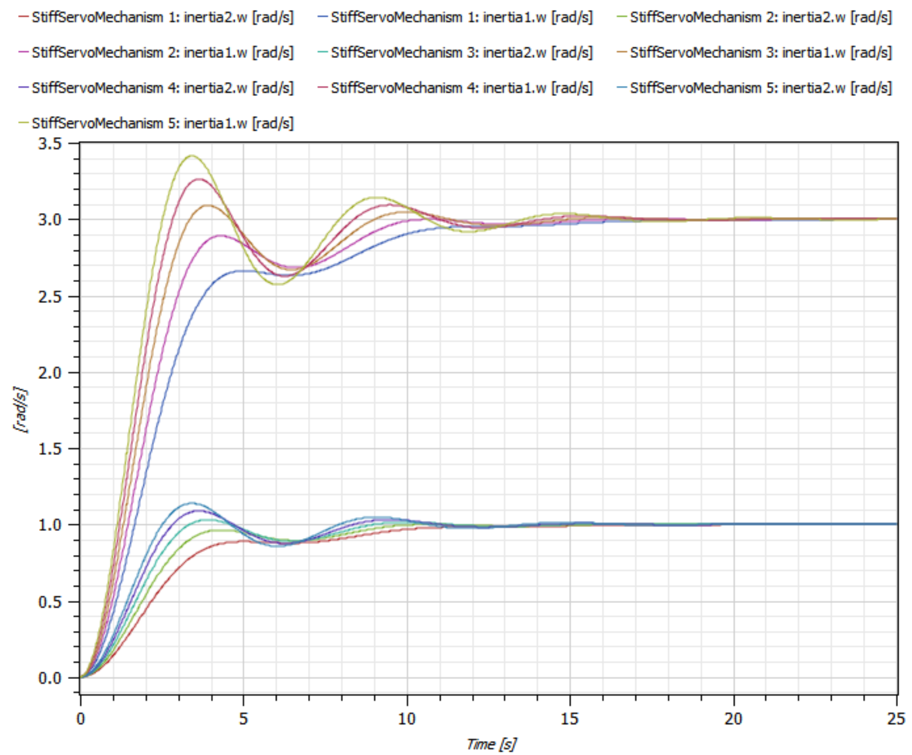
PI controlle → Modelica>>Blocks>>Continuo
Feedback → Modelica>>Blocks>>Math
Speed sensor → Modelica>>Mechanics>>Rotational>>Sensors

وقتی این مدل را شبیه‌سازی می‌کنیم، توجه خود را به پاسخ سرعت زاویه‌ای محور موتور و محور چرخدنده متمرکز کنید. همانطور که در شکل زیر مشاهده می‌کنید مدل برای ۲۵ ثانیه شبیه‌سازی شده است.



شکل ۱۷-۳. نمودارسرعت زاویه‌ایی اینرسی ۱ و اینرسی ۲
IntroductoryExamples.MultiDomain.StiffServoMechanism model

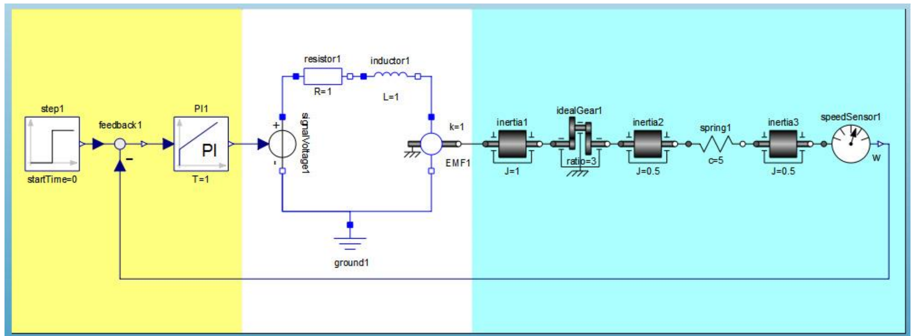
تا به حال برای کنترل‌کننده از مقادیر پیش‌فرض پارامترها استفاده می‌کردیم. با تغییر بهره K کنترل‌کننده، می‌توان پاسخ آن را بررسی کرد. در این مثال، بهره K را از ۱ تا ۲ به فاصله ۰.۲۵ تغییر می‌دهیم و نتایج را بررسی خواهیم نمود. با خلق یک آزمایش جدید برای هر شبیه‌سازی و رسم نمودار نتایج آنها در یک صفحه آنها را مقایسه می‌کنیم. آزمایشهای جدید با انتخاب گزینه New از منو فایل در شبیه‌ساز ساخته می‌شوند. برای هر آزمایش پارامترهای مناسب را تنظیم کرده و شبیه-سازی را انجام داده و نمودار نتایج را رسم می‌کنیم.



شکل ۱۸-۳. نمودارسرعت زاویه‌ای از اینرسی ۱ و اینرسی ۲
IntroductoryExamples.MultiDomain.StiffServoMechanism model

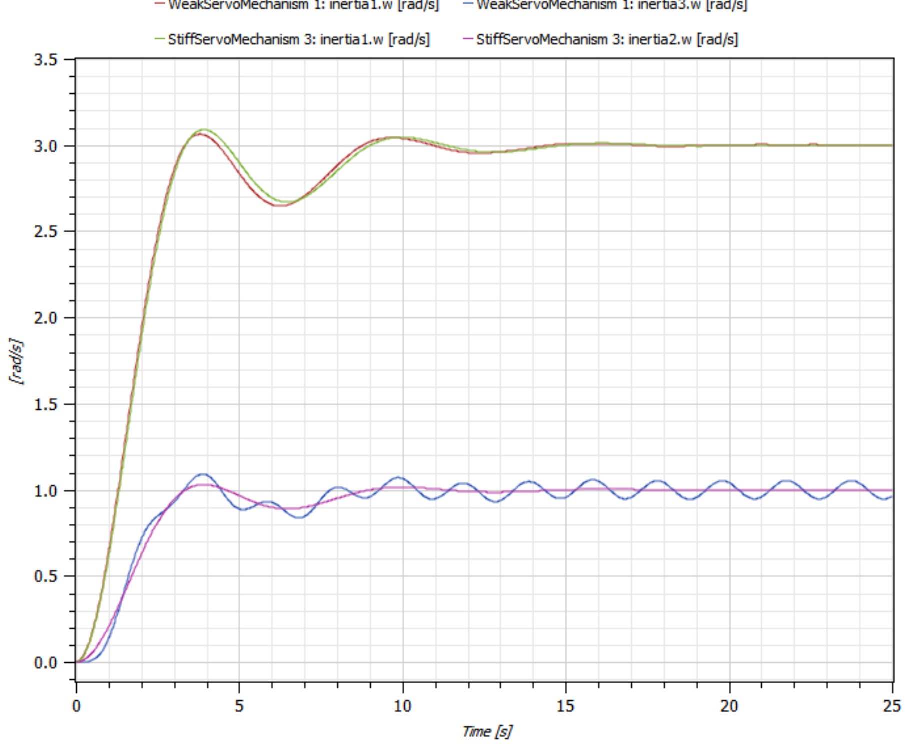
با بررسی و تحلیل پاسخ سرعت زاویه‌ای موتور و محورهای چرخدنده با استفاده از بهره‌های متفاوت کنترل‌کننده به این نتیجه می‌رسیم که با انتخاب k=1.5 پاسخ سریع با نوسانات کم خواهیم داشت.

در نهایت برای مقایسه، یک سیستم کنترل برای سیستم DCMotor و محور انعطاف‌پذیر خلق می‌کنیم.



شکل ۱۹-۳. دیاگرام IntroductoryExamples.MultiDomain.WeakServoMechanism.

قبل از شبیه‌سازی، بهره کنترل‌کننده را به k=1.5 تنظیم کنید و نتایج را با نتایج سیستم محور صلب مقایسه کنید.



شکل ۲۰-۳. مقایسه اینرسی بین مدل StiffServoMechanismو WeakServoMechanism با استفاده از بهره k=1.5.

همانطور که می‌بینید، طراحی کنترل‌کننده بر اساس مدل محور صلب به خوبی برای محور انعطاف‌پذیر نیز پاسخگو می‌باشد.

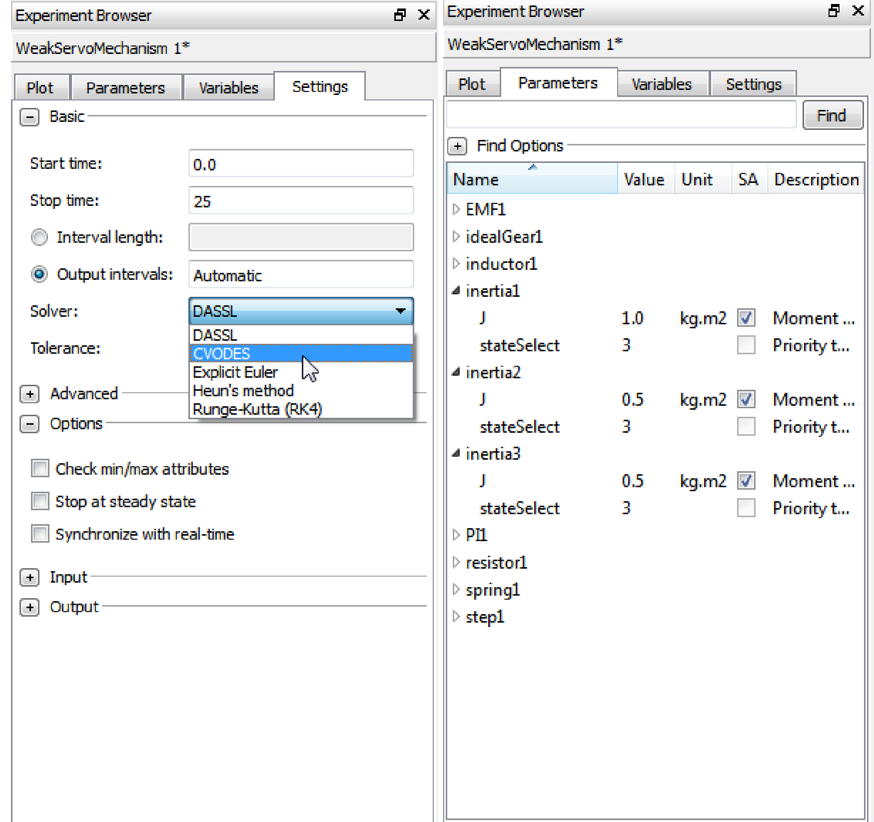
۳-۴) تحلیل میزان حساسیت

در این بخش به بررسی حساسیت کنترل‌کننده نسبت به تغییرات پارامترهای مختلف سیستم خواهیم پرداخت. این کار را با استفاده از حل‌کننده CVODES انجام می‌دهیم. این حل کننده تحلیل‌های حساسیت را پشتیبانی می‌کند. حساسیت $s_i(t)$ برای تغییرات متغیر حالت $y_i(t)$ نسبت به پارامتر p با رابطه زیر داده می‌شود:

$$s_i(t) = \frac{\partial y_i(t)}{\partial p} \quad (۳-۲)$$

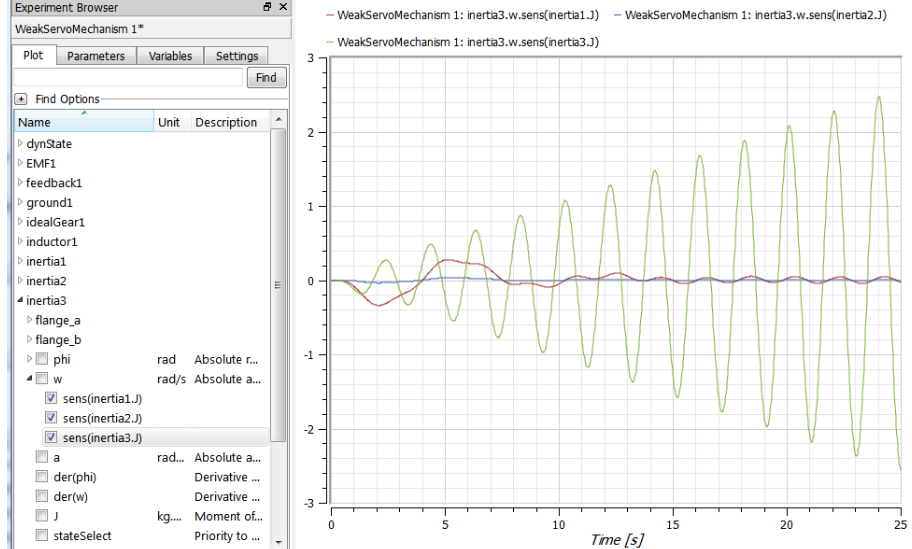
به عبارت دیگر در هر لحظه، حساسیت نشان می‌دهد که پاسخ $y_i(t)$ به ازاء تغییرات کوچک پارامتر p چقدر خواهد بود.

حال اجازه دهید حساسیت کنترل کننده طراحی شده را نسبت به تغییر ۳ اینرسی موجود بررسی کنیم. برای این کار در قسمت نمایش settings درکادر experiment، حل‌کننده CVODES را انتخاب کنید و در بخش نمایش پارامترها، مربع SA را برای اجزاء اینرسی۱.j و اینرسی۲.j و اینرسی۳.j انتخاب کنید.



شکل ۲۱-۳. انتخاب حل‌کننده CVODES و انتخاب inertia1.J, inertia2.J and inertia3.J برای تحلیل میزان حساسیت.

وقتی شبیه‌سازی تمام شد، می‌توانیم نتیجه تحلیل حساسیت را به صورت یک شاخه زیر هر متغیر ببینیم. در این مثال شما شاخه زیر متغیر intertia3.w را باز کنید و همه متغیرهای intertia1.J، intertia2.J و intertia3.J را انتخاب کنید، شکل زیر میزان حساسیت intertia3.w را نسبت به intertia1.J ، intertia2.J و intertia3.J نشان می‌دهد. می‌توانید مشاهده کنید که در شروع شبیه‌سازی intertia1.J اثر کمی روی پاسخ intertia3.w دارد. هر چقدر به طرف پایان شبیه-سازی نزدیک می‌شویم این اثر کمتر می‌گردد. از این گذشته اثر intertia2.J در کل شبیه‌سازی بسیار کم و قابل چشم‌پوشی است. ازطرف دیگر اینرسی intertia3.J اثر قابل توجه و شدیدی روی شبیه‌سازی دارد. بنابراین می‌توان نتیجه گرفت که کنترل کننده حساسیت بالایی نسبت به تغییرات intertia3.J دارد.



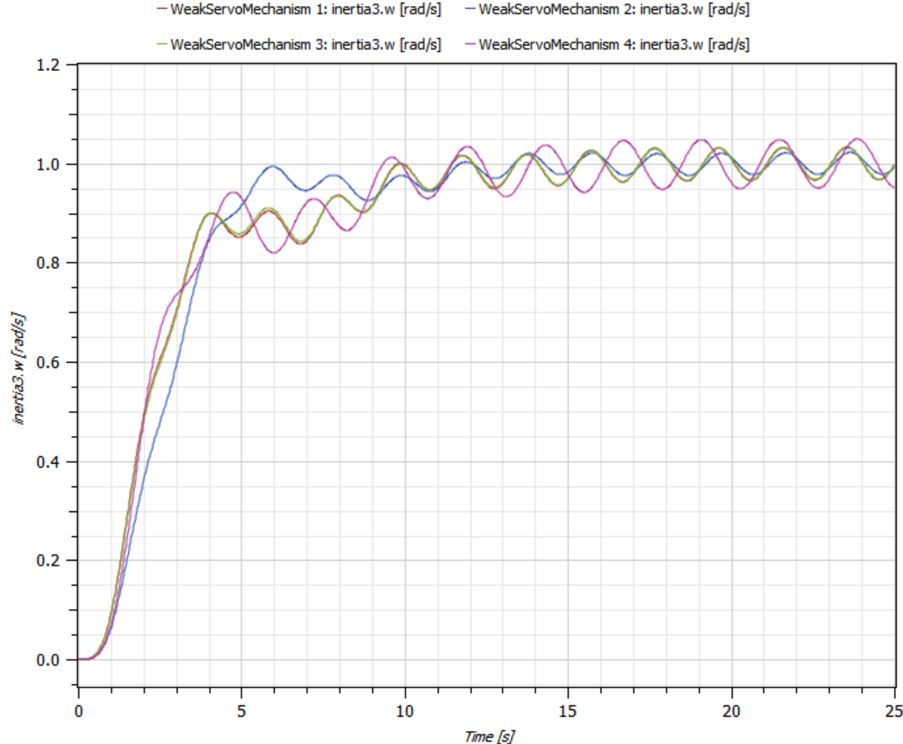
شکل ۲۲-۳. حساسیت intertia3.w نسبت به intertia1.J, intertia2.J و intertia3.J.

برای ارزیابی پاسخها به دست آمده، شبیه‌سازی‌های زیر را انجام می‌دهیم:

- WeakServoMechanism 1– پیش فرضهای اولیه
- WeakServoMechanism 2 - با افزایش دادن intertia2.J به میزان ۵۰ درصد
- WeakServoMechanism 3 -با افزایش دادن intertia1.J به میزان ۵۰ درصد
- WeakServoMechanism 4 - با افزایش دادن intertia3.J به میزان ۵۰ درصد

نتایج در شکل زیر نشان داده شده است و در آنجا می‌توانیم نتایج تحلیل‌هایمان را تثبیت کنیم:

- در WeakServoMechanism 2، تغییر intertia2.J تقریباً هیچ اثری در تمام شبیه‌سازی ندارد.
- در WeakServoMechanism 3، تغییر intertia1.J در شروع شبیه‌سازی اثر دارد و هرچه به سمت انتهای شبیه‌سازی پیش می‌رویم اثرش فقط به صورت تغییر کوچکی در دامنه نوسان باقی می‌ماند.
- در WeakServoMechanism 4، تغییر intertia3.J بیشترین اثر را دارد این تغییر باعث جابه‌جایی فاز نوسان می‌گردد، همچنین دامنه نوسانات را افزایش می‌دهد.



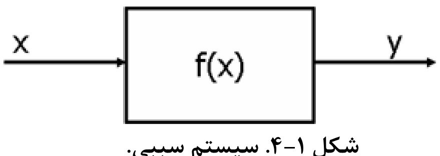
شکل ۳-۲۳. نتیجه تغییر intertia1.J, intertia2.J and intertia3.J.

فصل ۴ مدار الکتریکی ساده بر پایه قطعات

۴-۱) مدلسازی سببی و غیر سببی

تجهیزات مهندسی مانند موتورها و ترن برقی شامل تعداد زیادی از اجزاء فیزیکی مرتبط هستند. معمولاً این سیستمها با استفاده از بلوک دیاگرام مدلسازی می‌شوند. خطی که دو بلوک را به هم متصل می‌کند نشاندهنده ارتباط دو بلوک است. روابط حاکم بر این بلوکها قوانین فیزیکی است.

وقتی شبیه‌سازی با زبان Modelica انجام می‌گیرد، سیستم می‌تواند سببی یا غیرسببی باشد. بسیاری از ابزارهای شبیه‌سازی، محدود به مدلسازی سیستم‌های جریان سیگنال (signal flow) یا سببی هستند. نمونه واضح استفاد از مدلسازی سببی را می‌توان در نرم‌افزار Simulink (بخشی از نرم افزار MATLAB) دید. البته در سالهای اخیر استفاده از مدلسازی غیر سببی به Simulink هم رسیده است و بسیاری از کتابخانه‌های اضافه شده به این نرم‌افزار در ویرایشهای جدید امکان مدلسازی سیستمهای غیرسببی را نیز فراهم نموده‌اند. در اینگونه ابزارهای مدلسازی، سیگنال که یک عدد یا مجموعه اعداد متغیر با زمان در نظر گرفته می‌شود، یک جهته بوده و فقط به یک بلوک وارد می‌شود. آنگاه بلوک یک عملیات ریاضی کاملاً مشخص بر روی سیگنال انجام داده و سیگنال خروجی از طرف دیگر بلوک خارج می‌شود. یعنی جهت جریان اطلاعات در مسیرها باید کاملاً مشخص باشد. این روش برای مدلسازی سیستمهایی مفید است که می‌توان آنها را به شکل یکسری سیگنالهای یک جهته مشخص نمود، مانند سیستمهای کنترل و فیلترهای دیجیتال.



شکل ۱-۴. سیستم سببی.

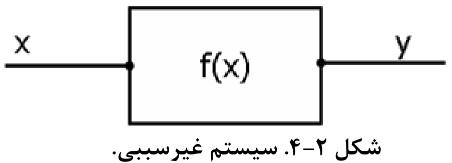
این متد مشابه اختصاص عبارت به یک متغیر و تعیین مقدار آن می‌باشد، محاسبات بر روی عبارت با متغیر یا متغیرهای معلوم در سمت راست انجام می‌گیرد و حاصل به متغیر سمت چپ اختصاص داده می‌شود.

$$y := f(x) \qquad (۴-۱)$$

اما مدلسازی سیستمهای واقعی با این روش بسیار زمانبر و مشکل است. مدلسازی سیستم‌های فیزیکی واقعی احتیاج به روش دیگری دارد. در مدلسازی غیرسببی، سیگنالی که بلوکها را به یکدیگر متصل می‌کند، می‌تواند در هر دو جهت انتقال پیدا کند. مشابه آن در برنامه‌نویسی یک دستور برابری ساده است:

$$y = f(x) \quad (۴-۲)$$

در این روش نرم‌افزار شبیه‌ساز با توجه به این که کدام مقادیر فیزیکی (برای مثال انرژی، جریان، گشتاور، حرارت و جریان جرمی) معلوم می‌باشد، اطلاعاتی در خصوص مسیر سیگنال نیز در اختیار خواهد داشت. بلوکها نیز دارای اطلاعاتی درباره قوانین فیزیکی حاکم (به صورت معادلات) و این که کدام متغیرهای فیزیکی باید معلوم فرض شوند، می‌باشد.

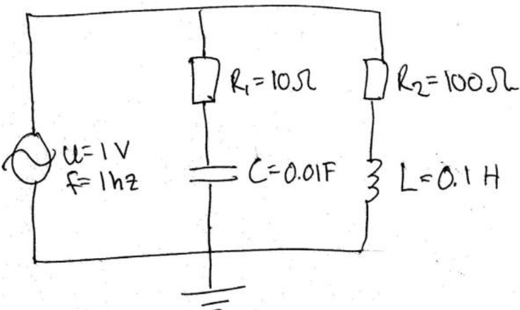


شکل ۲-۴. سیستم غیرسببی.

زبان Modelica امکان استفاده از هر دو روش را در اختیار کاربر قرار می‌دهد و کاربر می‌تواند همزمان با روش غیر سببی و با روش سببی مدلسازی خود را با توجه به نیاز زیر سیستم مورد نظر خود مدلسازی را انجام بدهد.

۲-۴) مدلسازی جریان سیگنال

مدلسازی بر اساس جریان سیگنال برای مسائل با روابط سببی کاملاً مشهود بسیار مناسب است. سیستمهایی که مسیر اطلاعات از ورودی تا خروجی کاملاً مشخص است. یک مثال از این نوع سیستم‌های برپایه سیگنال، سیستم کنترل است. مثلاً سیستم کنترل یک مخزن را در نظر بگیرید. سطح مخزن توسط یک حسگر اندازه‌گیری می‌شود و این سیگنال برای کنترل شیر وارد کنترل‌کننده شده و از آنجا وارد شیر می‌گردد. سیگنال هیچگاه نمی‌تواند مسیر معکوس را طی کند، سیگنال هیچگاه از شیر به کنترل کننده و از آنجا به سطح مخزن نمی‌رود (در حالت واقعی هیچگاه مقدار سیگنال باز و بسته بودن شیر نمی‌تواند سطح مخزن را تنظیم نماید!). در هر حال در بسیاری از موارد روابط سببی از پیش تعیین شده و مشخص وجود ندارد، برای مثال یک موتور با توجه به این که ورودی آن جریان باشد یا گشتاور، می‌تواند به عنوان یک موتور یا یک مولد مورد استفاده قرار گیرد. به عنوان یک مثال ساده مدار AC شکل ۳-۴را در نظر بگیرید.



شکل ۳-۴. پیش نویس شماتیک ازیک مدار AC.

مدار فوق برای نشان دادن تفاوت بین دو رویکرد جریان سیگنال (رویکرد سببی) و رویکرد قطعات (رویکرد غیرسببی) استفاده خواهد شد. در ادامه مدار فوق را با هر دو رویکرد، مدل کرده و تفاوت آنها را بررسی می‌کنیم.

۳-۴) مدار برپایه جریان سیگنال (سببی)

در این روش قبل از این که عملاً شروع به پیاده‌سازی مدل نماییم بایستی به موارد زیر توجه نمود:

- سیگنالهای ورودی و خروجی سیستم را مشخص نماییم.
- ترتیب قرارگرفتن معادلات را تنظیم کنیم.
- خروجی سیستم را به عنوان تابعی از ورودی بیابیم.

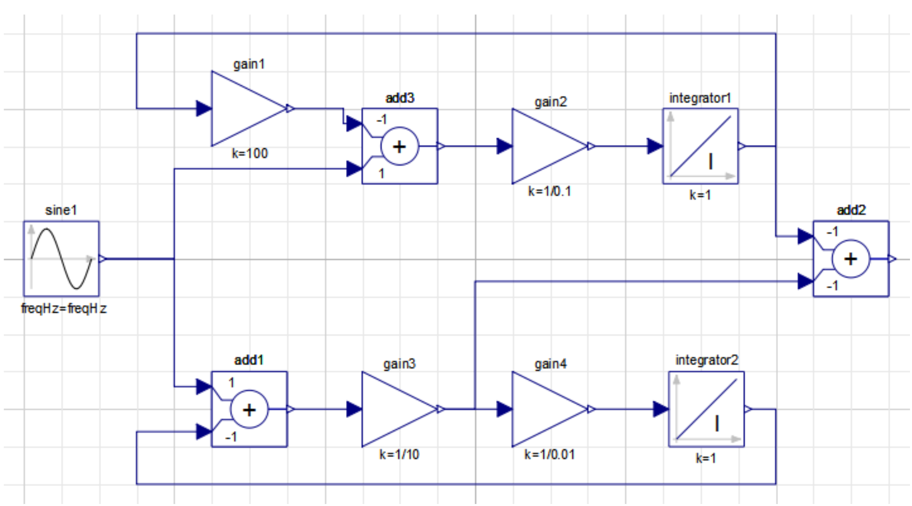
در این مثال می‌خواهیم جریان منبع ولتاژ را به عنوان تابعی از ولتاژ آن بررسی کنیم. با توجه به مدار ۳ معادله داریم:

$$
\begin{aligned}
u(t) &= R_1 i_1(t) + \frac{1}{C}\int i_1(t)dt \\
u(t) &= R_2 i_2(t) + L\frac{di_2(t)}{dt} \\
i(t) &= i_1(t) + i_2(t)
\end{aligned}
\qquad (۳-۴)
$$

در این معادلات i مجموع جریان منبع ولتاژ است، i_1 و i_2 به ترتیب جریان مقاومتهای الکتریکی ۱ و ۲ هستند. همان طور که در معادلات زیر نشان داده شده است، با استفاده از تبدیل لاپلاس بر روی معادلات بالا، i به عنوان تابعی از u به دست می‌آید:

$$i_1(t)=\frac{1}{R_1}\left(u(t)-\frac{1}{C}\int i_1(t)dt\right)$$
$$i_2(t)=\frac{1}{R_2}\left(u(s)-L\frac{di_2(t)}{dt}\right) \tag{۴-۴}$$

با استفاده از معادلات بالا می‌توانیم مدل جریان سیگنال مدار بالا را مانند شکل ۴-۴ ایجاد کنیم.

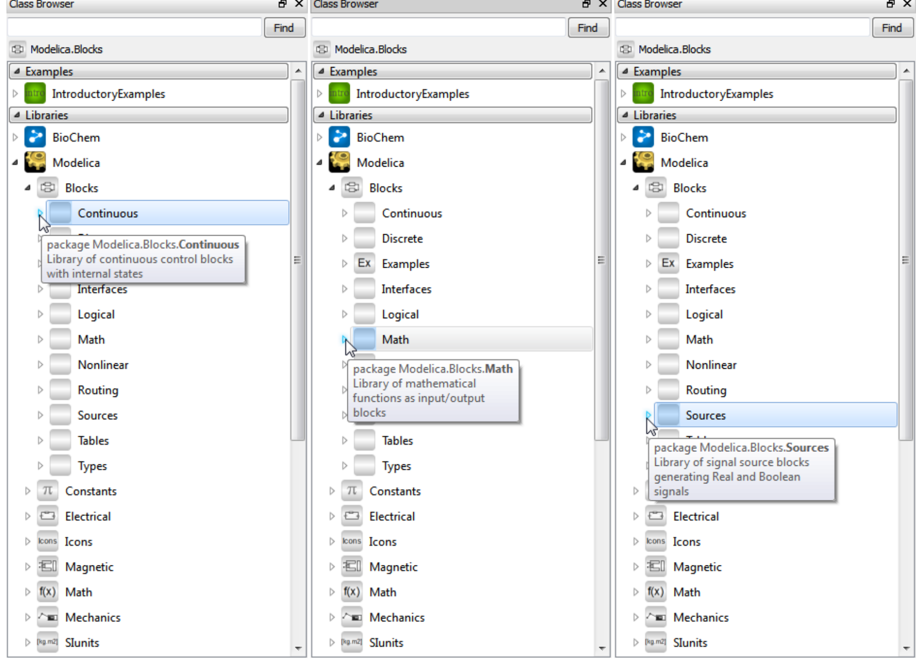


شکل ۴-۴. مدل IntroductoryExamples.ComponentBased.BlockCircuit.

یک مدل جدید بسازید، سپس تمام عضوها را از Librory Browser بیابید و آنها را به مدل جدید اضافه کنید. تمام عضوهای لازم برای ساختن سیستم در مسیر کتابخانه‌های زیر قرار دارند:

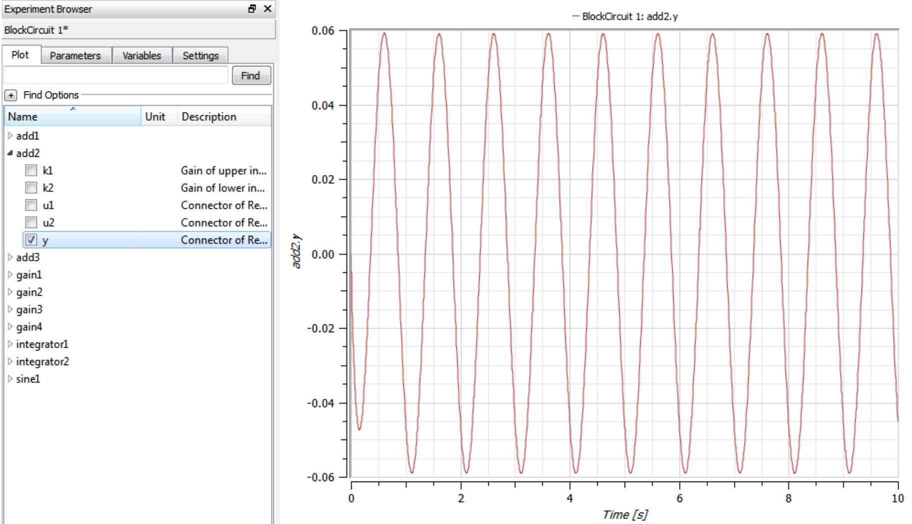
- Modelica>>Blocks>>Sources
- Modelica>>Blocks>>Math
- Modelica>>Blocks>>Continous

برای دیدن اعضاء کتابخانه Modelica>>Blocks>>Sources در Librory Browser، با تلیک روی علامت سمت چپ آیکون هر کتابخانه، ابتدا کتابخانه Modelica و به دنبال آن Blocks و Sources را باز کنید.



شکل ۵-۴. بازشدن کتابخانه Math , Sources و Continous درون Modelica>>Blocks.

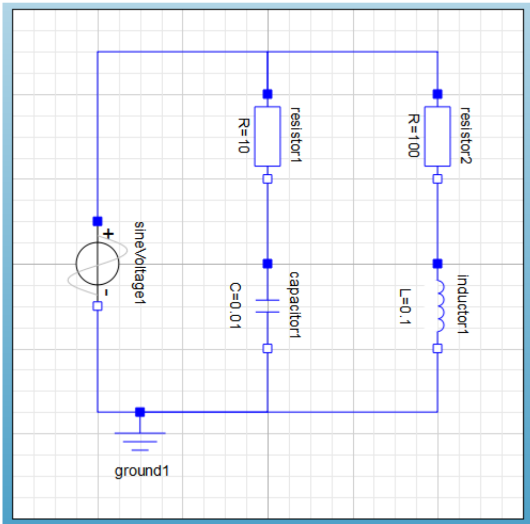
با استفاده از کشیدن و رها کردن (drag and drop) تمام عضوهای لازم را از Library Browser به مدل خود انتقال دهید و سپس بلوکهای را با ابزار اتصال به یکدیگر متصل کنید. مرکز شبیه‌سازی را فعال کنید و مدل را برای ۱۰ ثانیه مدلسازی کنید. جریان خروجی نتیجه add2 (قطعه جمع کننده) است. سیگنال‌های i_1 و i_2 به ترتیب از gain3 و integrator1 به دست می‌آید. جریان حاصل را در شکل ۴-۶ مشاهده می‌نمایید.



شکل ۶-۴. نمودار add2.y از IntroductoryExamples>>ComponentBased>>BlockCircuit بامقادیر پیش فرض.

۴-۴) مدار بر اساس قطعات (مدلسازی غیرسببی)

به طور طبیعی برای پیاده‌سازی مدار شکل ۳-۴ بر اساس قطعات تنها لازم است که عضوهای مدار را از کتابخانه به مدل خود انتقال دهید و آنها را به هم متصل کرده و پارامترهای هر یک را تنظیم کنید. این کار ما را به مدلی شبیه شکل اولیه مسئله می‌رساند. این مسئله اولین مزیت استفاده از روش غیرسببی است.



شکل ۷-۴. مدل IntroductoryExamples.ComponentBased.ElectricCircuit.

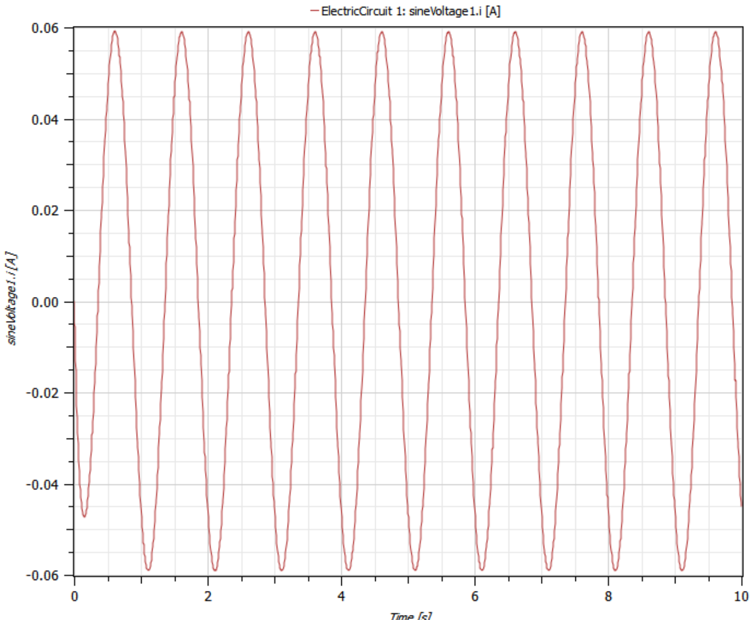
اگر می‌خواهید خودتان مدل را بسازید. عضو Sine Voltage درمسیر زیر قرار دارد:

Modelica>>Electrical>>Analog>>Sources

و بقیه عضوها درمسیرهای زیر می‌باشند:

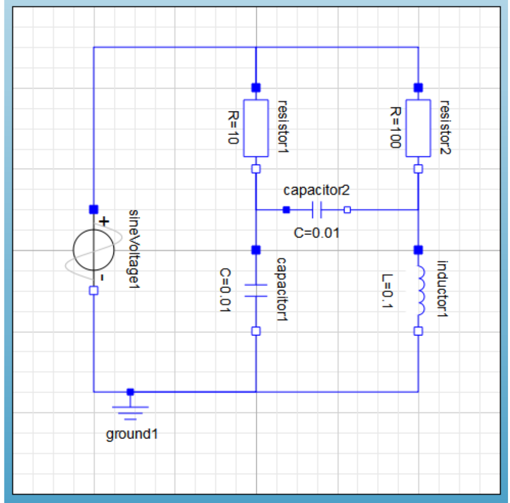
Modelica>>Electrica>>Analog>>Basic

توجه کنیدکه مقادیر برخی از پارامترها در مدل ما با مقادیر پیش فرض قطعات متفاوت است، بنابراین برای این که نتایج شبیه‌سازی یکسان باشد لازم است مقادیر را تغییر دهید. حالا می‌توانیم شبیه‌سازی را انجام داده و نتایج جریان سیگنال ولتاژ را رسم کنیم. همانطورکه انتظار داشتیم نتایج دقیقاً مطابق مدل سببی می‌باشد.



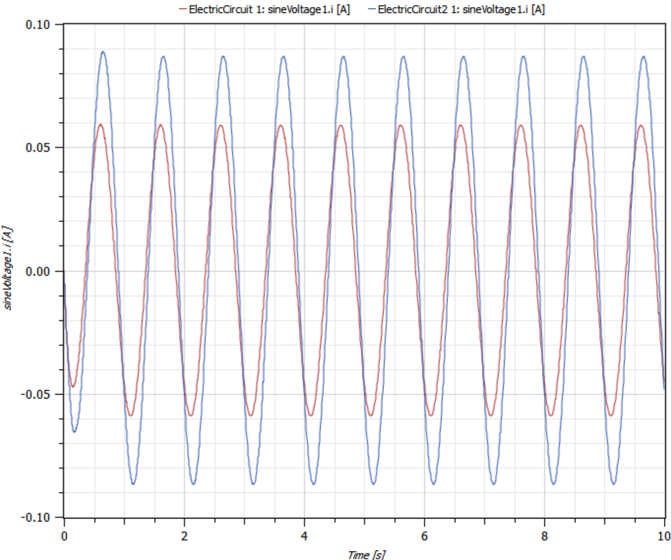
شکل ۴-۸. جریان IntroductoryExamples.ComponentBased.ElectricCircuit با مقادیر پارامتر پیش‌فرض.

این فصل را با اضافه کردن یک خازن دیگر به مدل همانند شکل زیر به پایان می‌رسانیم. خازن در مسیر کتابخانه Modelica.Electrical.Analog قراردارد.



شکل ۴-۹. مدار به همراه خازن IntroductoryExamples.ComponentBased.ElectricCircuit.

بعد از شبیه‌سازی مدل فوق، نتایج جریان‌ها را با یکدیگر مقایسه می‌کنیم. به نظر شما برای اضافه نمودن همین خازن به مدل سببی باید چه کاری انجام داد؟ همانگونه که مشاهده کردید دومین مزیت استفاده از روش غیرسببی، سادگی نگهداری مدل، تغییر و استفاده مجدد آن یا توسعه آن می‌باشد.



شکل ۱۰-۴. مقایسه جریانهای منبع در مدلهای ElectricCircuit و ElectricCircuit2.

تمرین

مدل بلوکی سببی مدار دوم را به دست آورید.

فصل ۵ قطعات کاربر – آونگ مرکب

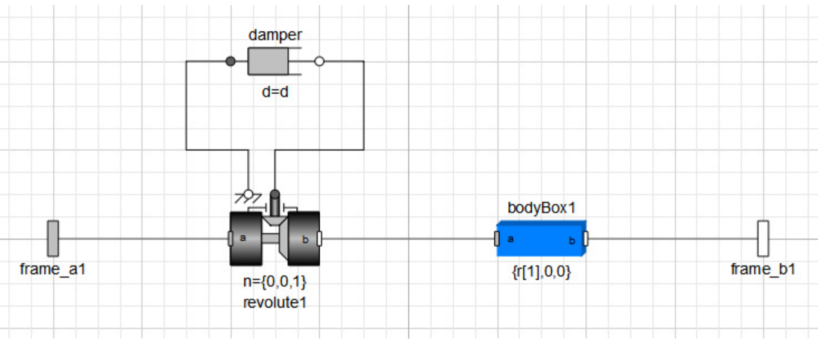
این فصل نشان می‌دهد که چگونه یک قطعه جدید بسازید و از آن استفاده کنید. یک آونگ مرکب را می‌توان به عنوان ترکیبی از آونگ‌ها با اتصال لولایی در نظر گرفت، هر آونگ می‌تواند حول اتصال خود به آونگ قبلی دوران کند. ما نشان خواهیم داد که چگونه می‌توان یک آونگ ساخت و از آن برای مدل کردن آونگ مرکب استفاده کرد.

۵-۱) آونگ

آونگ در مثالهای کتابخانه MultiBody وجود دارد و شامل یک جسم با امکان دوران حول یک مفصل لولایی می‌باشد. برای اضافه کردن اصطکاک به مفصل یک میرانه (Damper) به مفصل اضافه شده است.

برای ساختن مدل آونگ، لازم است یک مدل جدید بسازیم، قطعات مناسب را بیابیم، آنها را به مدل اضافه کنیم و آنها را با استفاده از ابزار اتصال به سایر قطعات متصل کنیم. برای آن که از یک مدل بتوان به عنوان یک قطعه استفاده کرد، لازم است برای مدل، درگاه اتصال بسازیم تا امکان اتصال به سایر قطعات دیگر را داشته باشد. نشان خواهیم داد که با استفاده از ابزار اتصال می‌توان به سادگی برای مدلها و قطعات، درگاه اتصال ایجاد کرد. همچنین پارامترهایی را برای انعطاف‌پذیری بیشتر به قطعه اضافه خواهیم نمود.

با ایجاد یک مدل جدید شروع کرده و آن را ”ChainLink“ نامگذاری می‌کنیم. قطعات لازم عبارتند از مفصل لولایی ”Revolute“ که در مسیر MultiBody.Joints قرار دارد، یک جسم ”BoxBody“ که در مسیر MultiBody.Parts قرار دارد و یک میرانه چرخشی ”Damper“ که در مسیر Modelica.Mechanics.Rotational آن را خواهید یافت. برای اضافه کردن قطعات به مدل ChainLink آنها را از جستجوگر کتابخانه به مدل خود بکشید. مدل در شکل ۵-۱ نمایش داده شده است.

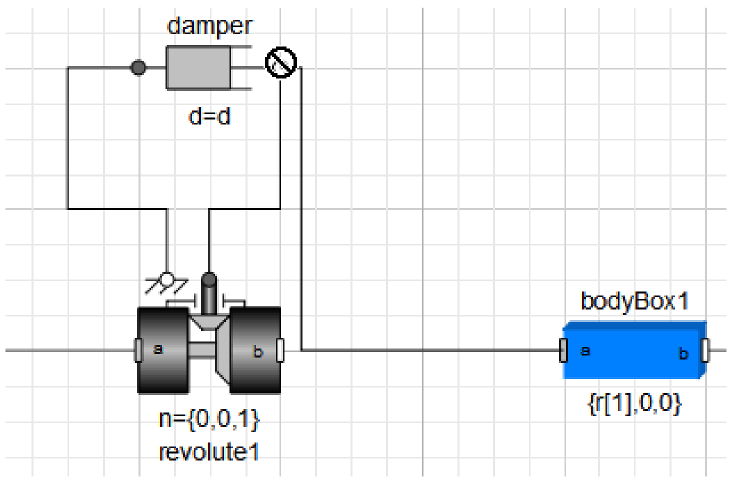


شکل ۵-۱. مدل ChainLink.

وقتی شما سه قطعه را به مدل افزودید، بایستی آنها را با استفاده از ابزار اتصال به هم متصل نمایید.

شکل ۲-۵. ابزار اتصال، واقع در نوار ابزار ویرایشگر مدل.

فقط درگاه‌های مشابه می‌توانند به یکدیگر متصل شوند. این قانون توسط ابزار اتصال کنترل می‌شود. اگر کاربر سعی کند دو درگاه ناسازگار را به یکدیگر متصل نماید، خط اتصال همانند شکل ۳-۵ از کار افتاده و آیکون خطا نمایش داده می‌شود. این کار باعث جلوگیری از ایجاد خطای ناخواسته در مدلسازی خواهد شد و بسیار مفید است.



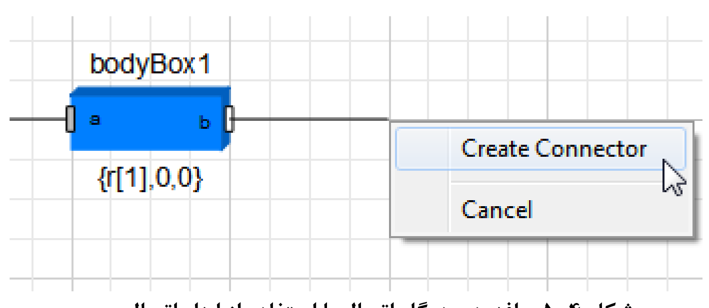
شکل ۳-۵.از کار افتادن خط اتصال هنگام سعی در اتصال بین دو قطعه ناسازگار.

اتصالات MultiBody به نام ”frames“ شناخته می‌شوند و نشاندهنده مختصات هستند. از آنجا که ما می‌خواهیم جسم حول یک نقطه دوران کند، frame_a1 قطعه boxBody1 را به frame_b1 لولا وصل می‌کنیم. میرانه با استفاده از ”flanges“ که یک اتصال مکانیکی یک بعدی است، به لولا متصل می‌گردد.

برای مثال، برای اتصال frame_a1 قطعه boxBody1 بهframe_b1 لولا، اشاره‌گر موشی را روی frame_a1 قطعه boxBody1 قرارداده و کلید چپ موشی را فشار داده، آنرا نگه دارید، اشاره‌گر را به محل frame_b1 لولا حرکت دهید و کلید را رها کنید تا اتصال برقرار گردد. به همین ترتیب دو فلنج میرانه، که اتصال یک بعدی مکانیکی هستند را به دو فلنج لولا متصل نمایید.

از آن جایی که می‌خواهیم از این مدل به عنوان قطعه در مدل‌های دیگر استفاده کنیم، باید به این مدل درگاه یعنی همان اتصال خارجی مناسب اضافه کنیم تا بتوان آن را به قطعات دیگر متصل

کرد. با وجود ابزار اتصال این کار ساده است. اشاره‌گر موشی را روی frame_b1 قطعه boxBody1 قرارداده و کلید چپ موشی را فشار داده، آن را نگه داشته و جابه‌جا نمایید تا به مکان مناسب برای قرارگیری درگاه جدید برسید، آنگاه برای خلق یک درگاه جدید راست تلیک کرده و گزینه خلق درگاه "Create Connector" را انتخاب نمایید. یک درگاه سازگار خلق می‌گردد. به همین ترتیب یک درگاه دیگر برای اتصال به frame_a1 لولا بسازید.



شکل ۴-۵ . افزودن درگاه اتصال با استفاده از ابزار اتصال.

حال می‌خواهیم برای افزایش انعطاف مدل ساخته شده، پارامترهایی را به آن بیفزاییم. این کار در نمایش متنی مدل امکانپذیر است. می‌دانید وقتی قطعات را به مدل اضافه و آنها را به هم متصل می‌کنیم، ویرایشگر مدل کد Modelica مربوط به هر عملیات و قطعه را تولید می‌کند. به حالت متنی نمایش مدل بروید تا کد Modelica تولید شده را ببینید. در نمایش متنی، مدل هر قطعه تعریف شده است و اتصالات بین هر دو قطعه در بخش معادلات با یک معادله اتصال نمایش داده می‌شود.

در ساختار متنی، بردار سه بعدی r برای مشخص کردن {طول، پهنا و بلندی} قطعه boxBody1 و ضریب تضعیف d را مانند متن زیر به عنوان پارامتر تعریف کنید.

```
Model ChainLink
    MultiBody.Joints.Revolute revolute1;
    MultiBody.Parts.BoxBody boxBody1(r=r);
    Modelica.Mechanics.Rotational.Damper damper1(d=d);
    MultiBody.Interfaces.Frame_a frame_a1;
    MultiBody.Interfaces.Frame_b frame_b1;
    parameter Real r[3]={1,0.1,0.1};
    parameter Real d=1.0;
equation
    connect(boxBody1.frame_b,frame_b1);
    connect(revolute1.frame_b,boxBody1.frame_a);
    connect(revolute1.frame_a,frame_a1);
    connect(damper1.flange_b,revolute1.axis);
```

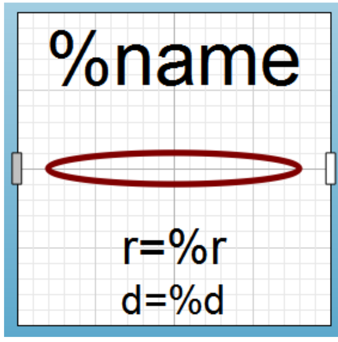
```
    connect(damper1.flange_a,revolute1.bearing);
end ChainLink;
```

حالا برای قطعه ساخته شده یک آیکون اختصاصی می‌سازیم. به نمایش آیکون بروید و با استفاده از ابزار گرافیکی واقع در نوار ابزار استاندارد، آیکونی برای قطعه ترسیم کنید.

شکل ۵-۵. ابزار گرافیکی واقع درنوار ابزار استاندارد.

توجه کنید که درگاه‌های قطعه که به وسیله ابزار اتصال ساخته شده‌اند، به طور خودکار به آیکون اضافه می‌گردد. در اینجا آیکون آونگ ساخته شده را با یک بیضی مشخص کرده‌ایم. برای تغییر خواص این بیضی روی آن دوبار تلیک کرده یا آن را انتخاب کرده و کلید Enter را بزنید. برای نمایش نام قطعه با استفاده از ابزار متنی نام آن را به شکل "%name" اضافه کنید. برای اضافه کردن هر پارامتر دیگر، کاربر باید "%ParameterName" را به متن اضافه کند. ما پارامترهای r و d را با اضافه کردن متنهای "%r" و "%d" اضافه می‌کنیم.

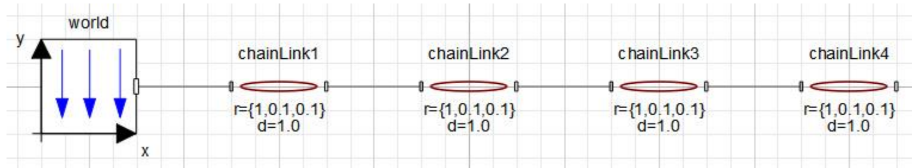
با این کار در هر بار استفاده از این مدل آونگ، هم نام آن و هم مقادیر متغیرهای r و d روی آیکون قطعه نمایش داده خواهد شد و باعث خوانایی بیشتر مدل می‌گردد.



شکل ۶-۵. نمایش آیکون آونگ.

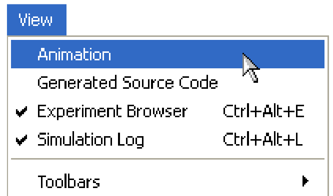
۵-۲) مدل آونگ مرکب

حال که مدل آونگ ساخته شد، مدل آونگ مرکب را با ترکیب چهار آونگ و یک فریم اولیه و اتصال دادن آنها می‌سازیم. مدل آونگ مرکب را در شکل ۷-۵ می‌بینید.

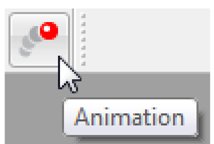


شکل ۷-۵. مدل آونگ مرکب.

به بخش شبیه‌سازی بروید و شبیه‌سازی را برای ۱۰ ثانیه انجام دهید. پس از انجام شبیه‌سازی پویانمایی آونگ مرکب با انتخاب گزینه Animation از منو View یا استفاده از آیکون نمایش پویا نمایی قابل نمایش است.

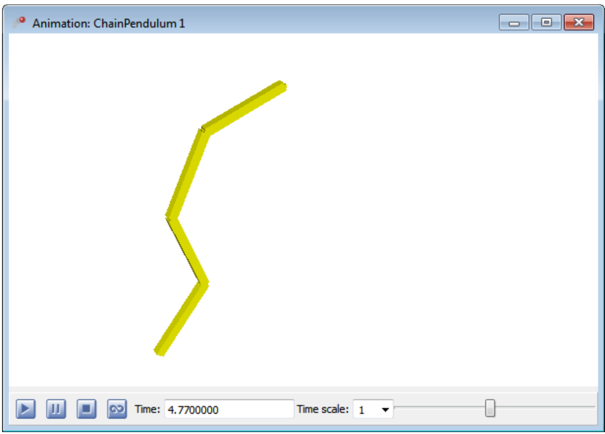


شکل ۸-۵. فعال کردن پویانمایی با استفاده از گزینه Animation از منو View.



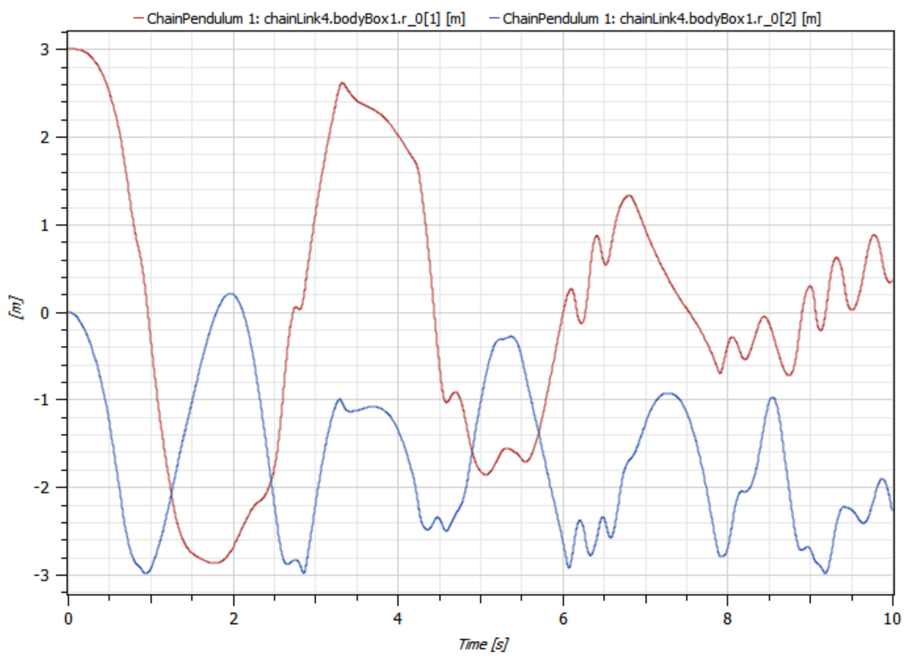
شکل ۹-۵. آیکون پویا نمایی.

شکل ۱۰-۵ پویانمایی آونگ مرکب را در ثانیه ۴.۷۷ را نمایش می‌دهد.



شکل ۱۰-۵. پویانمایی آونگ مرکب در ثانیه ۴.۷۷.

مکان انتهای آونگ در راستای x (محور افقی) و راستای y (محور عمودی) را در شکل ۱۱-۵ می‌بینید.



شکل ۱۱-۵. نمودار مکان افقی و مکان عمودی انتهای آونگ.

تمرین

خوانندگان مشتاق می‌توانند مدل آونگ مرکب عمومی‌تری را با استفاده از یک پارامتر به عنوان تعداد آونگها خلق کنند.

نکته: شما می‌توانید از حلقه for برای اتصال آونگ‌ها استفاده کنید. برای توضیحات بیشتر در خصوص ساختار و دستورات زبان Modelica به بخش سوم کتاب مراجعه نمایید.

فصل ۶ توابع خارجی و سیگنال Chirp

در حالی که نوشتن یک تابع در Modelica آسان است، گاهی اوقات فراخوانی روال‌های نوشته شده در C یا FORTRAN مناسب‌تر به نظر می‌رسد. این مثال چگونگی استفاده کردن از یک تابع خارجی نوشته شده در C را نشان می‌دهد.

۶-۱) تابع Chirp

سیگنال chirp یک موج سینوسی است با فرکانسی که به طور مداوم تغییر می‌کند و دارای خواص زیر:

- دارای بازه مشخص:

$$\Omega: \omega_1 \le \omega \le \omega_2$$

- یک دورتناوبی مشخص (بازه زمانی):

$$0 \le t \le M$$

ما از سیگنال زیر استفاده خواهیم کرد:

$$u(t) = A\cos\left(\omega_1 t + (\omega_2 - \omega_1)\frac{t^2}{2M}\right) \qquad (۶-۱)$$

فرکانس لحظه‌ایی در این سیگنال با دیفرانسیل گرفتن نسبت به زمان به دست می‌آید:

$$\omega_i = \omega_1 + \frac{t}{M}(\omega_2 - \omega_1) \qquad (۶-۲)$$

مشاهده می‌کنید که فرکانس لحظه‌ایی از حد پایین فرکانس تا حد بالا افزایش می‌یابد. دامنه و تغییرات خوب سیگنال، استفاده از این مدل را به عنوان فرکانس تحریک برای شناسایی سیستم امکان‌پذیر نموده است. در این مثال تابع chirp را در C تعریف کرده و سپس آن را به عنوان یک تابع خارجی در Modelica استفاده خواهیم کرد.

۶-۲) مدلسازی

برای فراخوانی یک تابع خارجی C، یک تابع Modelica به نام Chirp می‌سازیم. ساختن یک تابع مانند ایجاد یک مدل است با این تفاوت که محدودیت آن را مانند شکل زیر، Function انتخاب می‌نماییم (برای جزئیات بیشتر در خصوص چگونگی ساختن مدل به هر یک از مثالهای قبلی مراجعه کنید).

انتخاب تابع به عنوان محدودیت کلاس

شکل ۱-۶. ایجاد تابع.

```
function Chirp
    input Modelica.SIunits.AngularVelocity w_start;
    input Modelica.SIunits.AngularVelocity w_end;
    input Real A;
    input Real M;
    input Real t;
    output Real u "output signal";
    external "C" annotation(Include="#include \"Chirp.c\"");
end Chirp;
```

این تابع دارای پنج سیگنال ورودی، یک سیگنال خروجی و یک فراخوانی به تابع خارجی Chrip.c می‌باشد. در این تعریف فرض شده است که تابع Chirp.c این پنج ورودی را می‌گیرد و یک متغیر با دقت مضاعف باز می‌گرداند. اگر به دلایلی خواستید ترتیب ارسال متغیرها به تابع را عوض کنید، این کار با جابه‌جا کردن ترتیب تعریف متغیرهای تابع امکانپذیر است، متغیرها به ترتیب از بالا به پایین به تابع ارسال می‌گردند و نتیجه تابع در آخرین متغیر از تابع دریافت می‌گردد.

همچنین پیشنهاد می‌گردد برای خوانایی بیشتر و عدم ایجاد اشتباه خودتان ترتیب متغیرها را تعریف نمایید. برای مثال:

```
external "C" Chirp(t,A,M,w_start,w_end)
annotation(Include="#include \"Chirp.c\"");
```

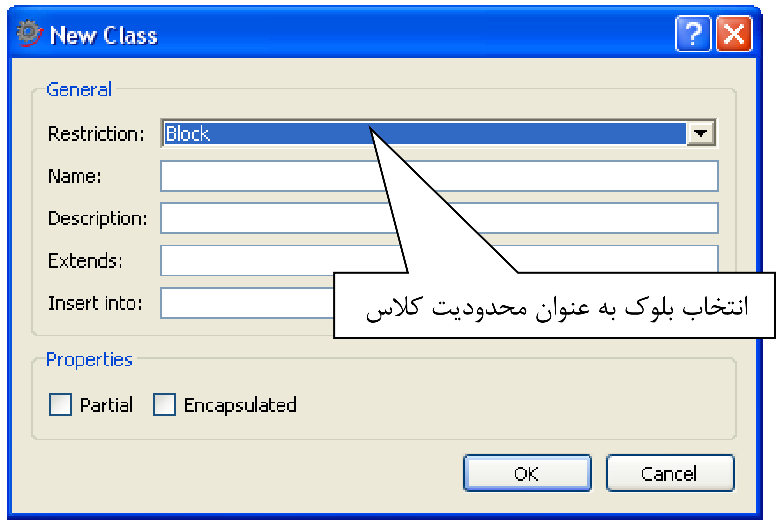
در مثال زیر ما یک تابع تعریف کرده‌ایم که از همین متغیرها به همین ترتیب استفاده خواهد کرد:

```
double Chirp(double w1, double w2, double A, double M, double time)
{
    double res;
    res=A*cos(w1*time+(w2-w1)*time*time/(2*M));
    return res;
}
```

تابع را می‌توانید در هر ویرایشگر متنی نوشته و آن را با نام Chirp.c ذخیره نمایید. در این مورد تابع C باید در همان کتابخانه توابع Modelica ذخیره شود. تابع را می‌توان در جاهای دیگر نیز قرار داد به شرط آنکه در مسیر کامل تابع تعریف و مشخصات آن نیز مطابق محل قرارگیری تغییر کند. برای مثال می‌توانید تابع را مستقیماً در ریشه دیسک C قراردهید:

```
external "C" annotation(Include="#include \"c:\\Chirp.c\"");
```

به محض اینکه تابع C ذخیره شد، تابع Modelica برای استفاده آماده است. برای این کار یک بلوک Modelica ایجاد کرده و تابع Chirp را از داخل آن فراخوانی می‌کنیم. ایجاد بلوک نیز مانند ایجاد مدل است با این تفاوت که محدودیت کلاس تعریف شده را از نوع Block انتخاب می‌کنیم (شکل ۲-۶).

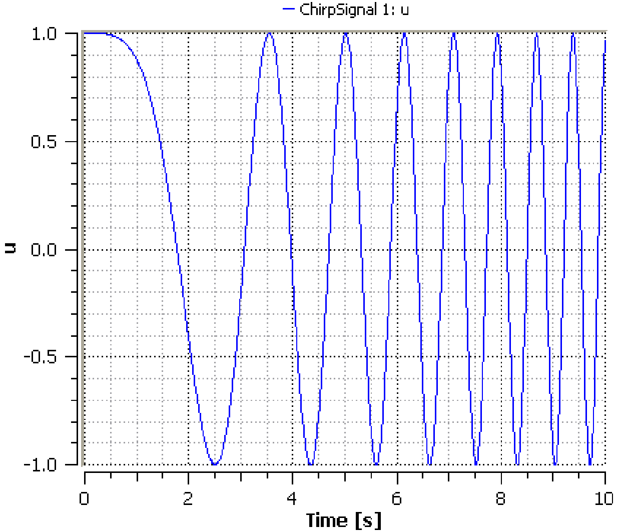


شکل ۲-۶. ایجاد بلوک.

فعلاً نگران تفاوت کلاسهای مختلف مانند مدل، تابع یا بلوک نباشید، در بخش سوم کتاب در این خصوص توضیح داده خواهد شد.

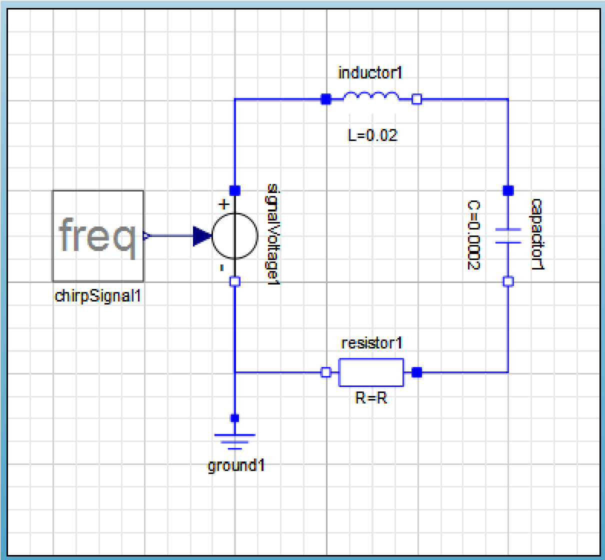
```
block ChirpSignal
        Modelica.Blocks.Interfaces.RealOutput u;
        parameter Modelica.SIunits.AngularVelocity w_start=0;
        parameter Modelica.SIunits.AngularVelocity w_end=10;
        parameter Real A=1;
        parameter Real M=10;
equation
    u=Chirp(w_start, w_end, A, M, time);
end ChirpSignal;
```

همان طورکه از نتیجه شبیه‌سازی در شکل زیر مشاهده می‌کنید، ما مقدار پارامترها را چنان تنظیم کرده‌ایم که سیگنال در ۱۰ ثانیه از صفر تا ۱۰rad/s افزایش یابد.



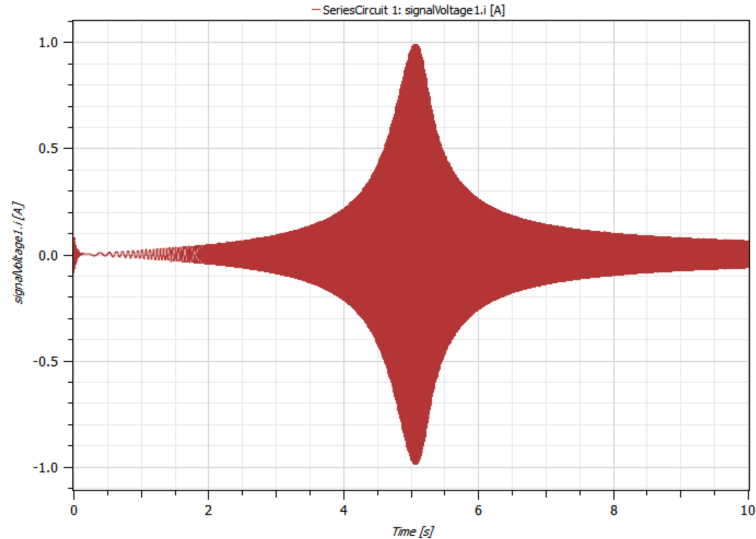
شکل ۳-۶. نمودار u سیگنال chirp مدل IntroductoryExamples.ExternalFunctions.SeriesCircuit.

حتماً متوجه شده‌اید که متغیر u همان طور که در بیشتر مدل‌های کتابخانه Modelica استفاده می‌گردد به عنوان یک درگاه از پیش تعریف شده modelica به صورت Modelica.Blocks.Interfaces.RealOutput تعریف شده است. در نتیجه ChirpSignal را می-توان در مدل‌های دیگر به عنوان منبع ورودی استفاده کرد. برای مثال برای ارزیابی فرکانس تشدید مدار الکتریکی زیر می‌توان از این سیگنال استفاده نمود.



شکل ۴-۶. IntroductoryExamples.ExternalFunctions.SeriesCircuit.

توجه کنیدکه مقادیر پیش‌فرض پارامترهای قطعات الکتریکی مطابق شکل بالا تغییر داده شده است. همچنین پارامترهای سیگنال chirp را چنان تغییر داده‌ایم که فرکانس 0 تا $1000\ ^{Rad}/_{S}$ را بپوشاند. سپس شبیه‌سازی را انجام داده و نتایج را بررسی می‌کنیم.



شکل ۵-۶. نمودار جریان i از IntroductoryExamples.ExternalFunctions.SeriesCircuit.

همانطورکه مشاهده می‌کنید در نزدیکی زمان ۵ ثانیه قله‌ای در سیگنال دیده می‌شود، که مطابقت دارد با فرکانس:

$$\omega_i = \omega_1 + \frac{t}{M}\left(\omega_2 - \omega_1\right) = 0 + \frac{5}{10}\left(1000 - 0\right) = 500\, rad/s \qquad (۶-۳)$$

یعنی فرکانس تشدید مدار بالا برابر $500\ ^{Rad}/_{S}$ است. فرکانس تشدید را در این مدار ساده می‌توان با استفاده از روش تحلیلی و با محاسبه، به دست آورد:

$$\omega_0 = \frac{1}{\sqrt{LC}} = \frac{1}{\sqrt{0.02 * 0.002}} = 500\, rad/s \qquad (۶-۴)$$

البته برای سیستم‌های پیچیده‌تر محاسبه فرکانس تشدید به روش تحلیلی بسیار مشکل است و در آنها استفاده از سیگنال Chirp بسیار عملی‌تر خواهد بود.

تمرین

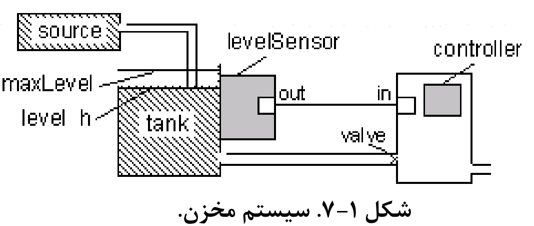
به سادگی می‌توان سیگنال Chirp را بدون استفاده از یک تابع خارجی به عنوان یک بلوک سیگنال Modelica پیاده سازی نمود. این کار به عنوان تمرین به عهده خواننده علاقه‌مند گذاشته شده است.

فصل ۷ شبیه سازی مخازن ذخیره

در این بخش یکی از مهمترین اصول شبیه‌سازی شیءگرا را خواهید آموخت که به شما در مدلسازی‌های پیچیده بسیار کمک خواهد کرد. در این بخش خواهید آموخت چگونه یک سیستم بزرگ را به بخش‌های کوچکتر شکسته و کتابخانه‌ای از تمامی قطعات لازم برای ساخت سیستم را ایجاد کنید. مثال‌های این بخش به شما می‌آموزد چگونه با استفاده از System Modeler یک مدل سلسله مراتبی ایجاد کنید. در ابتدا یک مدل ساده مخزن ذخیره ایجاد خواهیم کرد، سپس برای نزدیک شدن به مفهوم شیءگرا، با مدلسازی قطعات سیستم به صورت جداگانه، مدل مشابهی ایجاد می‌کنیم. در نهایت قابلیت انعطافی که این روش در اختیار ما قرار می‌دهد را برای آزمایش سناریوهای جدید استفاده خواهیم کرد.

۷-۱) مدل ساده مخزن

با یک سیستم ساده شامل یک مخزن به همراه یک کنترل‌کننده مطابق شکل ۷-۱ آغاز می-کنیم:



شکل ۷-۱. سیستم مخزن.

برای پیاده‌سازی مدل نیاز به معادلات ریاضی این سیستم داریم. تغییرات سطح آب داخل مخزن (h) نسبت به زمان($\dot{h}$)، تابعی از جریان ورودی و جریان خروجی مخزن و همچنین مساحت آن است:

$$\dot{h} = \frac{q_{in} - q_{out}}{A} \qquad (۷-۱)$$

به زبان ساده تغییرات حجم داخل مخزن برابر جریان ورودی به مخزن منهای جریان خروجی از مخزن است.

در این مثال جریان ورودی را به گونه‌ای در نظر می‌گیریم که به مدت ۱۵۰ ثانیه ثابت بوده و بعد جریان ۳ برابر گردد:

$$q_{in} = \begin{cases} flowLevel, & t < 150 \\ 3(flowLevel), & t \geq 150 \end{cases} \qquad (۷-۲)$$

در معادله بالا flowLevel یک پارامتر است. با کنترل جریان خروجی ما سعی خواهیم کرد که سطح آب درون مخزن را در یک سطح مرجع مطلوب یا سطح ref (سطح reference) نگه داریم. برای این منظور از کنترل‌کننده PI استفاده می‌کنیم. معادله ریاضی این کنترل کننده به شکل زیر است:

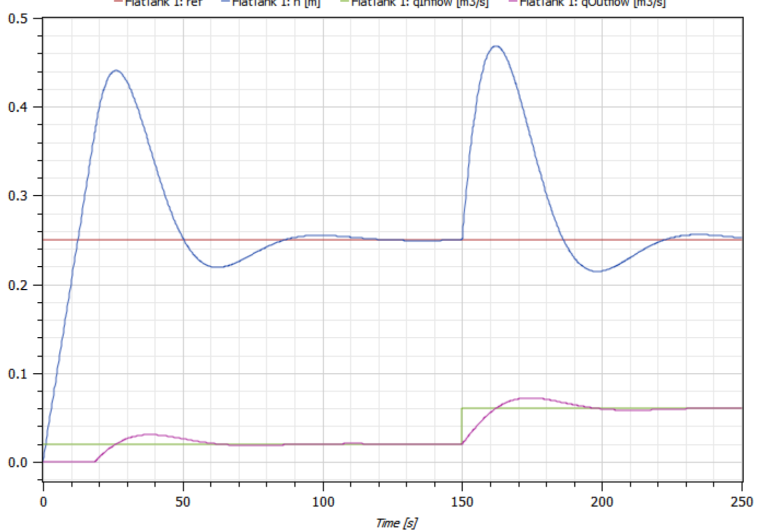
$$q_{out} = K\left(error(t) + \frac{1}{T}\int_{0}^{t} error(s)ds \right) \qquad (۷-۳)$$

که K بهره کنترل‌کننده و T ثابت زمانی این کنترل‌کننده است. در پایان، جریان خروجی را به مقادیر حداقل (minV) و حداکثر (maxV) محدود می‌کنیم. با این اطلاعات می‌توانیم کد Modelica این سیستم را مطالعه نماییم.

```
Model  FlatTank
  parameter  Real  flowLevel (unit="m3/s") =0.02;
  parameter  Real  area (unit="m2") =1;
  parameter  Real  flowGain (unit="m2/s") =0.05;
  parameter  Real  K=2 "Gain";
  parameter  Real  T(unit="s") =10  "Time constant";
  parameter  Real  minV=0, maxV=10;
  parameter  Real  ref=0.25  "Reference level for control";
  Real  h(start=0,unit="m") "Tank level";
  Real  qInflow(unit="m3/s") "Flow through input valve";
  Real  qOutflow(unit="m3/s") "Flow through output valve";
  Real  error "Deviation from reference level";
  Real  outCtr "Control signal without limiter";
  Real  x "State variable for controller";
equation
   assert(minV >= 0, "minV must be greater or equal to zero");
  der(h)=(qInflow - qOutflow)/area;
  qInflow=if time > 150 then 3*flowLevel else flowLevel;
  qOutflow=Functions.LimitValue(minV, maxV, -flowGain*outCtr);
  error=ref - h;
  der(x)=error/T;
  outCtr=K*(error + x);
end FlatTank;
```

```
function LimitValue
    input Real pMin;
    input Real pMax;
    input Real p;
   output Real pLim;
algorithm
   pLim:=if p > pMax then pMax else if p < pMin then pMin else p;
end LimitValue;
```

با شبیه‌سازی این مدل برای ۲۵۰ ثانیه می‌توانیم ببینیم که سطح مخزن شروع به افزایش کرده و به سطح مبنا رسیده و از آن بیشتر می‌شود. با بیشتر شدن سطح نسبت به سطح مبنا، خروجی باز شده و بعد از ۱۵۰ ثانیه سطح آب در مخزن متعادل می‌شود. در این لحظه جریان ورودی به صورت ناگهانی افزایش می‌یابد و قبل از این که کنترل‌کننده بتواند تغییرات را مدیریت کرده و سطح را پایدار کند، سطح آب افزایش می‌یابد. این مسئله در شکل ۲-۷ مشخص است.



شکل ۲-۷. نمودار سطح آب در مخزن وجریان ورودی و خروجی با مقادیر پیش فرض پارامترها.

۷-۲) مدل مخزن بر اساس قطعات

پیاده‌سازی مدل مخزن براساس قطعات نیاز به تلاش و وقت بیشتری دارد، اما وقتی شروع به انجام آزمایشات و بررسی سناریوهای مختلف با این مخزن می‌کنیم، سود زمان هزینه شده را باز خواهیم یافت. این روش مخصوصاً برای سیستم‌های پیچیده بسیار مناسب است. در کل روش شیءگرا، روش اصولی مدلسازی است و مدلسازی با آن بسیار آسان‌تر از روشهای دیگر است.

وقتی که از روش مدلسازی شیءگرا بر اساس قطعات استفاده می‌کنیم، ابتدا سعی می‌کنیم همه اجزاء و ساختار سیستم را درک کرده و آن را به صورت سلسله مراتبی از بالا به پایین مشخص می‌کنیم. وقتی که اجزای یک سیستم و ارتباطات بین این اجزاء دقیقاً تعیین شد، می‌توانیم اولین مرحله مدلسازی یعنی ایجاد درگاه‌ها و تعریف مدل‌های لازم را شروع کنیم.

در واقع مدلسازی شیءگرا دیدگاه بسیار منطقی و ساده‌ای دارد، به سیستم مخزن توجه نمایید. در شکل ۱-۷ می‌توان دید که این سیستم به صورت طبیعی دارای ساختار جزء به جزء است. پنج عضو در شکل قابل شناسایی است: اولین جزء خود مخزن است، اجزاء دیگر عبارتند از منبع مایع، حسگر سطح، شیر و کنترل‌کننده. برای ساده‌سازی در این مرحله مدل بسیار ساده‌ایی از حسگر سطح و شیر را در نظر خواهیم گرفت (یک متغیر اسکالر ساده برای هر یک از این اجزاء). برای سادگی به جای این که برای هر یک از این اجزاء کلاس‌های جدید ایجاد کنیم، این اجزاء را به صورت متغیرهایی ساده و حقیقی (Real) در مدل مخزن بیان می‌کنیم.

قدم بعدی تعیین ارتباطات بین اجزاء است. به سادگی مشخص است که جریان سیال از منبع به وسیله یک لوله به مخزن انتقال می‌یابد. سیال همچنین از طریق یک شیر کنترل شده از مخزن خارج می‌شود. کنترل‌کننده نیاز به مقدار اندازه‌گیری شده سطح سیال در مخزن دارد و این اندازه‌گیری توسط حسگر انجام می‌گیرد. در نتیجه باید یک مسیر از حسگر مخزن به کنترل‌کننده در نظر گرفته شود.

برای اتصال اجزاء باید درگاه‌های ارتباطی ایجاد کنیم. سپس نمونه‌هایی از این کلاس‌ها (درگاه) را به شیء اضافه می‌نماییم. در حقیقت مدل سیستم باید طوری طراحی شود که تنها ارتباط بین یک جزء و سایر اجزاء سیستم از طریق درگاه‌ها انجام شود.

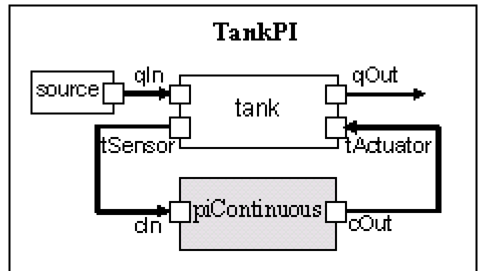
سرانجام باید به فکر استفاده مجدد، تعمیم و طراحی عمومی اجزاء باشیم. اگر پیش‌بینی می‌کنید که ممکن است به نمونه‌های متفاوت یک جزء نیاز داشته باشید، در این صورت بهترین روش ایجاد یک کلاس پایه با مشخصات مشترک است تا بتوان سایر کلاسهای مشابه را از آن مشتق کرد. برای مثال در مورد سیستم مخزن، انتظار داریم نمونه‌های مختلف کنترل‌کننده استفاده گردد. بنابراین ایجاد یک کلاس پایه برای کنترل‌کننده‌های سیستم مخزن برایمان مفید خواهد بود. در ابتدا با کنترل‌کننده PI آغاز می‌کنیم.

ساختار مدل سیستم مخزن با استفاده از روش شیءگرا[1] بر اساس قطعات[2] در شکل ۳-۷ دیده می‌شود.

---

[1] Object oriented

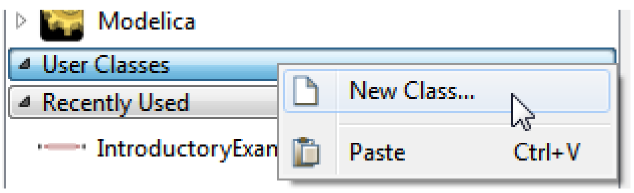
[2] Component base



شکل ۳-۷. سیستم مخزن با روش شیءگرا بر اساس قطعات.

ابتدا سه نوع کلاس مختلف که در ساختن این مدل به کار خواهند رفت را تعریف می‌کنیم: درگاه‌ها، توابع و قطعات. بنابراین برای نگهداری قطعات مخزن، بسته‌ای[1] را به عنوان کتابخانه اصلی ایجاد می‌کنیم. این کتابخانه خود شامل سه بسته یا کتابخانه دیگر است. بسته Package، کلاسی است که مانند یک کتابخانه برای نگهداری کلاس‌های مربوط به هم به کار می‌رود. ما از این به بعد آن را به نام کتابخانه خطاب می‌کنیم. برای ایجاد یک کتابخانه جدید روی ریشه درخت User Classes در Library Browser راست تلیک کرده و گزینه New Class را مانند شکل زیر انتخاب کنید. همچنین می‌توانید روی کتابخانه‌ایی که می‌خواهید کتابخانه جدید شما به آن اضافه گردد، راست تلیک کرده و سپس کلاس جدید (New class) را انتخاب کنید. روش دیگر استفاده از منو File و انتخاب گزینه New Class و انتخاب Package به عنوان محدودیت نوع کلاس است. کتابخانه‌های اصلی که به صورت پیش‌فرض در نرم‌افزار وجود دارد، برای جلوگیری از تغییرات ناخواسته قفل شده‌اند و شما مجاز به ایجاد کتابخانه درون آنها نیستید.

شما می‌توانید برای کتابخانه‌ها نیز مانند مدل‌ها آیکون بسازید. این کار به خوانایی بیشتر کتابخانه کمک می‌کند.



شکل ۴-۷. منوی ساختن New class.

در پنجره باز شده نوع کلاس را Package انتخاب کنید و آن را "Tank" بنامید. روی گزینه Ok تلیک کنید تا کتابخانه جدید ایجاد گردد. این کتابخانه در درخت اصلی ظاهر خواهد شد. با تلیک

---

[1] package

راست روی نام این کتابخانه جدید می‌توانید مدلها و کتابخانه‌های دیگر را به آن بیفزایید. شما باید سه کتابخانه به این کتابخانه اضافه نمایید. این کار را در بخشهای آینده انجام خواهیم داد.

۷-۲-۱) درگاه ها

درگاه، محل ارتباط هر مدل با دنیای بیرونش است. اکنون آماده هستیم تا درگاه‌های لازم برای اجزاء مخزن را ایجاد کنیم. ابتدا باید کتابخانه‌ای برای نگهداری همه این درگاه‌ها ایجاد نماییم. پس یک کتابخانه جدید به نام Interface در کتابخانه اصلی مخزن یعنی Tank ایجاد می‌کنیم. برای این کار به سادگی روی کتابخانه Tank تلیک راست کرده و گزینه New Class in Tank را انتخاب نمایید و در پنجره ظاهر شده نوع کلاس را به Package تغییر داده و در بخش نام Interface را وارد نمایید. نوشتن توضیحات اختیاری است. اولین درگاه را در کلاس Tank ایجاد نمایید. ایجاد کلاس درگاه نیز مانند ایجاد سایر کلاس‌ها می‌باشد فقط با این تفاوت که آن را از نوع Connector انتخاب می‌کنیم. درگاه یا connector کلاسی است که هیچ معادله‌ای نخواهد داشت، فقط تعدادی متغیر دارد که آنها را از یک درگاه به درگاه دیگر انتقال خواهد داد. برای اتصال این درگاه‌ها مانند گذشته، از ابزار اتصال استفاده خواهیم نمود.

یک کلاس درگاه برای خواندن سطح سیال ایجاد کنید و آن را ReadSignal بنامید. کد این درگاه را مانند کد زیر ویرایش نمایید (مثال Hello World را برای چگونگی ویرایش متنی مدلها و ساختن آیکن مدل ملاحظه کنید):

```
connector ReadSignal "Reading fluid level"
	Real val(unit="m");
end ReadSignal;
```

این درگاه فقط یک متغیر به نام val از نوع Real با واحد متر دارد. هر گاه این درگاه به درگاه مشابه وصل گردد، مقدار این متغیر بین دو درگاه انتقال خواهد یافت.

برای این درگاه یک آیکون مانند مثلث که به سمت راست اشاره می‌کند با رنگ سبز بسازید.

درگاه دوم برای انتقال سیگنال برای تنظیم عملگر موقعیت باز و بسته بودن شیر می‌باشد، برای درگاه دوم نیز کلاس و آیکون ایجاد کنید. آیکون این درگاه را مانند درگاه بالا و با رنگ نارنجی ترسم کنید.

```
connector ActSignal "Signal to actuator for setting valve position"
	Real act;
end ActSignal;
```

در نهایت یک درگاه برای جریان مایع ایجاد کرده و آیکون آن را به شکل یک دایره تو پر آبی رنگ در میان یک دایره بزرگتر به رنگ سفید بکشید. این درگاه هم به عنوان جریان ورودی و هم به عنوان جریان خروجی استفاده خواهد شد.

```
connector LiquidFlow "Liquid flow at inlets or outlets"
        Real lflow(unit="m3/s");
end LiquidFlow;
```

۷-۲-۲) ایجاد تابع محدودیت برای شیرکنترل

قبل از این که قطعات مختلف سیستم را ایجاد نماییم، تابعی برای محدود کردن جریان گذرنده از شیر کنترل ایجاد می‌کنیم. شیر کنترل در حالت واقعی حدی برای عبور جریان دارد. وقتی شیر کاملاً بسته است، جریان گذرنده از آن صفر است و حتی اگر کنترل‌کننده به شیرکنترل دستور بدهد که بسته‌تر بشود شیر در همین حالت باقی خواهد ماند و مقدار جریان از صفر کمتر نخواهد شد. از طرفی هنگامی که شیرکنترل کاملاً باز است حداکثر جریان ممکن را از خود عبور می‌دهد. حداکثر جریان عبوری از شیرکنترل تابعی از مشخصات فیزیکی سیال عبوری، مشخصات فیزیکی شیر و فشار دو طرف آن است. اگر سیستم کنترل هنگامی که شیرکنترل کاملاً باز است به شیر دستور بدهد که بیشتر باز شو، شیر نمی‌تواند بیشتر باز شود! بنابراین باید محدودیت جریان به شیر اعمال گردد. ما در این مدل تابع ساده‌ای را برای اعمال این محدودیت به شیر خواهیم ساخت.

تابع LimitValue تضمین می‌کند که مقدار عددی حداکثر جریان از حد مربوط به شرایط کاملاً باز و بسته شیر تجاوز نخواهد کرد. یک کتابخانه جدید به نام Fuctions بسازید و تابع LimitFunctionرا در این کتابخانه جدید ایجاد نمایید. چون قرار است LimitFunction یک تابع باشد، این کلاس را از نوع تابع (Functions) تعریف کنید. سپس برای این تابع یک آیکون بسازید.

```
Function  LimitValue;
   Input  Real  pMin;
   Input  Real  pMax;
   input  Real  p;
   output  Real  pLim;
algorithm
  pLim:=if p > pMax then pMax else if p < pMin then pMin else p;
end limitvalve  :
```

۷-۲-۳) اجزای مخزن

قدم بعدی ایجاد سه قطعه سیستم مخزن است. با ایجاد کتابخانه قطعات داخل کتابخانه Tank شروع کنید. این کتابخانه جدید را Components بنامید.

۱-۷-۲-۳) منبع سیال

سیال وارد شده به مخزن باید از جایی منشاء گرفته باشد. بنابراین ابتدا از ساخت یک منبع سیال شروع می‌کنیم. این بخش سیال سیستم مخزن را تأمین خواهد کرد و باید جریان آن در t=150 ثانیه به طور ناگهانی سه برابر جریان قبلی افزایش یابد. این تغییر یک مسئله کنترل جالب برای کنترل‌کننده مخزن خلق می‌کند و کنترل کننده باید از پس آن برآید. مدل منبع سیال را در کتابخانه قطعات ایجاد کنید و آن را LiquidSource بنامید. برای اتصال این منبع به سایر بخشها از یک درگاه سیال استفاده خواهیم نمود. پس در حالتی که مدل را با استفاده از میانبر ctrl+2 در حالت Diagram View قرار داده‌اید، درگاه LiquidFlow (این درگاه را در بخش قبل ایجاد کرده‌ایم) را از کتابخانه درگاه‌ها انتخاب کرده و به مدل خود بکشید. در پنجره اصلی مدل روی این درگاه تلیک راست کرده و گزینه properties را انتخاب کنید. پنجره خواص این درگاه به شکل ۵-۷ نمایش داده خواهد شد.

شکل ۵-۷. پنجره خواص.

نام این درگاه را به qOut تغییر بدهید. پنجره را با تلیک روی کلید OK ببندید. برای تغییر نام همچنین می‌توانستید از گزینه Rename در تلیک راست استفاده نمایید. با استفاده از میانبر ctrl+1 به حالت نمایش آیکون بروید و برای این قطعه یک آیکون شبیه به منبع سیال بکشید. با استفاده از میانبر ctrl+3، این مدل را در حالت متنی باز کرده و آن را به صورتی ویرایش نمایید که شامل دستورات زیر نیز باشد.

```
model  LiquidSource;
  parameter  Real  flowLevel=0.02;
  Tank.Interfaces.LiquidFlow qOut;
equation
  qOut.lflow=if  time > 150 then 3*flowLevel else flowLevel;
end LiquidSource;
```

۲-۷-۲-۳)مخزن

قطعه بعدی خود مخزن است. در کتابخانه قطعات یک مدل مخزن تحت عنوان "Tank" ایجاد کنید. مدل مخزن دارای چهار درگاه ارتباط می‌باشد، qIn برای جریان ورودی، qOut برای جریان خروجی، tSensor برای اندازه‌گیری سطح آب و qActuator برای تنظیم موقعیت شیر در خروجی مخزن، چون شیر به عنوان جزئی از مخزن در نظر گرفته شده است. در بخش آموزش OpenModelica شیر را به صورت مجزا مدل خواهیم نمود. این درگاه‌ها را از کتابخانه درگاه که قبلاً آنرا ایجاد کرده‌اید، بکشید و به مدل مخزن اضافه نمایید. شما باید دو درگاه برای سیال از نوع LiquidFlow و یک درگاه سیگنال از نوع ReadSignal و یک درگاه سیگنال شیر از نوع ActSignal به مدل خود اضافه نمایید. نام این اتصالات را به ترتیب به qIn، qOut، tSensor و qActuator تغییر بدهید. برای این کار روی هر درگاه در مدل مخزن تلیک راست کرده و گزینه Properties یا Rename را انتخاب نمایید و نام درگاه را تغییر دهید.

معادله اصلی تعیین‌کننده رفتار این مخزن معادله بقای جرم در شکل ساده آن با فرض جریان فشار ثابت است. اگر فشار دو طرف شیر را ثابت فرض کنیم، جریان خروجی تابعی از موقعیت شیر است. موقعیت شیر از طریق پارامتر flowGain و تابع LimitValue تعیین می‌گردد.

```
model  Tank;
    parameter   Real  area(unit="m2")=0.5;
    parameter   Real  flowGain(unit="m2/s")=0.05;
    parameter   Real  minV=0,maxV=10;
    Real    h(start=0.0,unit="m")  "Tank level";
    Hierarchical.Interfaces.ReadSignal tSensor "Connector, sensor reading tank
level (m)";
    Hierarchical.Interfaces.LiquidFlow qIn "Connector, flow (m3/s) through
input valve";
    Hierarchical.Interfaces.LiquidFlow qOut "Connector, flow (m3/s) through
output valve";
    Hierarchical.Interfaces.ActSignal tActuator "Connector, actuator controlling
input flow";
equation
assert(minV >= 0, "minV - minimum Valve level must be >= 0 ");
   der(h)=(qIn.lflow - qOut.lflow)/area;
equation
   qOut.lflow=Functions.LimitValue(minV, maxV, flowGain*tActuator.act);
   tSensor.val=h;
end  tank;
```

این مدل از درگاه‌هایی که پیش از این شرح داده شد و نیز تابع LimitFunction استفاده می‌کند.

۴-۲-۷) کنترل کننده

سرانجام کنترل‌کننده‌ها را می‌سازیم. ابتدا یک کنترل‌کننده PI خواهیم ساخت و بعد آن را با انواع دیگر جایگزین خواهیم کرد. رفتار یک کنترل‌کننده PI (تناسبی و انتگرالگیر) با دو معادله زیر تعریف می‌شود:

$$\frac{dx}{dt} = \frac{error}{T}$$
$$outCtr = K*(error + x) \quad (۷-۴)$$

در این روابط x متغیر حالت و error تفاوت بین مقدار مرجع و مقدار واقعی سطح مایع است. مقدار واقعی مایع از حسگر خوانده می‌شود، T ثابت زمانی، outCtr سیگنال خروجی به عملگر شیر برای کنترل موقعیت آن و K مقدار بهره کنترل‌کننده است. این دو معادله در کلاس کنترل‌کننده PIcontinuousController قرار خواهند گرفت، اگر به خط دوم تعریف این کلاس توجه کنید خواهید دید که کلاس فعلی، کلاس BaseController را توسعه داده است. به این معنی که این کلاس شامل کلاس پایه BaseController نیز هست و از آن مشتق گردیده است و معادلات خاص خودش را نیز دارد. بعداً کلاس BaseController را به گونه‌ای خواهیم نوشت که قابل گسترش باشد.

```
model PIcontinuousController
   extends BaseController(K=2,T=10);
   Real x "State variable of continuous PI controller";
equation
   der(x)=error/T;
   outCtr=K*(error + x);
end PIcontinuousController;
```

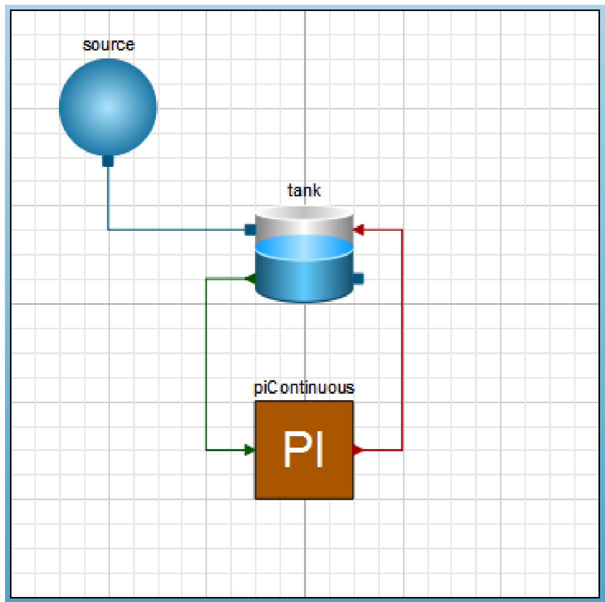
هر دو کنترل‌کننده PI و PID که بعداً تعریف می‌گردد، رفتار کلاس پایه BaseController را به ارث می‌برند. کلاس پایه BaseController شامل پارامترهای عمومی، متغیرهای حالت و دو اتصال‌دهنده است (یکی برای قرائت حسگر و دیگری برای کنترل وضعیت شیر).

```
partial model BaseController;
   parameter Real Ts(unit="s")=0.1 "Time period between discrete samples";
   parameter Real K=2 "Gain";
   parameter Real T(unit="s")=10 "Time constant";
   parameter Real ref "Reference level";
```

```
    Real error "Deviation from reference level";
    Real outCtr "Output control signal";
    IntroductoryExamples.Hierarchical.Interfaces.ReadSignal cIn "Input sensor
level, connector";
    IntroductoryExamples.Hierarchical.Interfaces.ActSignal cOut "Control to
actuator, connector";
equation
    error=ref - cIn.val;
    cOut.act=outCtr;
end BaseController;
```

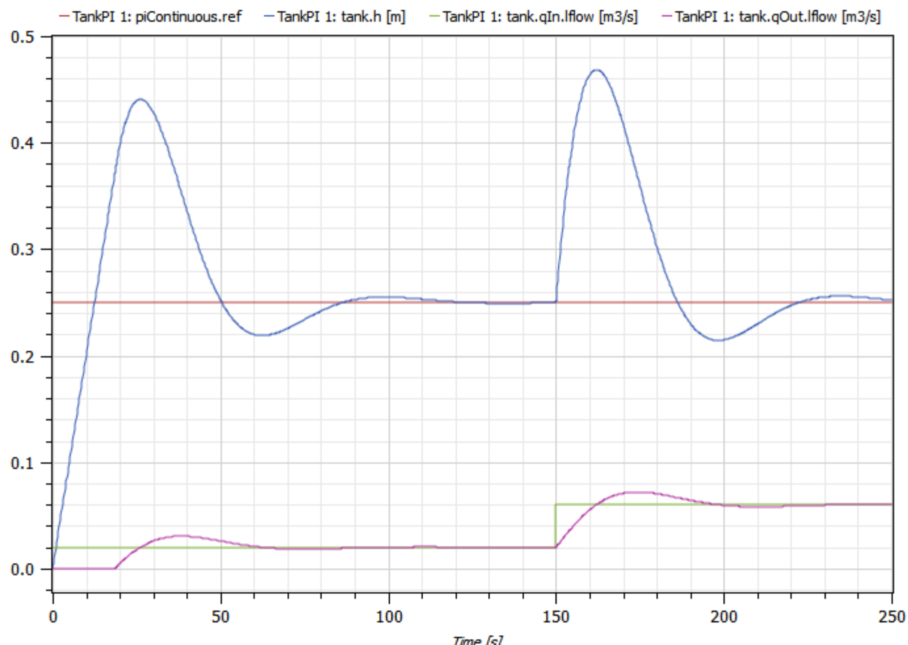
۷-۲-۵) سیستم مخزن کوچک

وقتی مراحل بالا به پایان رسید، می‌توانیم با استفاده از drag-and-drop کردن قطعات، مدل سیستم مخزن خودمان را بسازیم. این کار را مطابق شکل ۶-۷ انجام می‌دهیم.



شکل ۶-۷. مدل IntroductoryExamples.Hierarachical.TankPI.

شبیه‌سازی ۲۵۰ ثانیه‌ایی نتایجی مشابه با نتایج سیستم Flat tank ارائه خواهد داد.



شکل ۷-۷. نمودار سطح مخزن و جریان ورودی وخروجی، مخزن PI با مقادیر پیش فرض.

۷-۳) مخزن با کنترل کننده PID پیوسته

برای آشنایی با قدرت روش شیءگرا، یک سیستم TankPID را تعریف خواهیم کرد که تقریباً همانند سیستم TankPI است. تنها تفاوت این دو مدل این است که در اینجا به جای کنترل‌کننده PI از کنترل‌کننده PID استفاده خواهد شد. به وضوح برتری رویکرد شیءگرا براساس قطعات را نسبت به روش سنتی (کل مدل در یک سیستم) خواهید دید، زیرا قطعات سیستم را به آسانی می‌توان جایگزین کرد.

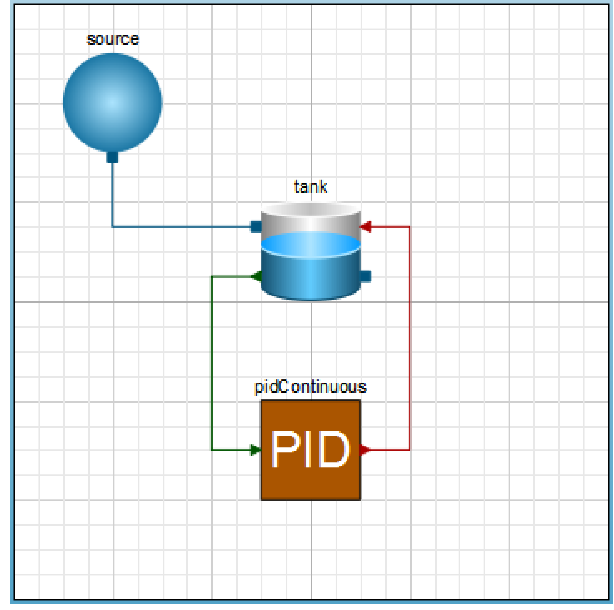
یک کنترل‌کننده PID (تناسبی، انتگرالگیر و مشتق‌گیر) را می‌توان با روشی مشابه کنترل-کننده PI مدل کرد. معادلات اساسی یک کنترل‌کننده PID به شکل زیر است:

$$\begin{aligned} \frac{dx}{dt} &= \frac{error}{T} \\ y &= T\,\frac{derror}{dt} \\ outCTR &= K\,(error + x + y) \end{aligned} \qquad (۷-۵)$$

با استفاده از این معادلات و نیز کلاس پایه BaseController، کنترل‌کننده PID را ایجاد می‌کنیم:

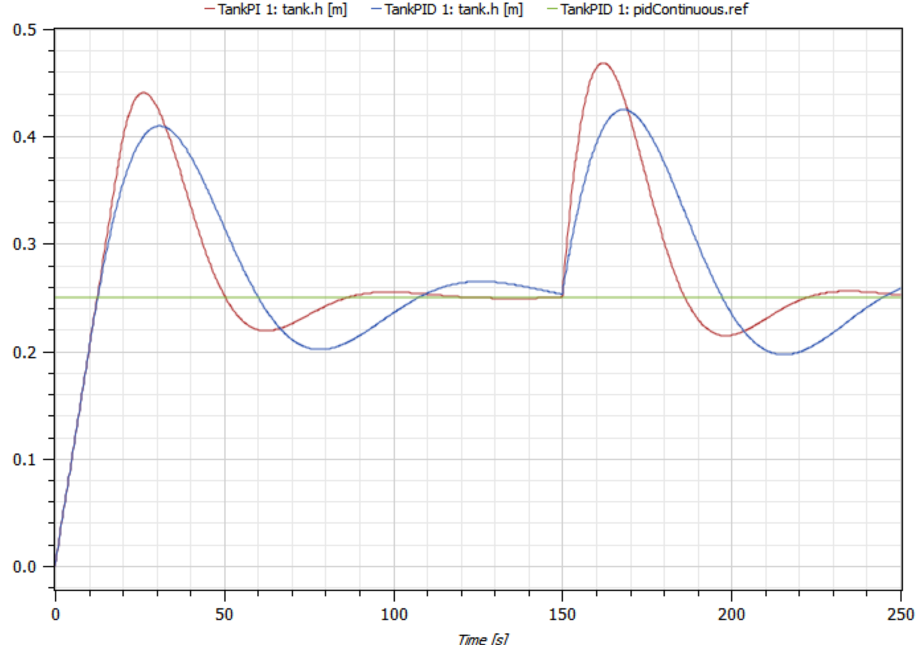
```
model PIDcontinuousController
    extends BaseController(K=2,T=10);
    Real x "State variable of continuous PID controller";
    Real x "State variable of continuous PID controller";
equation
    der(x)=error/T;
    y=T*der(error);
    outCtr=K*(error + x + y);
end PIDcontinuousController;
```

اکنون می‌توانیم سیستم مخزن با کنترل‌کننده PID را با استفاده از روش drag-and-drop بسازیم.



شکل ۸-۷. مدل IntroductoryExamples.Hierarachical.TankPID.

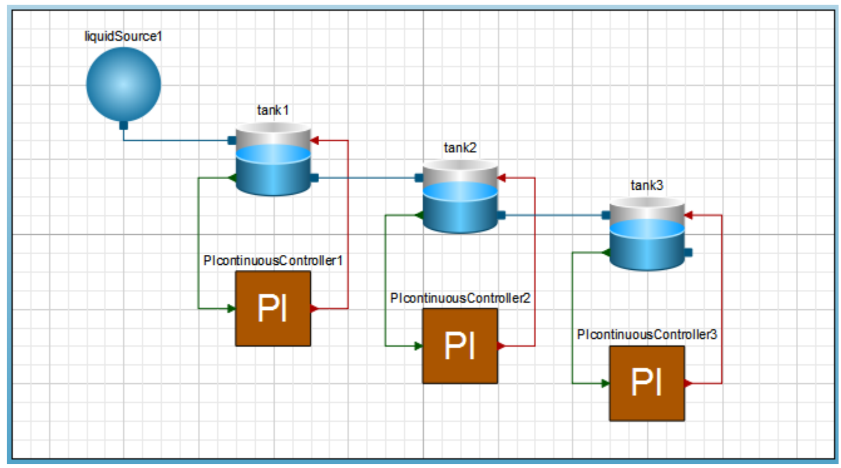
دوباره به مدت ۲۵۰ ثانیه شبیه‌سازی را انجام داده و نتایج را با نتایج قبلی مقایسه می‌کنیم.



شکل ۹-۷. مقایسه سطح مخزن بین مدل‌های TankPI وTankPID.

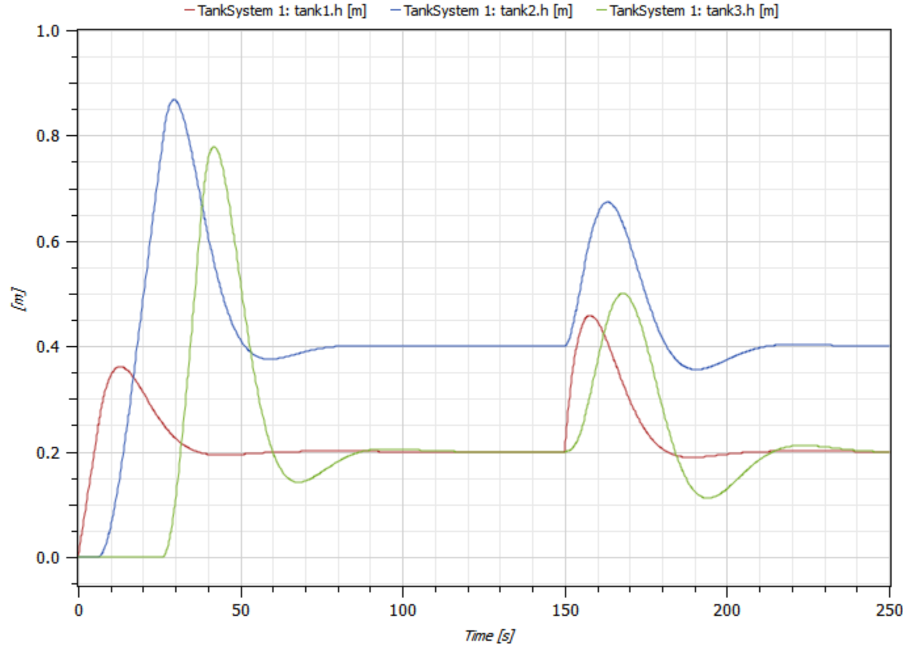
۷-۴) سیستم با سه مخزن

در نهایت به لطف روش مدلسازی شیءگرا براساس قطعات، می‌توانیم سیستمهای بزرگتر را به راحتی ترکیب کنیم.



شکل ۱۰-۷. مدل IntroductoryExamples.Hierarachical.TankSystem.

با شبیه‌سازی این سیستم، می‌توانیم چگونگی کنترل سطح هر کدام از مخزن‌ها را مطالعه کنیم.



شکل ۱۱-۷. ارزیابی سطح مخزن مدل TankSystem.

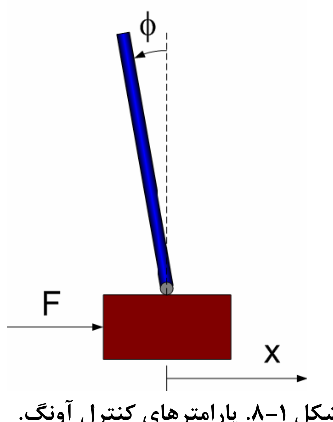
توجه داشته باشید که سطح مرجع مخزن دوم ۰/۴ متر است در حالی که دو مخزن دیگر دارای سطح مرجع ۰/۲ متر هستند.

فصل ۸ آونگ معکوس

حفظ تعادل یک چوب بلند روی یک انگشت، یکی از بازیهای جالب بچه‌هاست. همین سیستم با استفاده از تجهیزات مکانیکی و کنترلی قابل ساخته است. متعادل نگه داشتن این ساختار که آونگ معکوس نامیده می‌شود یکی از مسائل جذاب مهندسی کنترل است. در این بخش برای نشان دادن توانایی نرم‌افزار، نگاهی به سیستم آونگ معکوس خواهیم انداخت. این مدل با تجهیزات ساده موجود در کتابخانه modelica ایجاد شده است و دارای پویانمایی گرافیکی نیز می‌باشد.

۸-۱) آونگ معکوس

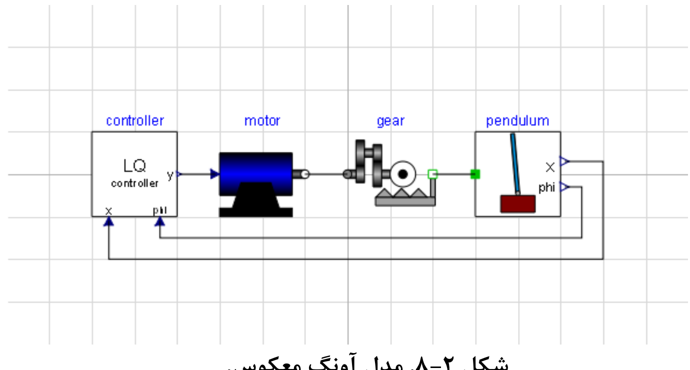
یک مسئله کلاسیک مهندسی کنترل، یک آونگ معکوس است. سیستم آونگ شامل موتور الکتریکی، چرخدنده و یک آونگ متصل به ارابه‌ای کوچک است. هدف عمود نگه داشتن آونگ در شرایطی است که گاهی به آونگ ضربه‌هایی زده می‌شود تا تعادل آن به هم بریزد و آونگ بیفتد. موتور الکتریکی ارابه را به حرکت در می‌آورد و آونگ به ارابه آویزان است. بایستی با کنترل موتور ارابه را به صورتی حرکت داد که آونگ به صورت عمودی در بالا قرار گرفته و تعادل خود را در مقابل اغتشاشات وارده مثلاً ضربه خارجی حفظ نماید. مکان ارابه به وسیله کنترل‌کننده با طراحی LQ[1] کنترل می‌گردد. اولویت اول کنترل‌کننده عمود نگه داشتن آونگ معکوس است.



شکل ۱-۸. پارامترهای کنترل آونگ.

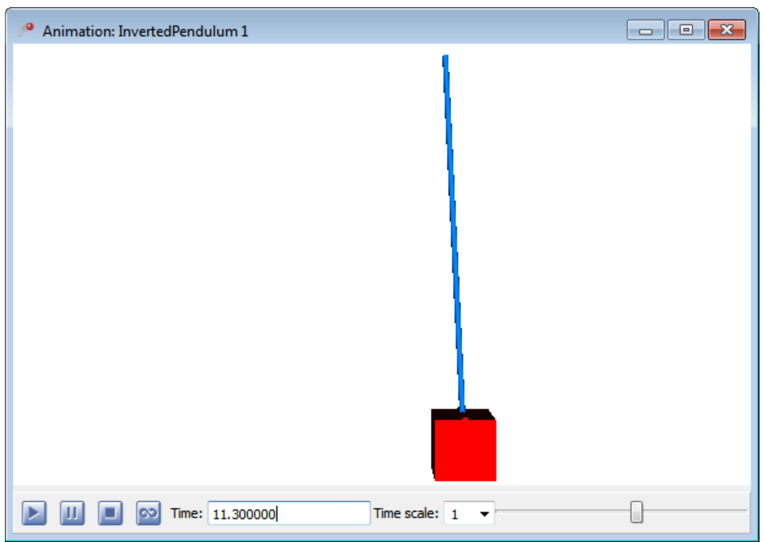
برای کنترل آونگ، کنترل‌کننده مکان ارابه و زاویه آونگ φ را به عنوان ورودی از سیستم می‌گیریم و با کنترل کردن موتور الکتریکی سیستم را کنترل می‌کند. مدل این ساختار در شکل زیر دیده می‌شود. این مدل در مثال‌های نرم‌افزار در دسترس است.

[1] Linear Quadratic



شکل ۲-۸. مدل آونگ معکوس.

مدل را بیابید و آن را برای ۲۰ ثانیه شبیه‌سازی کنید سپس پنجره Animation را از منو View باز کرده و پویانمایی تولید شده را ببینید. شکل ۳-۸ پنجره پویانمایی را در ثانیه ۱۱.۳ با استفاده از پالس به عنوان سیگنال مرجع (referenceType=2) نشان می‌دهد.



شکل ۳-۸. پویانمایی آونگ.

مکان ارابه با تغییر پارامتر referenceType قابل تغییر است. با انتخاب گزینه جدول زمان (referenceType=3) شما می‌توانید سیگنال دلخواه خودتان را ایجاد کنید. همچنین پارامترهای مختلف سیستم را تغییر دهید، برای مثال محدودیت حداکثر مقدار سیگنال کنترل‌کننده (controller.uMax) یا طول آونگ (pendulum.l_pendulum) و نتایج را ارزیابی کنید.

با استفاده از موشی می‌توانید زاویه دید شبیه‌سازی سه بعدی را تغییر بدهید.

فصل ۹ توصیه هایی در مدلسازی

این فصل در زمینه مدلسازی به کمک زبان Modelica به شما اطلاعاتی خواهد داد و توصیه‌هایی برای مدلسازی با نرم افزار SystemModeler بیان خواهد شد.

۹-۱) مقادیر اولیه

روشهای مختلفی برای مقدار دهی اولیه به متغیرهای در زبان Modelica وجود دارد. در این بخش معمولترین روشهای مقدار دهی اولیه بیان خواهد شد.

۹-۱-۱) خاصیت Start

مقدار دهی اولیه متغیرهای حالت را می‌توان با استفاده از خاصیت start در تعریف متغیر مشخص نمود یا می‌توان معادلات مقدار دهی اولیه را در بخش intial equation/algorithm اضافه نمود. اگر یک متغیر حالت داشته باشیم مشخص کردن مقدار اولیه متغیر در شروع شبیه‌سازی لازم است. اگر برای متغیری مقدار اولیه تعریف نشده باشد، با توجه به نوع آن متغیر از مقدار اولیه پیش‌فرض استفاده خواهد شد. مقدار اولیه پیش‌فرض برای متغیرهایی از نوع Real و Integer برابر صفر و برای متغیرهایی از نوع Boolean مقدار اولیه false است.

در حالی که متغیرهای حالت حتماً در شروع شبیه‌سازی مقدار اولیه دارند، ممکن است سایر متغیرهای غیر حالت مقدار اولیه نداشته باشند. برای متغیرهای غیر حالت، مقدار اولیه به عنوان حدس اولیه در نظر گرفته خواهد شد. برای مثال در مدل زیر متغیر x یک متغیر حالت است (چون در رابطه der استفاده شده است) و مقدار اولیه ۱ دارد، بنابراین مقدار x در شروع شبیه‌سازی برابر ۱ خواهد بود.

```
model A
      Real x(start=1);
equation
      der(x) = -x + 1;
end A;
```

این نکته که مترجم زبان چگونه متغیر حالت را انتخاب می‌کند به عوامل زیادی بستگی دارد. برای انتخاب متغیر حالت، متغیرهایی که در عبارت der استفاده می‌شوند ارجحیت دارند. برای کنترل نحوه انتخاب متغیر حالت می‌توان از خواص stateSelect یا fixed در تعریف متغیرها استفاده نمود.

۹-۱-۲) مقادیر حدس

مقادیر اولیه می‌تواند به حل کننده در یافتن مقدار صحیح متغیرهای غیرحالت کمک کند. رابطه Z^2=6-Z دارای دو جواب ۲ و ۳- است. برای کمک به حل کننده برای یافتن مقدار صحیح، می‌توان

مقدار شروع (حدس اولیه) برای Z تعریف نمود. این مقدار بایستی تا حد ممکن به حل واقعی نزدیک باشد. برای مثال به مدلهای زیر توجه نمایید، اگر مقدار اولیه ۴ برای Z تعریف شود، حل برابر Z=2 خواهد بود در حالی که اگر مقدار ۵- به عنوان حدس اولیه مشخص شود نتیجه Z=-3 خواهد بود.

```
model B1
Real z(start=4);
equation
z^2=6-z;
end B1;

Result: z=2

model B2
      Real z(start=-5);
equation
      z^2=6-z;
end B2;

Result: z=-3
```

حدس اولیه زمانی مورد استفاده قرار می‌گیرد که با معادلات غیر خطی سرو کار داشته باشیم و مخصوصاً زمانی که بخواهیم شبیه‌سازی را از حالت پایدار شروع کنیم. در این حالت، مقدار اولیه متغیرهای حالت به عنوان حدس اولیه مورد استفاده قرار خواهند گرفت.

۳-۱-۹) بخش معادلات و الگوریتمهای مقدار دهی اولیه

یک روش دیگر برای مشخص کردن مقدار اولیه متغیرها استفاده از بخش معادلات و الگوریتمهای مقدار دهی اولیه است.

```
model C
      Real x;
initial algorithm
      x:=1;
equation
      der(x) = -x + 1;
end C;
```

در حالی که در مدل A (در بخش ۱-۱-۹) مقدار شروع x را می‌توان با یک تصحیح به سادگی تغییر داد (تصحیح اغلب از خارج از مدل به آن اعمال می‌گردد)، در مدل C، در شروع شبیه‌سازی همیشه مقدار x برابر ۱ خواهد بود و نمی‌توان آن را تصحیح نمود. از آنجایی که می‌توان مقدار اولیه را با استفاده از معادلات محاسبه نمود، بخش معادلات اولیه از خاصیت مقدار اولیه بسیار قویتر است. برای مثال در مدل D، مقدار اولیه متغیر x در شروع شبیه‌سازی حالت پایدار خواهد داشت چون از عبارت der(x)=0 در بخش معادلات اولیه استفاده شده است.

```
model D
      Real x;
      Real y(start=3);
initial equation
      der(x)=0;
equation
      der(x) = -x + y + 1;
      der(y)= -y +2;
end D;
```

۲-۹) اتفاقات[1]

یک اتفاق زمانی در شبیه‌سازی ایجاد خواهد شد که برای مثال مقدار عبارت شرطی دستور if تغییر کند. وقتی یک اتفاق رخ می‌دهد، حل کننده متوقف شده و سعی می‌کند تا زمان دقیق اتفاق را بیابد. این سعی کردن ممکن است زمانبر باشد و پیشنهاد می‌گردد تا حد ممکن تعداد آنها را کاهش داد.

یک روش برای جلوگیری از سعی‌های غیر ضروری استفاده از عملگر noEvent است. این عملگر به حل‌کننده خواهد گفت که نیازی به ایجاد یک اتفاق نیست. این عملگر فقط زمانی قابل استفاده است که متغیر در زمان اتفاق پیوسته باشد (لزومی به مشتق پذیر بودن متغیر در این زمان نیست).

```
model EventTest
      Real x1;
      Real x2;
equation
      x1=if time<3 then time else 3 "Event generated at
      time=3";
      x2=noEvent(if time<3 then time else 3) "No event
      generated";
end EventTest;
```

همچنین می‌توان از عملگر noEvent برای جلوگیری از ارزیابی‌های غیرمجاز استفاده نمود. مثلاً برای جلوگیری از تقسیم بر صفر، برای حصول اطمینان از عدم ارزیابی عبارت sin(x)/x در نقطه x=0 از عملگر noEvent در حاشیه abs(x)>0 استفاده شده است.

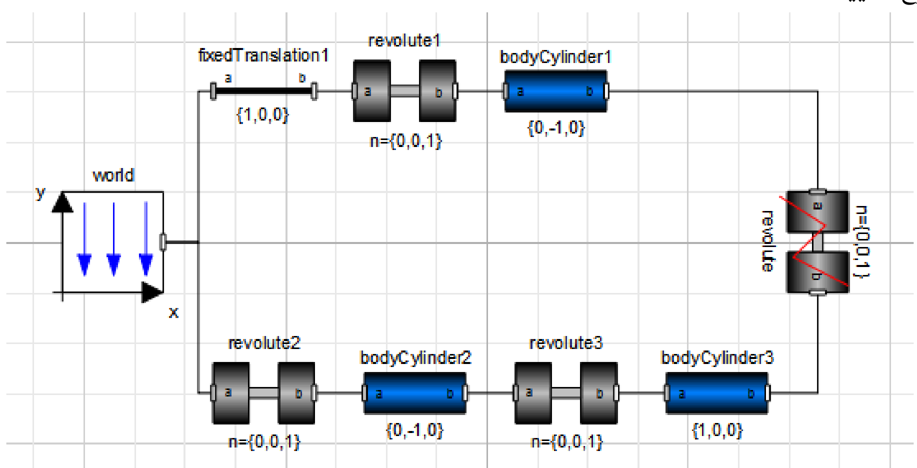
```
model GuardEval
      Real x;
      Real y;
equation
      x=time - 1;
```

[1] Events

```
    y=if noEvent(abs(x) > 0) then sin(x)/x else 1;
end GuardEval;
```

۹-۳) کتابخانه MultiBody

کتابخانه Modelica.MultiBody از کتابخانه‌های Modelica برای مدلسازی سیستمهای مکانیکی سه بعدی است. استفاده از این مدلهای سه بعدی احتیاج به دقت بیشتری دارد زیرا همه قطعات بایستی در فضا به یکدیگر متصل شوند و مقدار اولیه بایستی صحیح باشد تا سیستم شروع به شبیه‌سازی نماید. نرم‌افزار SystemModeler امکانات دیدن مدلهای خلق شده توسط این کتابخانه را به صورت سه بعدی فراهم نموده است تا کاربر بتواند به راحتی حرکت اجزاء را تحلیل نماید. در بخش زیر برخی از موارد مهمی که هنگام کار با کتابخانه MultiBody بایستی رعایت شود ذکر گردیده است. برای مطالعه بیشتر و کار با این کتابخانه می‌توانید با مثالهای موجود در این کتابخانه شروع نمایید.



شکل ۱-۹. مثال یک حلقه مسطح.

۹-۳-۱) مقدار اولیه

برای داشتن یک مدل فیزیکی صحیح لازم است که کاری کرد تا سیستم متغیرهای حالت را به درستی مقدار دهی نماید. در این مثال با استفاده مساوی قراردادن خاصیت stateSelect با StateSelect.always متغیر revolute3، این متغیر را به عنوان متغیر حالت انتخاب شده است. وقتی این مدل ترجمه گردد، متغیرهای حالت مناسب توسط نرم‌افزار انتخاب خواهند شد. مقدار اولیه این متغیرهای حالت را می‌توانید در برگ Variables در Si,ulation Center ببینید.

PlanarLoop 1*

Plot | Parameters | Variables | Settings

Find

Find Options

Name	Initial Value	Unit	Description
revolute3			
phi	1.56	rad	Relative rotation angle from frame_a to frame_b
w	0.0	ra...	First derivative of angle phi (relative angular velocity)

شکل ۲-۹. متغیرهای حالت انتخاب شده برای مدل شکل قبل.

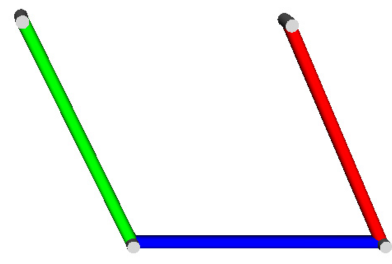
همان طور که در شکل ۲-۹ مشاهده می‌کنید، مقدار زاویه اتصال revolute1 با سرعت 0.0 rad/s و زاویه 1.56 rad شروع خواهد شد. این مقادیر را می‌توان مستقیماً در برگه Variable نرم‌افزار Model Center تغییر داد.

۲-۳-۹) زاویه و موقعیت شیء

خیلی مهم است که مدل از نظر فیزیکی ساختار درستی داشته باشد، در غیر این صورت احتمال عدم موفقیت شبیه‌سازی زیاد است. سعی کنید از جهت‌های ساده برای اجزائی مانند BodyCylinder و BodyBox استفاده نمایید، برای مثال r={1,0,1} و r={0,-1,0} . اگر جهت دیگری مورد نیاز است، phi (زاویه) مفصل را تغییر دهید یا از اجزائی مانند FixedRotation و FixedTranslation که در کتابخانه MultiBody.Parts وجود دارد استفاده نمایید.

۳-۳-۹) پویا نمایی

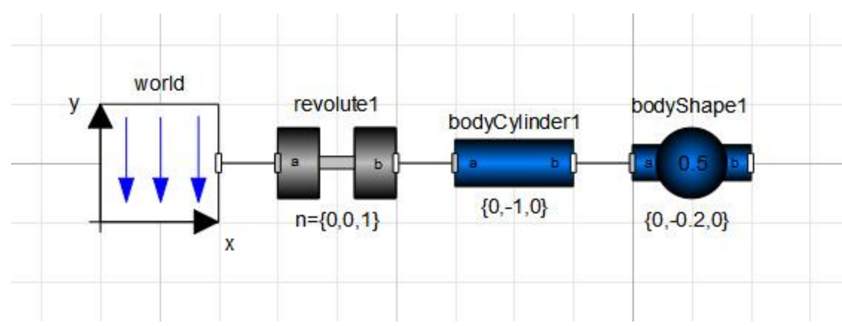
برخی از اجزاء کتابخانه MultiBody دارای اطلاعاتی در خصوص پویا نمایی هستند. این اجزاء عبارتند از Body، BodyBox، BodyCylinder و BodyShape و در کتابخانه MultiBody.Parts قرار دارند. همچنین بخش‌های تجمسی وجود دارد که خواصی مانند سرعت و شتاب دارند، مانند: MultiBody.Visualizers.



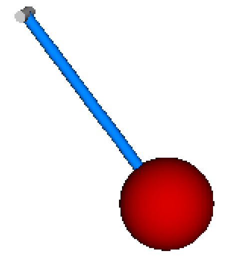
شکل ۳-۹. پویا نمایی مدل یک حلقه ساده شامل سه میله.

همانطور که از نام این اجزاء مشخص است، BodyBox به شکل یک جعبه و BodyCylinder به شکل یک استوانه است. اگر احجام دیگری مانند کره، مخروط، چرخ و ... مورد نیاز باشد، می‌توان از BodyShape با تنظیم shapeType به شکل مورد نظر استفاده نمود. اگر صرفاً پویا نمایی بدون خواص فیزیکی مد نظر باشد می‌توان از Shape استفاده نمود.

شکل ۴-۹یک آونگ را نشان می‌دهد که یک کره به انتهایش متصل است و از قطعه BodyShape برای پویا نمایی کره استفاده شده است.



شکل ۴-۹. مدل یک آونگ با یک کره در انتهایش.

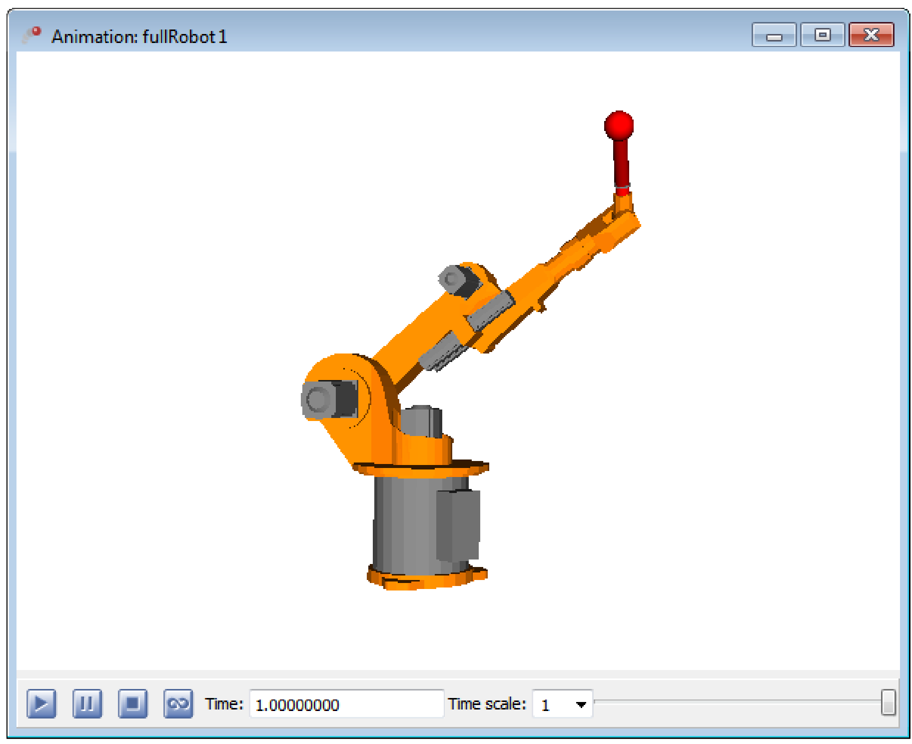


شکل ۵-۹. پویا نمایی ایجاد شده توسط مدل بالا.

۴-۳-۹) استفاده از شکل‌های CAD

در مستندات کتابخانه MultiBody بیان شده است که می‌توان با استفاده از عبارت shapeType "1", "2", ..., "n" شکل‌های خارجی به صورت فایلهایی با قالب DXF نیز استفاده نمود. در این دستور فرض شده است نام فایلها عبارتند از "1.DXF"، "2.DXF" و ... و "n.DXF" . اضافه بر این نرم‌افزار SystemModeler از اطلاعات CAD به صورت فایلهایی با قالب OBJ نیز پشتیبانی می‌کند. همچنین می‌توان به جای عدد از هر اسم فایلی استفاده نمود. اگر آزمایش ایجاد شده ذخیره شده باشد، نرم‌افزار پوشه محل ذخیره را به دنبال فایلهای CAD جستجو خواهد نمود.

در صورتی که جستجو بی‌نتیجه باشد، پوشه محل ذخیره مدل جستجو خواهد شد. یک روش دیگر برای آدرس دهی فایل CAD مورد نظر، مشخص کردن مسیر کامل فایل CAD می‌باشد.



شکل ۶-۹. نمای پویا نمایی
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot.

۴-۹) پیشنهادات عمومی

پیشنهادات زیر می‌تواند باعث جلوگیری از برخی مشکلات معمول گردد:

- روش خطایابی در زبانهای بر پایه معادلات (مثل Modelica) با روش معمول در زبانهای برنامه نویسی متفاوت است. در این زبانها امکان استفاده از Break Point و محاسبه دستور به دستور وجود ندارد. بنابراین سعی کنید مدل را قدم به قدم ایجاد کنید، برای مثال برای هر جزء از سیستم کامل آزمایش‌های کوچکی ایجاد کنید. اگر خطایی ایجاد شد، یافتن خطا در یک سیستم کوچک بسیار آسانتر است.

- استفاده از گزینه ارزیابی کلاس در هنگام ساخت هر قطعه و آزمایش کردن هر قطعه به صورت مجزا بسیار مفید است. اگر ارزیابی یک قطعه موفقیت‌آمیز باشد، احتمال عملکرد صحیح قطعه در کنار سایر قطعات بیشتر است. یک قطعه کامل بایستی همیشه تعداد معادله و متغیر یکسان را در مرحله ارزیابی نمایش دهد. توجه داشته باشید مدلهای جزئی طبیعتاً تعداد معادله و متغیر متفاوتی دارند.
- اگر شبیه‌سازی زمان زیادی می‌گیرد، ممکن است به علت وجود تعداد زیاد اتفاقات باشد. بعد از پایان شبیه‌سازی می‌توانید تعداد اتفاقات را در log شبیه‌سازی ببینید. در صورت امکان سعی نمایید با استفاده از عملگر noEvent از تعداد اتفاقات بکاهید یا مدل خود را تغییر دهید. توجه نمایید که وقتی یک سیستم نمونه‌گیری را مدلسازی می‌نمایید، ایجاد یک اتفاق در هر نمونه زمانی طبیعی است.
- سعی کنید وقتی یک سیستم کاملاً مشخص را شبیه‌سازی می‌نمایید از تنظیم بازه‌های خروجی اتومات خودداری نمایید. چون در این حالت داده‌ها در هر قدم حل در فایل خروجی نوشته خواهد شد و می‌تواند به طرز وحشتناکی باعث کاهش کارایی شبیه‌سازی برای سیستمهای بزرگ گردد. وقتی رفتار سیستم مشخص است، پیشنهاد می‌گردد از بازه‌های خروجی ثابت استفاده نمایید یا از بازه طولی ثابت به جای آن استفاه نمایید.

بخش دوم
محیط OpenModelica

فصل ۱۰ نرم افزار OpenModelica

نرم‌افزار OpenModelica یک نرم‌افزار Open Source برای مدل‌سازی و شبیه‌سازی بر اساس زبان Modelica می‌باشد. یکی از بهترین روشهای یادگیری زبان Modelica آشنایی با این نرم افزار است و به همین دلیل استفاده از آن روز به روز در حال افزایش است. در این بخش برخی از توانایی‌های این محیط را مورد بررسی قرار خواهیم داد. ابتدا با محیط ساده این نرم‌افزار آشنا خواهیم شد و شبیه‌سازی‌های مختلفی را در این محیط انجام خواهیم داد. سپس با محیط‌های گرافیکی مختلفی که امکان برقراری ارتباط با این زبان را دارند آشنا می‌شویم.

هدف کوتاه مدت از خلق نرم‌افزار OpenModelica ایجاد یک محیط محاسباتی کارآمد برای زبان Modelica است. محیطی که در آن امکانات کامل این زبان پیاده‌سازی شده باشد. بدین منظور این نرم‌افزار ابزارها و کتابخانه‌های مناسبی را پشتیبانی می‌کند. زبان Modelica یک زبان بسیار مناسب برای ایجاد و اجرای الگوریتم‌های محاسباتی سطح بالا مانند طراحی سیستم‌های کنترل، حل معادلات غیرخطی و یا بهینه‌سازی الگوریتم‌هایی است که در سیستمهای پیچیده استفاده می‌شود.

هدف بلند مدت خلق این نرم‌افزار، پیاده‌سازی کامل زبان Modelica در یک محیط برنامه‌نویسی می‌باشد. هدف دیگر، ایجاد امکانات مناسب برای تحقیق و آزمایش در زمینه طراحی زبان[1] Modelica یا فعالیت‌های تحقیقاتی دیگر می‌باشد. با این وجود هدف OpenModelica، رسیدن به سطح کارایی و کیفیت محیط‌های تجاری زبان Modelica که در پیاده‌سازی مدل‌های پیچیده به کار می‌روند، نمی‌باشد.

در نرم‌افزار Open Modelica اغلب عبارات، الگوریتم‌ها و امکانات معادلاتی زبان Modelica پیاده-سازی شده است. در این محیط معادلات و توابع مدل‌ها ابتدا به کدهای C ترجمه می‌شود و در نهایت کدهای C تولید شده با کتابخانه‌ای از توابع سودمند، کتابخانه run time و یک حل کننده عددی DAE ترکیب می‌گردد.

[1] language design

۱-۱۰) OMNotebook و DrModelica

ابتدا با کاربردی‌ترین بخش نرم‌افزار OpenModelica یعنی OMNoteBook و DrModelica آشنا خواهیم شد. این فصل شامل بخش‌های زیر می‌باشد:

- ویرایشگر متن OpenModelica که OMNotebook نامیده می‌شود.
- سیستم آموزشی DrModelica که از OMNotebook برای آموزش Modelica استفاده می‌کند.

۲-۱۰) ویرایشگر متن با برنامه نویسی Literate

ویرایشگر متن با برنامه‌نویسی Literate، شامل اسناد پویایی است که این اسناد می‌تواند دارای محاسبات فنی، متن و گرافیک باشد. استفاده از این نوع ویرایشگر متن برای فعالیت‌های آموزشی، آزمایشی، شبیه‌سازی، کدنویسی، مستندسازی، ذخیره مدل و ... بسیار مناسب می‌باشد.

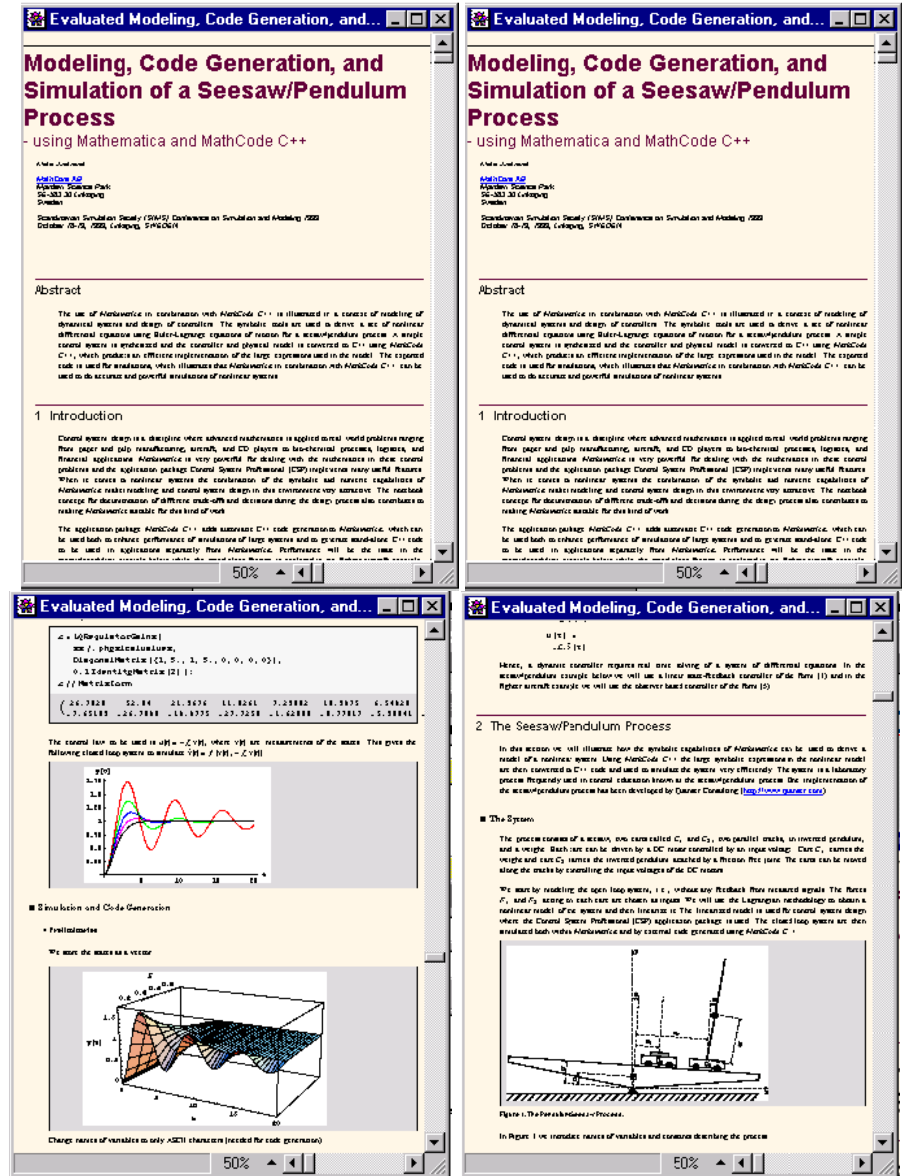
۱-۲-۱۰) ویرایشگر متن OMNotebook و Mathematica

برنامه‌نویسی Literate (Knuth, 1984) نوعی از برنامه‌نویسی است که در آن برنامه‌ها همراه با متون، محاسبات، شکل‌ها و ... در یک سند جمع‌آوری می‌شود. ویرایشگر متن Mathematica (Wolfram, 1997) یکی از اولین سیستم‌های WYSIWYG [1] است که برنامه‌نویسی Literate را پشتیبانی می‌کند. اگر نرم‌افزارهای Mathematica و MathModelica را در سیستم خود داشته باشید، می‌توانید نمونه چنین اسنادی را در مثال‌های این نرم‌افزارها ببینید. ویرایشگر متن MathModelica یک ابزار پیشرفته برای نرم‌افزار Mathematica می‌باشد که به جز امکان نوشتن فرمولهای ریاضی، امکانات زیاد دیگری را نیز برای آن فراهم می‌آورد.

نرم افزار OMNotebook چنین ویرایشگر متنی است که البته امکانات ساده‌تری نسبت به MathModelica دارد. این نرم افزار محیطی را برای پشتیبانی از زبان Modelica فراهم می‌نماید.

---

[1] What-You-See-Is-What-You-Get



شکل ۱-۱۰. مثال هایی از دفتر یادداشت‌های Mathematica در محیط مدل‌سازی و شبیه‌سازی MathModelica.

مستندات قدیمی مانند کتاب‌ها و گزارشات، همواره یک ساختار سلسله مراتبی دارند. آنها به بخش‌ها، زیر بخش‌ها، پاراگراف‌ها و غیره تقسیم می‌شوند. همه این مستندات برای دستیابی آسان‌تر دارای سر فصل می‌باشند. این ساختار در دفتر یادداشت‌های الکترونیکی هم وجود دارد. هر یادداشت مربوط به یک فایل می‌باشد و شامل یک ساختار درختی از سلول‌ها است. یک سلول می‌تواند حاوی

محتویات متفاوتی باشد و حتی دیگر سلول‌ها را در برگیرد. سلسله مراتب‌های سلولی در این دفتر یادداشت‌ها یادآور بخش‌ها و زیر بخش‌ها در اسناد قدیمی مانند کتاب‌ها می‌باشد. شکل ۲-۱ نمونه‌ای از دفتر یادداشتهای Mathematica در محیط مدلسازی و شبیه سازی MathModelica را نشان می‌دهد.

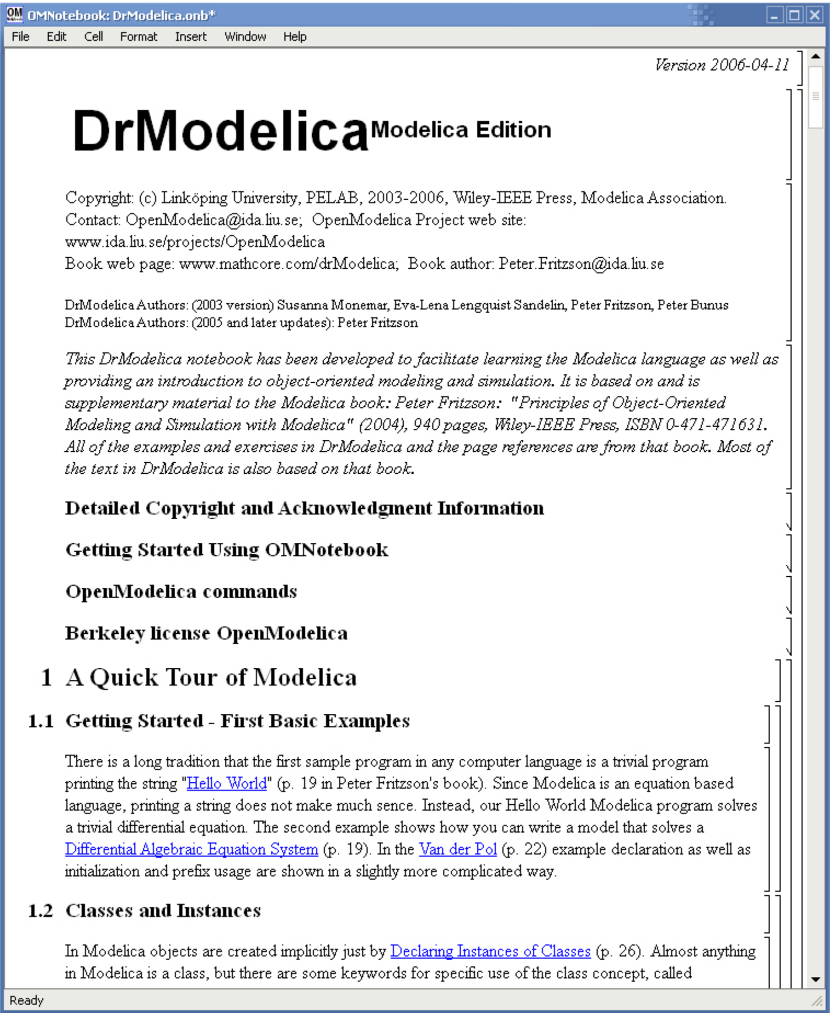
۱۰-۳) سیستم آموزشی DrModelica

درک کدها کار سختی می‌باشد. به خصوص کدهایی که به وسیله فرد دیگری نوشته شده است. برای اهداف آموزشی لازم است کدهای برنامه و توضیحاتی در مورد آنها در یک زمان نشان داده شود. به علاوه مهم است که نتیجه اجرای کدها نشان داده شود.

در مدل‌سازی و شبیه‌سازی نیز مهم است که کدهای اصلی، مستندات و توضیحات کدها، نتایج شبیه‌سازی مدل و مستندات نتایج شبیه‌سازی را در یک سند داشته باشیم. دلیل این نیاز به فرایند حل مسئله در شبیه‌سازی محاسباتی باز می‌گردد. حل مسئله یک فرایند تکراری است. اغلب بعد از تفسیر و ارزیابی نتایج محاسبات نیاز داریم مدل ریاضی اصلی را اصلاح کنیم و آن را دوباره اجرا نماییم. اگر بتوان همه مستندات، مدل ریاضی و نتایج شبیه سازی را با هم در یک برنامه داشته باشیم فرایند حل مسئله بسیار سریعتر انجام خواهد شد.

اکثر محیط‌های مدل‌سازی مبتنی بر معادلات، بر فراهم نمودن الگوریتم‌های عددی کارآمد متمرکز شده‌اند و به تسهیل روند آموزش و یادگیری زبان برنامه‌نویسی کمتر توجه شده است. این مطلب دلیل ایجاد DrModelica برای آموزش مدل‌سازی و شبیه‌سازی با استفاده از Modelica می‌باشد. ویرایش قبلی DrModelica با استفاده از محیط MathModelica ایجاد شده بود، اما در ویرایش جدید DrModelica براساس نرم‌افزار OMNotebook و ابزارهای آن نوشته شده است و در این فصل به آموزش آن خواهیم پرداخت.

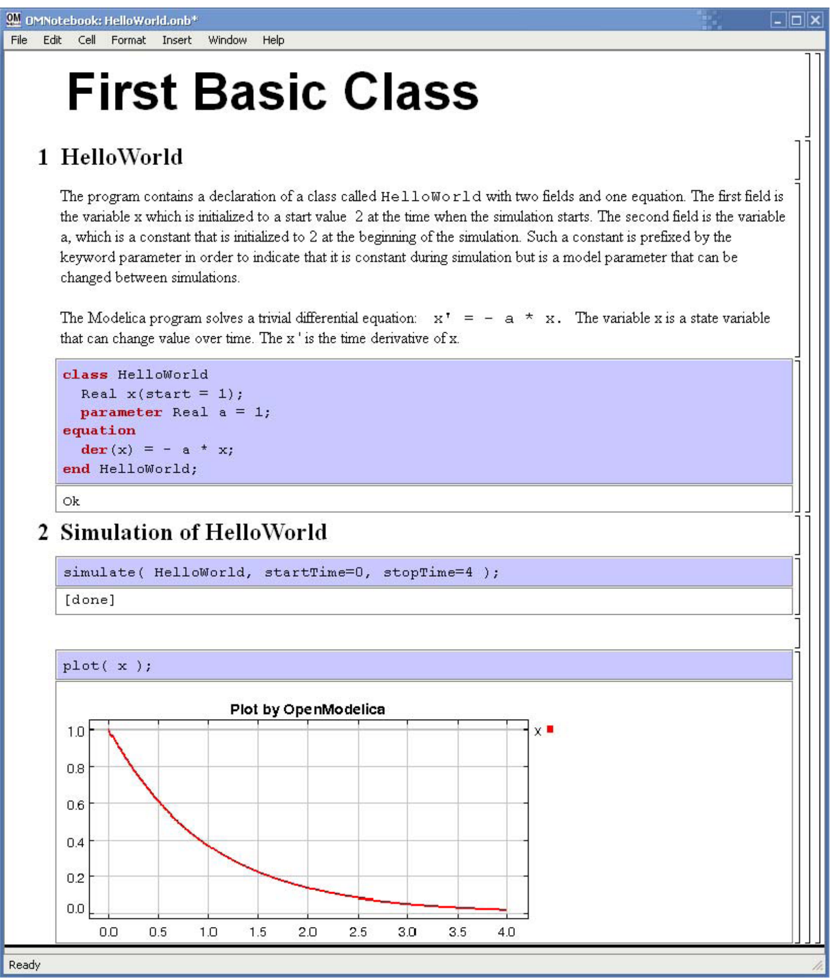
مستندات DrModelica به عنوان یک دفتر یادداشت الکترونیکی، دارای ساختار سلسله مراتبی می‌باشد. در یادداشت صفحه اول، فهرستی از مطالب موجود در کل صفحات به همراه لینک‌های آنها وجود دارد. این یادداشت ویژه، اولین صفحه‌ای است که کاربر آن را خواهد دید، که برای دیدن این صفحه، نرم افزار OMNoteBook را اجرا نمایید.



شکل ۲-۱۰..اولین صفحه OMNotebook از سیستم آموزشی DrModelica.

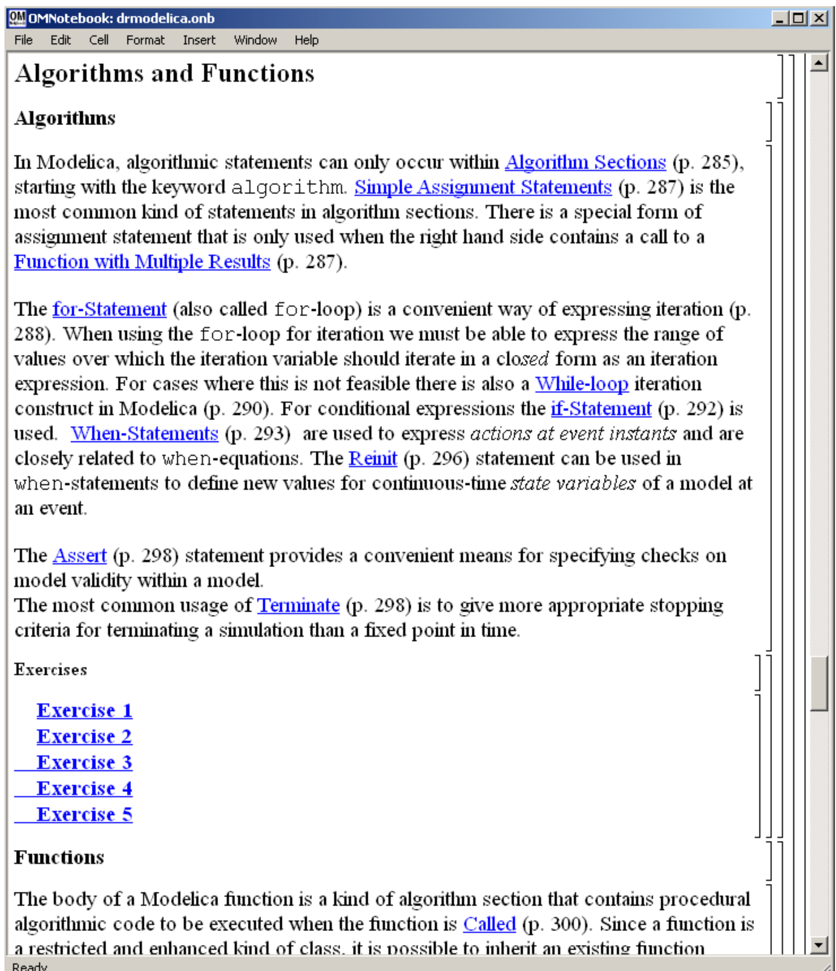
در هر فصل از DrModelica، یک خلاصه کوتاه از فصل‌های مرتبط از کتاب "Principles of Object-Oriented Modeling and Simulation with Modelica 2.1" که توسط Peter Fritzon نوشته شده، به کاربر نشان داده می‌شود. این کتاب تا زمان نوشته شدن این متن بهترین کتاب موجود برای آموزش زبان modelica می‌باشد. این خلاصه که در شکل ۲-۱۰ نشان داده شده است تعدادی از کلمات کلیدی را معرفی می‌کند که با تلیک بر روی آنها کاربر به سمت یادداشت‌های دیگر که جزئیات کلمات کلیدی را تشریح می‌کنند، راهنمایی می‌شود.

حال در DrModelica لینک "HelloWorld" را یافته و تلیک کنید. با این کار دفتر یادداشت جدید HelloWorld باز خواهد شد (شکل ۳-۱۰). در این یادداشت مانند اولین مثال این کتاب یک معادله دیفرانسیل شبیه سازی شده است.



شکل ۳-۱۰. کلاس HelloWorld شبیه سازی شده و طراحی شده با استفاده از DrModelicaی ویرایش OMNoteboo .

اطلاعات موجود در یادداشت‌ها ثابت نیست و کاربر می‌تواند چیزهایی را اضافه کرده، تغییر دهد یا حذف کند. همچنین کاربر می‌تواند یک یادداشت کاملاً جدید برای نوشتن کدهای خودش یا کپی برداشتن از مثال‌های موجود در سایر یادداشت‌ها ایجاد کند. این یادداشت جدید می‌تواند به سایر یادداشت‌ها لینک شود.



شکل ۴-۱۰.فصل مربوط به الگوریتم‌ها و توابع در صفحه اصلی DrModelica.

حال که اولین مثال DrModelica را باز کردیم خوب است آنرا دوباره شبیه سازی نماییم. برای این کار در اولین پنجره بنفش رنگ که در آن کد برنامه نوشته شده است تلیک کنید، از این پس این پنجره را سلول ورودی می‌نامیم. روی آیکون Evaluate ( ) تلیک کنید یا از میانبر ctrl+Enter استفاده نمایید. این کار باعث ترجمه کلاس HelloWorld و ایجاد نمونه قابل شبیه-سازی از این کلاس خواهد شد. همین عملیات را برای سلول ورودی دوم انجام دهید تا شبیه سازی از زمان startTime تا stopTime انجام گیرد. با توجه به سخت افزار و قدرت کامپیوتر شما این شبیه سازی ممکن است چند ثانیه طول بکشد. برای مشاهده مقدار x در سلول ورودی بعدی دستور plot(x) وارد شده است. این دستور را به روش قبل اجرا نمایید. نمودار x رسم خواهد شد. از منوی Cell گزینه Add InputCell را انتخاب نمایید یا از میانبر ctrl+shift+I برای ایجاد یک سلول

ورودی جدید استفاده نمایید. در این سلول دستور plot2(x) را بنویسید و آن را اجرا کنید. با این کار نمودار x در دو پنجره شامل یک پنجره اصلی و یک پنجره جدید باز خواهد شد. به همین سادگی شما اولین شبیه‌سازی خود را با استفاده از محیط OpenModelica انجام دادید.

همانطور که در مثال قبل مدل ساده HelloWorld نشان داده شد، زمانی که یک کلاس با موفقیت ارزیابی شد، کاربر می‌تواند آن را شبیه سازی و نتایج آن را ترسیم کند.

بعد از خواندن هر فصل از DrModelica، شما می‌توانید اطلاعات جدید به دست آمده مرتبط را با انجام مثال‌های آن فصل تمرین کنید. این تمرین‌ها برای شفاف کردن ساختار زبان به صورت مرحله به مرحله نوشته شده است. انجام تمرین باعث درک عمیق‌تر مطالب خواهد شد. این تمرینات حاوی سوالات تئوری یا عملی برنامه نویسی می‌باشند.

قبل از ادامه این بخش مناسب است با بعضی از امکانات و اصطلاحات OMNotebook آشنا شوید.

۱-۳-۱۰) سلول‌ها[1]

همه چیزهایی که درون سند OMNotebook می‌باشند از سلول‌ها ساخته شده‌اند. اساساً هر سلول شامل مجموعه‌ای از اطلاعات است. این داده‌ها می‌تواند عکس، متن و یا سلول‌های دیگر باشند. OMNotebook دارای چهار نوع سلول می‌باشد: Headercell، Textcell، Inputcell و Groupcell. سلول‌ها به صورت یک ساختار درختی مرتب می‌شوند که در آن یک سلول می‌تواند شامل یک یا چند سلول دیگر باشد.

- سلول متنی یا Textcell

از این سلول برای نمایش متن و عکس‌های معمولی استفاده می‌شود. هر Textcell دارای سبکی است که تعیین کننده چگونگی نمایش متن است. سبک این سلول با استفاده از منوی Format->Styles می‌تواند تغییر کند که سبک‌های مختلف این نوع سلول عبارتند از text، title و subtitle. همچنین Textcell قابلیت ارتباط دادن به سایر اسناد را نیز دارا می‌باشد.

- سلول ورودی یا Inputcell

این نوع سلول از نظر گرامری دستورات را ارزیابی می‌کند. از این رو برای نوشتن کدهای برنامه مانند کد Modelica استفاده می‌شود. با فشار دادن کلیدهای ترکیبی shift+Enter کار ارزیابی انجام می‌شود. تمامی متن‌های این سلول به مترجم

[1] cells

OpenModelica(که به نام OMC[1] شناخته می‌شود) فرستاده خواهد شد و در آنجا ارزیابی شده و نتیجه آن زیر سلول ورودی نمایش داده می‌شود. با دوبار تلیک کردن بر روی علامت سلول در نمای درختی Inputcell، نتایج آن را می‌توان مخفی نمود.

- سلول گروهی Groupcell

این نوع سلول برای گرد آوری دیگر سلول‌ها استفاده می‌شود. یک Groupcell می‌تواند باز یا بسته باشد. وقتی که یک Groupcell باز است، تمامی سلول‌های درون آن قابل رویت هستند، اما زمانی که Groupcell بسته است، تنها اولین سلول درون آن قابل رویت می‌باشد. هنگامی که کاربر بر روی علامت سلول در نمای درختی دو بار تلیک می‌کند، وضعیت Groupcell تغییر می‌یابد.

۲-۳-۱۰) مکان نماها[2]

یک سند OMNotebook دارای سلول است و در نتیجه شامل متن نیز می‌باشد. بنابراین دو نوع مکان نما برای موقعیت یابی مورد نیاز است که عبارتند از مکان نمای متنی و مکان نمای سلولی.

- مکان‌نمای Textcursor

مکان‌نمای بین حروف می‌باشد که مانند مکان نماهای عمومی نرم‌افزارهای ویرایشگر به شکل یک خط کوچک عمودی در سلول ظاهر می‌شود. با تلیک کردن بر روی متن یا با استفاده از کلیدهای جهت دار می‌توان آن را جابه‌جا نمود.

- مکان‌نمای Cellcursor

این مکان‌نما نشان می‌دهد که در حال حاضر کدام سلول فعال می‌باشد. مکان نمای اصلی یک خط سیاه افقی کلفت در قسمت زیرین سلول می‌باشد. با تلیک کردن بر روی یک سلول یا تلیک کردن بین سلول‌ها یا استفاده از منوی Cell>>Next Cell یا Cell>>Previous Cell می‌توان محل این مکان نما را تعیین نمود. همچنین با کلیدهای ترکیبی ctrl+Up و ctrl+Down می‌-توان محل این مکان‌نما را تغییر داد.

۴-۱۰) ایجاد مدل مخزن

برای یادگیری بهتر این محیط با توسعه مثال مخزن فصل قبل شروع می‌کنیم. این کار را با شناخت قدم به قدم مدل مخزن موجود در محیط OMNotebook انجام خواهیم داد. سپس این

[1] OpenModelica Compiler/Interpreter
[2] Cursors

مدل را توسعه می‌دهیم. مدل مخزن ساده را با تلیک بر روی لینک دوم موجود در بخش 12-1 Modeling a Tank System دفترچه DrModelica باز می‌کنیم. عنوان این مبحث Using the Object Oriented Component-Based Approach است.

ابتدا لازم است این یادداشت را مطالعه نماییم. این یادداشت را مانند مثال مخزن بخش اول از ایجاد درگاه‌ها شروع می‌کنیم. ترتیب کلاسهای تعریف شده در یادداشت اصلی کمی متفاوت است. می‌توانید برای درک بهتر قدم به قدم این کلاسها را در یک یادداشت جدید ایجاد نمایید. برای ایجاد یک یادداشت جدید از گزینه File>>New استفاده نمایید و برای ایجاد سلولهای محاسباتی از منو Cell>>Insert Inputcell را انتخاب نمایید.

ابتدا با خواندن سطح مخزن از درگاه سیگنال خروجی شروع می‌کنیم. یک درگاه به نام RealSignal ایجاد شده است که یک متغیر به نام val از نوع حقیقی را انتقال می‌دهد. با توجه به این که زبان Modelica امکان کنترل واحدهای مهندسی را دارد و بسیاری از این واحدها را می-شناسد، در این تعریف نیز برای متغیر val واحد متر در نظر گرفته شده است. استفاده از واحد باعث اشکال زدایی سریعتر مدلسازی خواهد شد. یک عادت مناسب در مدلسازی حرفه‌ای مشخص نمودن واحد متغیرها است.

```
connector ReadSignal "Reading fluid level"
	Real val(unit = "m");
end ReadSignal;
```

سیگنال عملگر شیر را مطالعه نمایید. این درگاه مقدار باز و بسته بودن شیر را تنظیم خواهد نمود.

```
connector ActSignal "Signal to actuator for setting valve position"
	Real act;
end ActSignal;
```

حال درگاه سیال را مطالعه نمایید. این درگاه جریان سیال گذرنده را مشخص می‌نماید.

```
connector LiquidFlow "Liquid flow at inlets or outlets"
	Real lflow(unit = "m3/s");
end LiquidFlow;
```

قبل از بررسی مخزن، تابع محدود کننده وضعیت شیر را مشاهده نمایید. این تابع برای محدود نمودن جریان عبوری از شیر در یک بازه، مطابق شیر کاملاً باز یا شیر کاملاً بسته استفاده می‌گردد.

```
function limitValue
	input Real pMin;
	input Real pMax;
	input Real p;
	output Real pLim;
algorithm
	pLim := if p>pMax then pMax
	else if p<pMin then pMin
	else p;
```

```
end limitValue;
```

توجه نمایید که در تعریف تابع، متغیرهای ورودی و خروجی به صورت کاملاً واضح مشخص شده‌اند. همچنین در تعریف تابع به جای equation از algorithm استفاده می‌گردد. در توابع فقط استفاده از الگوریتم مجاز است. معادلات نوشته شده در بخش الگوریتم مانند زبانهای برنامه‌نویسی سنتی، به صورت دستور به دستور اجرا خواهد شد. استفاده از بخش الگوریتم در مدلها نیز مجاز است. به نحوه نوشتن دستور if…then…else توجه نمایید.

مدل نوشته شده برای مخزن را به صورت زیر مشاهده می‌نمایید.

```
model Tank
	ReadSignal tSensor "Connector, sensor reading tank level (m)";
	ActSignal tActuator "Connector, actuator controlling input flow";
	LiquidFlow qIn "Connector, flow (m3/s) through input valve";
	LiquidFlow qOut "Connector, flow (m3/s) through output valve";
	parameter Real area(unit = "m2") = 0.5;
	parameter Real flowGain(unit = "m2/s") = 0.05;
	parameter Real minV= 0, maxV = 10; // Limits for output valve flow
	Real h(start = 0.0, unit = "m") "Tank level";
equation
	assert(minV>=0,"minV - minimum Valve level must be >= 0 ");
	der(h) = (qIn.lflow - qOut.lflow)/area; // Mass balance equation
	qOut.lflow = limitValue(minV, maxV, -flowGain*tActuator.act);
	tSensor.val = h;
end Tank;
```

در چهار دستور اول این مدل درگاههایی به نام tSensor، tActuator، gIn و gOut تعریف شده است. توضیحات نوشته شده بعد از هر درگاه جزئی از ساختار تعریف محسوب می‌گردد. این توضیحات موجب خوانایی بیشتر مدل خواهد شد. دستورات پنجم تا هفتم سه پارامتر را از نوع Real تعریف نموده‌اند. مقدار پارامترها برعکس متغیرها در طول شبیه‌سازی ثابت است. فقط در ابتدای شبیه‌سازی به پارامترها مقدار اولیه داده می‌شود. مشخص نمودن این مقدار اولیه الزامی است. تعریف پارامتر به شکل زیر است:

parameter type name(**unit** = "unit") = value;

برای اضافه نمودن توضیحات به متن کد می‌توان مانند زبان C++ از علامت // قبل از توضیحات استفاده نمود. دستور assert برای تعریف محدودیت متغیرهای مساله به کار می‌رود. مثلاً در مدل بالا دستور assert مشخص می‌نماید که حداقل مقدار minV می‌تواند صفر باشد. دستورات دیگر کاملاً مشابه مدل نوشته شده در بخش گذشته است.

بعد از مدل مخزن، مدل منبع سیال را ایجاد خواهیم نمود:

```
model LiquidSource
	LiquidFlow qOut;
	parameter Real flowLevel = 0.02;
```

```
equation
	qOut.lflow = if time > 150 then 3*flowLevel else flowLevel;
end LiquidSource;
```

مانند مثال گذشته برای ایجاد مدل کنترل کننده PI، ابتدا مدل کنترل کننده پایه را ایجاد می‌نماییم و کنترل کننده‌های دیگر را از آن مشتق می‌نماییم.

```
partial model BaseController
	parameter Real Ts(unit = "s") = 0.1 "Time period between discrete
	samples";
	parameter Real K = 2 "Gain";
	parameter Real T(unit = "s") = 10 "Time constant";
	ReadSignal cIn "Input sensor level, connector";
	ActSignal cOut "Control to actuator, connector";
	parameter Real ref "Reference level";
	Real error "Deviation from reference level";
	Real outCtr "Output control signal";
equation
	error = ref - cIn.val;
	cOut.act = outCtr;
end BaseController;
```

این مدل با استفاده از عبارت partial در ابتدای تعریف مدل به صورت جزئی تعریف شده است. مترجم زبان Modelica همه مدلها را برای داشتن تعداد معادلات و مجهولات برابر کنترل می‌نماید و اگر تعداد معادلات و مجهولات برابر نباشد مترجم خطایی ایجاد خواهد نمود. در مدلهای جزئی کنترل تعداد صورت نمی‌گیرد. مدل جزئی به عنوان پایه برای ایجاد مدلهای دیگر کاربرد دارد. مدلهای مشتق شده، همه متغیرها و معادلات مدل پایه را شامل می‌شوند.

مدل کنترل کننده PI را از مدل پایه ایجاد می‌نماییم. دستور extend در اولین خط این مدل این کار را انجام می‌دهد.

```
model PIcontinuousController
	extends BaseController(K = 2, T = 10);
	Real x "State variable of continuous PI controller";
equation
	der(x) = error/T;
	outCtr = K*(error + x);
end PIcontinuousController;
```

وقتی که تمام کلاسهای لازم برای ایجاد سیستم اصلی ایجاد گردید، مدل اصلی شامل مخزن و تجهیزات دیگر را ایجاد می‌نماییم.

```
model TankPI
	LiquidSource source(flowLevel=0.02);
	PIcontinuousController piContinuous(ref=0.25);
	Tank tank(area=1);
equation
	connect(source.qOut, tank.qIn);
```

```
        connect(tank.tActuator, piContinuous.cOut);
        connect(tank.tSensor, piContinuous.cIn);
end TankPI;
```

در جمله اول یک منبع به نام source از نوع LiquidSource تعریف شده است. به همین ترتیب یک کنترل کننده PI و یک مخزن با نامهای piContinuos و tank تعریف شده است. در بخش معادلات، دستورات موجود این قطعات را به هم وصل می‌کند. به راحتی می‌توان مدل کامل را شبیه‌سازی نمود و نمودار پاسخ سطح مخزن نسبت به تغییرات منبع را رسم نمود.

دستور شبیه سازی مدل برای ۲۵۰ ثانیه را وارد نمایید.

```
simulate( TankPI, stopTime=250 )
```

و سطح مخزن را رسم نمایید.

```
plot( tank.h );
```

۱۰-۵) ایجاد مدل پیشرفته مخزن

برای توسعه یادداشت بالا آن را با نام دیگری ذخیره نمایید، زیرا می‌خواهیم کدهای نوشته شده را ویرایش نماییم. برای ذخیره یادداشت از منو ... File>>Save As استفاده نمایید.

هدف ما شبیه‌سازی سیستم مخزن با یک قطعه جدید به نام شیرکنترل است که رفتاری شبیه شیرکنترل واقعی دارد. در مثالهای گذشته برای سادگی، این قطعه به عنوان بخشی از مخزن در نظر گرفته شده بود؛ اما اکنون به سطحی از دانش رسیده‌ایم که می‌توانیم مدل بهتری از کل سیستم ایجاد نماییم. برای آن که بتوانیم از شیرکنترل استفاده نماییم باید معادله حاکم بر شیرکنترل را تحلیل کنیم. برای استفاده از شیرکنترل باید در درگاه سیال مقدار فشار را در کنار مقدار جریان داشته باشیم. قبل از ایجاد مدل شیر لازم است درگاه سیال را به صورتی تغییر دهیم که فشار را نیز به عنوان یک پارامتر انتقال دهد که در این صورت درگاه‌های دیگر تغییری نمی‌کنند.

```
connector LiquidFlow "Liquid flow at inlets or outlets"
        Real lflow(unit = "m3/s");
        Real lpressure(unit = "Pa");
end LiquidFlow;
```

۱۰-۵-۱) شیرکنترل

شیرکنترل مانند شیر معمولی است با این تفاوت که کنترل مقدار باز و بسته شدن آن به وسیله سیگنال خارجی تعیین می‌گردد. جریان عبوری شیر کنترل مانند سایر شیرها تابعی از چگالی سیال، مقدار باز بودن شیر، مشخصات فیزیکی و فشار سیال دو طرف آن است. در خصوص رفتار شیرهای کنترل به تفصیل در کتاب شبیه‌سازی و مدلسازی قطعات مکانیکی توضیح داده شده است. به طور ساده می‌توان گفت رفتار شیر به شکل معادله ریاضی زیر است:

$$q_{valve} = K_1 Cv \sqrt{\frac{\Delta P}{\rho}} \quad (۱۰-۱)$$

در این معادله Cv ضریب جریان شیر است که بستگی به مشخصات شیر و واحدهای انتخاب شده در محاسبات خواهد داشت. مقدار K_1 نشاندهنده مقدار باز بودن شیر است که $K_1=1$ نشان دهنده شیر کاملاً باز و $K_1=0$ نشان دهنده شیر کاملاً بسته است. $\rho, \Delta P$ به ترتیب اختلاف فشار دو سر شیر و چگالی سیال گذرنده از شیر است. در این مدل چگالی سیال را ثابت در نظر می‌گیریم. مدل شیرکنترل را به شکل زیر بنویسید. این مدل دو درگاه سیال و یک درگاه سیگنال کنترل دارد. به محدودیت جریان عبوری شیر نیز توجه داشته باشید و از تابع محدودیت استفاده نمایید.

```
model ControlValve
	ActSignal cvActuator "Connector, actuator controlling input flow";
	LiquidFlow qIn "Connector, flow (m3/s) through input";
	LiquidFlow qOut "Connector, flow (m3/s) through output";
	parameter Real Cv         ;
	parameter Real rho (unit = "kg/m3") = 1000;
	parameter Real minCtrlSig= 0, maxCtrlSig = 1; // Limits for output
	valve flow
	Real DeltaP(unit = "Pa");
	Real cvPosition;
equation
	qIn.lflow = qOut.lflow; // Mass balance equation
	DeltaP= qIn.lpressure - qOut.lpressure; // Pressure balance
	cvPosition = limitValue(minCtrlSig, maxCtrlSig, cvActuator.act);
	qOut.lflow = Cv * cvPosition * sqrt(DeltaP/rho);
end ControlValve;
```

برای آن که در انتهای شیر نیز فشار مشخصی داشته باشیم لازم است که محلی برای تخلیه سیال خروجی از شیرکنترل وجود داشته باشد. برای ایجاد این تخلیه مانند منبع سیال یک چاهک سیال ایجاد می‌نماییم که فشار مشخصی دارد. فشار این چاهک را ثابت و برابر یک اتمسفر در نظر می‌گیریم (1atm = 100000Pa)

```
model LiquidSink
	LiquidFlow qIn;
	parameter Real SinkPressure(unit = "Pa") = 100000; //Pressure
equation
	qIn.lpressure = SinkPressure;
end LiquidSink;
```

با توجه به اضافه شدن پارامتر فشار به درگاه سیال همه قطعاتی که از این درگاه استفاده می‌کنند، مانند منبع سیال و مخزن باید رابطه‌ای با فشارها داشته باشند. از مدل منبع سیال شروع می‌کنیم، فشار منبع سیال را ثابت و برابر ۱ اتمسفر فرض می‌کنیم:

```
model LiquidSource
	LiquidFlow qOut;
	parameter Real flowLevel = 0.02;
	parameter Real SourcePressure = 100000; //Pressure
equation
	qOut.lflow = if time > 150 then 3*flowLevel else flowLevel;
	qOut.lpressure = SourcePressure;
end LiquidSource;
```

با توجه به این که می‌خواهیم شیرکنترل را به صورت مجزا مدل کنیم باید مدل مخزن را بدون در نظر گرفتن شیرکنترل بنویسیم. از طرفی برای پیدا کردن رابطه فشار با مخزن فرض می‌کنیم که لوله خروجی سیال در کف مخزن قرار دارد. فشار کف مخزن برابر فشار اتمسفر به علاوه فشار آب داخل مخزن است. در این صورت فشار خروجی مخزن به سطح سیال داخل مخزن و چگالی سیال بستگی دارد و هر چه سطح سیال در مخزن (h) بالاتر باشد، فشار خروجی مخزن بیشتر خواهد بود. با فرض ثابت بودن فشار اتمسفر داریم:

$$P_{tank} = \rho g h + P_{atm} \tag{۱۰-۲}$$

مدل نهایی مخزن به شکل زیر خواهد بود:

```
model Tank
	ReadSignal tSensor "Connector, sensor reading tank level (m)";
	LiquidFlow qIn "Connector, flow (m3/s) through input valve";
	LiquidFlow qOut "Connector, flow (m3/s) through output valve";
	constant Real g = 9.8;
	parameter Real area(unit = "m2") = 0.5;
	parameter Real atmPressure = 100000; //Pressure
	parameter Real rho (unit = "kg/m3") = 1000;
	Real h(start = 0.0, min=0.0, unit = "m") "Tank level";
equation
	der(h) = (qIn.lflow - qOut.lflow)/area; // Mass balance equation
	qOut.lperessure = rho * g * h + atmPressure;
	tSensor.val = h;
end Tank;
```

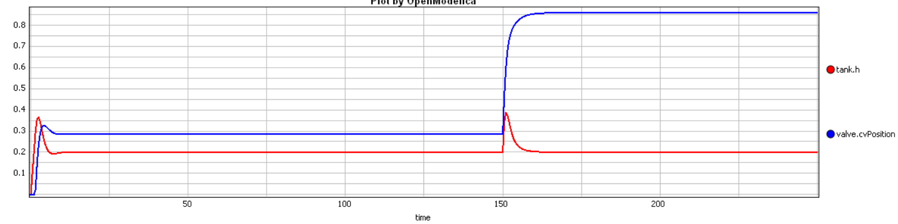
اکنون تمام قطعات آماده هستند، در نتیجه مدل نهایی سیستم را برای شبیه سازی ایجاد می‌-کنیم:

```
model TankPI
        LiquidSource source(flowLevel=0.2);
        LiquidSink sink;
        PIcontinuousController piContinuous(K=-1,T=1,ref=0.2);
        Tank tank(area=1);
        ControlValve valve(Cv=0.5);
equation
        connect(source.qOut, tank.qIn);
        connect(tank.qOut, valve.qIn);
        connect(valve.qOut, sink.qIn);
        connect(tank.tSensor, piContinuous.cIn);
        connect(piContinuous.cOut, valve.cvActuator);
end TankPI;
```

این مدل را شبیه سازی نمایید. بهتر است در پایان این عبارت از (;) استفاده ننمایید تا اگر خطایی در این مرحله دیده شد، گزارش آن قابل مشاهده باشد و دستور را مانند مثال زیر اجرا کنید.

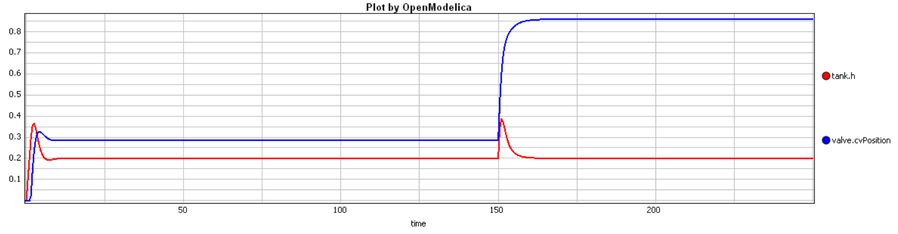
```
simulate(myTank,startTime =0,stopTime=250)
```

نمودار وضعیت شیر و همچنین سطح مخزن را رسم نمایید که در اینجا در



نشان داده شده است.

```
plot({tank.h,valve.cvPosition})
```



شکل ۵-۱۰.نمودار وضعیت شیر و سطح مخزن.

۱۰-۶) آشنایی با OMShell

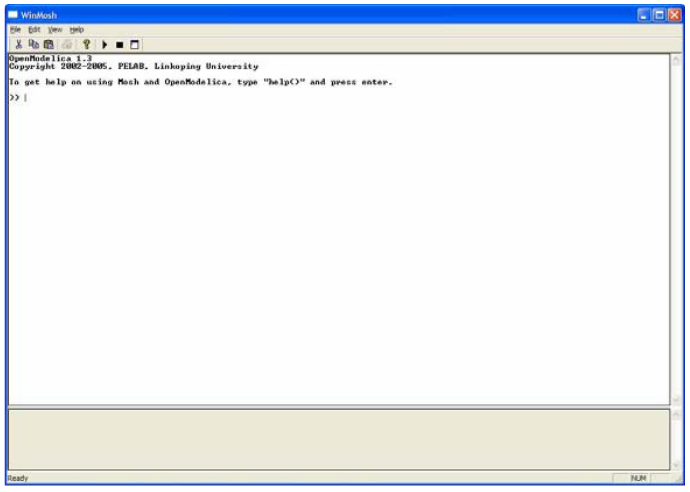
محیط OMShell امکان باز نمودن مدلهای موجود و شبیه سازی آنها را فراهم می نماید. با استفاده از دستورات این محیط می‌توان محاسبات ساده‌ای را انجام داد.

۱۰-۷) بخش Interactive session همراه با مثال

در ادامه با استفاده از Interactive session handler نحوه کار با محیط OpenModelica (OMShell نامیده می‌شود) آموزش داده می‌شود. بیشتر این مثال‌ها در فایل UsersGuideExamples.onb در شاخه testmodels موجود می‌باشد.

۱۰-۷-۱) شروع کار با Interactive session

از منوی Start ویندوز OpenModelica Shell را انتخاب می‌کنیم. پنجره‌ای مطابق شکل ۶-۱۰ ظاهر خواهد شد.



شکل ۶-۱۰. پنجره OpenModelica Shell در محیط ویندوز.

حال عبارت زیر را وارد می‌کنیم که به وسیله آن عبارت 1:12 به صورت دامنه‌ای از اعداد ۱ تا ۱۲ تولید شده که به متغیر x اختصاص داده می‌شوند:

```
>> x := 1:12
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

۱۰-۷-۲) امتحان کردن تابع Bubblesort

تابع bubblesort را با انتخاب از منوی File>>Load Model یا با نوشتن دستور زیر Load کنید:

```
>> loadFile("C:/OpenModelica1.5.0/testmodels/bubblesort.mo")
true
```

در ادامه از تابع bubblesort جهت مرتب کردن بردار x به صورت نزولی استفاده می‌کنیم که در نتیجه بردار مرتب شده به همراه نوع آن بازگردانده می‌شود. توجه کنید که بردارخروجی از نوع Real[:] می‌باشد که خود زیرمجموعه نوع Real[12] می‌باشد. ورودی تابع که برداری از نوع Integer است با استفاده از قوانین مربوط به تبدیل انواع در Modelica به صورت خودکار به نوع Real تبدیل می‌شود. اگر قبلاً تابع ترجمه نشده باشد، به هنگام فراخوانی به صورت خودکار ترجمه می‌شود.

```
>> bubblesort(x)
{12.0,11.0,10.0,9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0,1.0}
```

یک فراخوانی دیگر:

```
>> bubblesort({4,6,2,5,8})
{8.0,6.0,5.0,4.0,2.0}
```

با استفاده از تابع کاربردی system می‌توان دستورات سیستم عامل را در محیط OMShell اجرا کرد.

در سیستم عامل ویندوز، فقط نتیجه دستور system در پنجره‌های اصلی نمایش داده می‌شود. کد نمایش داده شده موفقیت یا شکست اجرای دستور را مشخص می‌نماید(۰= موفقیت و ۱= شکست). برای مثال:

```
>> system("dir")
0
>> system("Non-existing command")
1
```

یکی دیگر از دستورات از پیش تعریف شده cd (تغییر دایرکتوری جاری) می‌باشد. نتیجه این دستور به صورت رشته برگردانده می‌شود. اگر این دستور را با پرانتز بدون هیچ پارامتری استفاده نمایید دایرکتوری فعلی را نمایش خواهد داد.

```
>> cd()
"C:\OpenModelica1.5.0\testmodels"
>> cd("..")
"C:\OpenModelica1.5.0"
>> cd("C:\\OpenModelica1.5.0\\testmodels")
"C:\OpenModelica1.5.0\testmodels"
```

۳-۷-۱۰) کتابخانه Modelica و مدل DCMotor

حال ما یک مدل را بارگذاری می‌کنیم. مدلی که در برگیرنده کتابخانه استاندارد Modelica می‌باشد. برای این کار کافی است منوی File>>Load Modelica Library را انتخاب کنید.

```
>> loadModel(Modelica)
true
```

همچنین فایل محتوی مدل موتور dc را به صورت زیر بارگذاری می‌کنیم.

```
>> loadFile("C:/OpenModelica1.5.0/testmodels/dcmotor.mo")
true
```

این مدل به صورت زیر شبیه‌سازی می‌شود. این شبیه‌سازی را برای مدت ده ثانیه انجام می‌دهیم (توجه داشته باشید که مدل بعد از بارگذاری با نام آن شناخته می‌شود):

```
>> simulate(dcmotor,startTime=0.0,stopTime=10.0)
record
    resultFile = "dcmotor_res.plt"
end record
```

حال اگر بخواهیم کد مدل را لیست کنیم:

```
>> list(dcmotor)
"model dcmotor
        Modelica.Electrical.Analog.Basic.Resistor r1(R=10);
        Modelica.Electrical.Analog.Basic.Inductor i1;
        Modelica.Electrical.Analog.Basic.EMF emf1;
        Modelica.Mechanics.Rotational.Inertia load;
        Modelica.Electrical.Analog.Basic.Ground g;
        Modelica.Electrical.Analog.Sources.ConstantVoltage v;
equation
        connect(v.p,r1.p);
        connect(v.n,g.p);
        connect(r1.n,i1.p);
        connect(i1.n,emf1.p);
        connect(emf1.n,g.p);
        connect(emf1.flange_b,load.flange_a);
end dcmotor;
"
```

همانطور که در لیست بالا مشاهده می‌کنید فقط توابع اصلی مدل نمایش داده شده است و اجزاء مدل فقط به صورت یکسری توابع لیست شده‌اند. چنانچه بخواهیم تمام مدل شامل معادلات مربوط به اجزا را در کنار هم داشته باشیم با اجرای دستور زیر به این خواسته می‌رسیم.

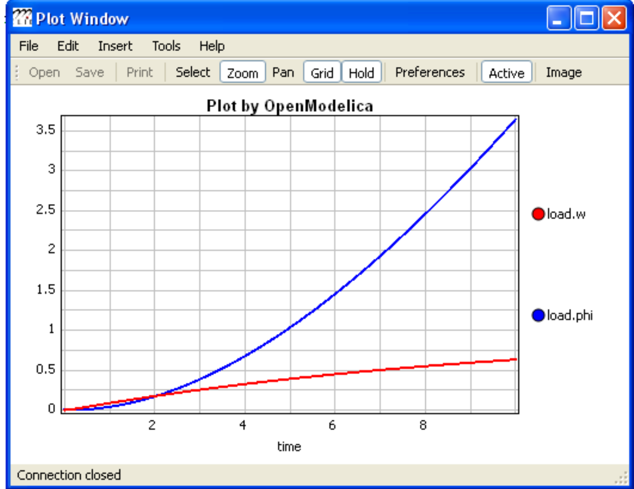
```
>> instantiateModel(dcmotor)
"fclass dcmotor
```

```
    Real r1.v "Voltage drop between the two pins (= p.v - n.v)";
    Real r1.i "Current flowing from pin p to pin n";
    Real r1.p.v "Potential at the pin";
    Real r1.p.i "Current flowing into the pin";
    Real r1.n.v "Potential at the pin";
    Real r1.n.i "Current flowing into the pin";
    parameter Real r1.R = 10 "Resistance";
    Real i1.v "Voltage drop between the two pins (= p.v - n.v)";
    Real i1.i "Current flowing from pin p to pin n";
    Real i1.p.v "Potential at the pin";
    Real i1.p.i "Current flowing into the pin";
    Real i1.n.v "Potential at the pin";
    Real i1.n.i "Current flowing into the pin";
    parameter Real i1.L = 1 "Inductance";
    parameter Real emf1.k = 1 "Transformation coefficient";
    Real emf1.v "Voltage drop between the two pins";
    Real emf1.i "Current flowing from positive to negative pin";
    Real emf1.w "Angular velocity of flange_b";
    Real emf1.p.v "Potential at the pin";
    Real emf1.p.i "Current flowing into the pin";
    Real emf1.n.v "Potential at the pin";
    Real emf1.n.i "Current flowing into the pin";
    Real emf1.flange_b.phi "Absolute rotation angle of flange";
    Real emf1.flange_b.tau "Cut torque in the flange";
    Real load.phi "Absolute rotation angle of component (= flange_a.phi =
    flange_b.phi)";
    Real load.flange_a.phi "Absolute rotation angle of flange";
    Real load.flange_a.tau "Cut torque in the flange";
    Real load.flange_b.phi "Absolute rotation angle of flange";
    Real load.flange_b.tau "Cut torque in the flange";
    parameter Real load.J = 1 "Moment of inertia";
    Real load.w "Absolute angular velocity of component";
    Real load.a "Absolute angular acceleration of component";
    Real g.p.v "Potential at the pin";
    Real g.p.i "Current flowing into the pin";
    Real v.v "Voltage drop between the two pins (= p.v - n.v)";
    Real v.i "Current flowing from pin p to pin n";
    Real v.p.v "Potential at the pin";
    Real v.p.i "Current flowing into the pin";
    Real v.n.v "Potential at the pin";
    Real v.n.i "Current flowing into the pin";
    parameter Real v.V = 1 "Value of constant voltage";
equation
    r1.R * r1.i = r1.v;
    r1.v = r1.p.v - r1.n.v;
```

```
        0.0 = r1.p.i + r1.n.i;
        r1.i = r1.p.i;
        i1.L * der(i1.i) = i1.v;
        i1.v = i1.p.v - i1.n.v;
        0.0 = i1.p.i + i1.n.i;
        i1.i = i1.p.i;
        emf1.v = emf1.p.v - emf1.n.v;
        0.0 = emf1.p.i + emf1.n.i;
        emf1.i = emf1.p.i;
        emf1.w = der(emf1.flange_b.phi);
        emf1.k * emf1.w = emf1.v;
        emf1.flange_b.tau = -(emf1.k * emf1.i);
        load.w = der(load.phi);
        load.a = der(load.w);
        load.J * load.a = load.flange_a.tau + load.flange_b.tau;
        load.flange_a.phi = load.phi;
        load.flange_b.phi = load.phi;
        g.p.v = 0.0;
        v.v = v.V;
        v.v = v.p.v - v.n.v;
        0.0 = v.p.i + v.n.i;
        v.i = v.p.i;
        emf1.flange_b.tau + load.flange_a.tau = 0.0;
        emf1.flange_b.phi = load.flange_a.phi;
        emf1.n.i + v.n.i + g.p.i = 0.0;
        emf1.n.v = v.n.v;
        v.n.v = g.p.v;
        i1.n.i + emf1.p.i = 0.0;
        i1.n.v = emf1.p.v;
        r1.n.i + i1.p.i = 0.0;
        r1.n.v = i1.p.v;
        v.p.i + r1.p.i = 0.0;
        v.p.v = r1.p.v;
        load.flange_b.tau = 0.0;
end dcmotor;
"
```

پس از انجام شبیه‌سازی، برخی از نتایج شبیه‌سازی را رسم می‌کنیم که در شکل ۷-۱۰ نشان داده شده است:

```
>> plot({load.w,load.phi})
True
```



شکل ۷-۱۰.نتیجه شبیه سازی.

۱۰-۷-۴) تابع val

تابع val(variableName,time) مقدار یک متغیر را در یک زمان مشخص باز می‌گرداند که این مقدار با استفاده از درونیابی از نتایج شبیه‌سازی محاسبه و استخراج خواهد شد. برای آشنایی بیشتر شبیه‌سازی توپ در بخش بعدی را ببینید.

۱۰-۷-۵) مدل‌های BouncingBall و Switch

اکنون مثال BouncingBall را که شامل عبارتهای when و if می‌باشد را بارگذاری و شبیه-سازی می‌کنیم.

```
>> loadFile("C:/OpenModelica1.5.0/testmodels/BouncingBall.mo")
true
>> list(BouncingBall)
"model BouncingBall
        parameter Real e=0.7 "coefficient of restitution";
        parameter Real g=9.81 "gravity acceleration";
        Real h(start=1) "height of ball";
        Real v "velocity of ball";
        Boolean flying(start=true) "true, if ball is flying";
        Boolean impact;
        Real v_new;
equation
        impact=h <= 0.0;
        der(v)=if flying then -g else 0;
        der(h)=v;
```

```
        when {h <= 0.0 and v <= 0.0,impact} then
                v_new=if edge(impact) then -e*pre(v) else 0;
                flying=v_new > 0;
                reinit(v, v_new);
        end when;
end BouncingBall;
"
```

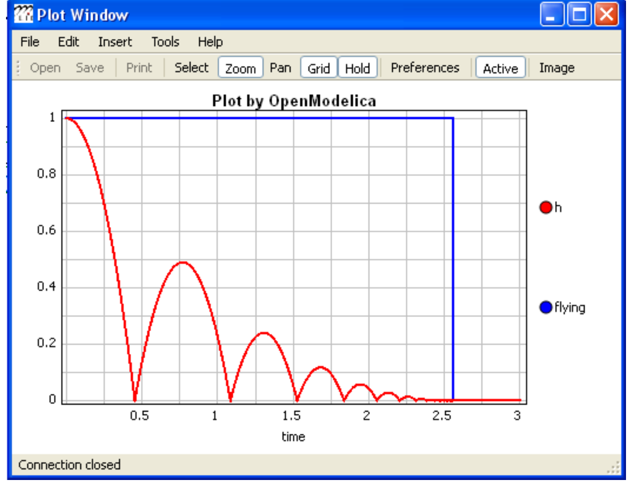
به جای اجرای دستورات simulate و plot به صورت تکی، می‌توانید با استفاده از دستور runScript فایل sim_BouncingBall.mos که شامل دستورات زیر می‌باشد و قبلاً تهیه شده است را اجرا کنید. توجه داشته باشید قبل از این که بتوانید از این فایل استفاده نمایید باید پوشه کاری را به پوشه testmodels تغییر دهید:

```
    loadFile("BouncingBall.mo");
    simulate(BouncingBall, stopTime=3.0);
    plot({h,flying});

>> runScript("sim_BouncingBall.mos")
"true
record
    resultFile = "BouncingBall_res.plt"
end record
true
true"
```

همچنین می‌توانید این دستورات را مانند بخش قبلی به صورت دستی وارد کنید. پس دستورات زیر را به ترتیب اجرا نمایید که نتیجه آن در شکل ۸-۱۰ نشان داده شده است.

```
loadFile("BouncingBall.mo");
simulate(BouncingBall, stopTime=3.0);
plot({h,flying});
```



شکل ۸-۱۰.نتیجه شبیه سازی.

حال یک مدل سوئیچ را برای امتحان معادلات if وارد می‌کنیم. (می‌توان این کار را با کپی کردن متن از یک فایل دیگر و انداختن کد در محیط OMShell و فشار دادن کلید enter انجام داد)

```
>> model Switch
        Real v;
        Real i;
        Real i1;
        Real itot;
        Boolean open;
equation
        itot = i + i1;
        if open then
                v = 0;
        else
                i = 0;
        end if;
        1 - i1 = 0;
        1 - v - i = 0;
        open = time >= 0.5;
end Switch;
Ok
```

با استفاده از دستور زیر مدل سوئیچ را برای مدت زمان یک ثانیه شبیه‌سازی می‌نماییم.

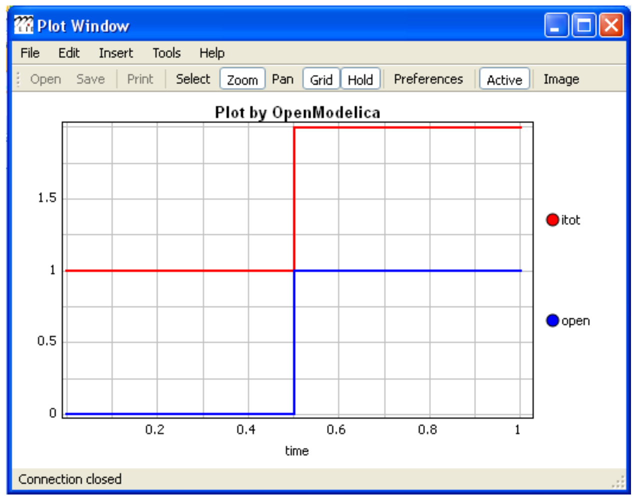
```
>> simulate(Switch, startTime=0, stopTime=1);
```

با استفاده از تابع val(variableName,time) مقدار itot را در زمان time=0 تعیین می‌کنیم:

```
>> val(itot,0)
1.0
```

متغیرهای itot و open را رسم می‌کنیم که در شکل ۹-۱۰ نشان داده شده است.

```
>> plot({itot,open})
True
```



شکل ۹-۱۰. رسم متغیرهای open و itot.

توجه کنید که تغییر متغیر open از نادرست(۰) به درست(۱) باعث افزایش itot از ۱ به ۲ می‌شود.

۶-۷-۱۰) پاک کردن تمامی مدل‌ها

در ابتدا تمامی کتابخانه‌ها و مدل‌های بارگذاری شده را پاک می‌کنیم:

```
>> clear()
true
```

لیست مدل‌های بارگذاری شده نشان می‌دهد که هیچ مدلی باقی نمانده است:

```
>> list()
```

۷-۷-۱۰) مدل VanDerPol و رسم پارامتری

اکنون مدل VanDerPol را از طریق منوی File->Load Model بارگذاری می‌کنیم.

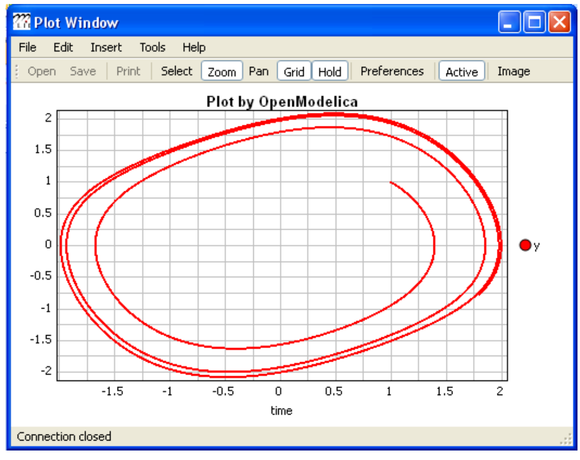
```
>> loadFile("C:/OpenModelica1.5.0/testmodels/VanDerPol.mo"))
true
```

سپس این مدل را شبیه‌سازی می‌کنیم:

```
>> simulate(VanDerPol,stopTime=20)
record
    resultFile = "VanDerPol_res.plt"
end record
```

برای رسم نتایج به صورت پارامتری از دستور زیر استفاده می‌کنیم که در شکل ۱۰-۱۰ نشان داده شده است.

```
plotParametric(x,y);
```



شکل ۱۰-۱۰.رسم نتایج به صورت پارامتری.

کد یک دست مدل VanDerPol را مشاهده می‌کنیم:

```
>> instantiateModel(VanDerPol)
"fclass VanDerPol
        Real x(start=1.0);
        Real y(start=1.0);
        parameter Real lambda = 0.3;
equation
        der(x) = y;
        der(y) = -x + lambda * (1.0 - x * x) * y;
end VanDerPol;
```

۸-۷-۱۰) برنامه نویسی با دستور if و حلقه‌های for-loop و while-loop

یک حلقه ساده برای جمع کردن مقادیر صحیح:

```
>> k := 0;
for i in 1:1000 loop
        k := k + i;
end for;
>> k
500500
```

یک حلقه تودرتو برای جمع کردن مقادیر صحیح و حقیقی:

```
>> g := 0.0;
h := 5;
for i in {23.0,77.12,88.23} loop
        for j in i:0.5:(i+1) loop
                g := g + j;
                g := g + h / 2;
        end for;
        h := h + g;
end for;
```

با قرار دادن یک یا چند semicolon بین یک یا چند متغیر یا عبارت، می‌توان مقدار بیش از یک متغیر را مشاهده نمود:

```
>> h;g
1997.45
1479.09
```

یک حلقه for-loop برای متصل کردن المانهای رشته‌ای به یکدیگر:

```
>> i:="";
lst := {"Here ", "are ","some ","strings."};
s := "";
for i in lst loop
        s := s + i;
end for;
>> s
"Here are some strings."
```

حلقه while-loop معمولی برای اتصال ۱۰ رشته "abc ":

```
>> s:="";
i:=1;
while i<=10 loop
        s:="abc "+s;
        i:=i+1;
end while;
>> s
```

"abc abc abc abc abc abc abc abc abc abc "

یک دستور ساده if را ببینید. با قرار دادن متغیر بعد از semicolon، مقدار آن بعد از ارزیابی برگردانده می‌شود:

```
>> if 5>2 then a := 77; end if; a
77
```

یک دستور if-then-else به همراه elseif:

```
>> if false then
        a := 5;
elseif a > 50 then
        b:= "test"; a:= 100;
else
        a:=34;
end if;
```

نگاهی به متغیرهای a و b:

```
>> a;b
100
"test"
```

۱۰-۷-۹) متغیرها، توابع و انواع متغیرها

اکنون به یک متغیر، یک بردار اختصاص می‌دهیم.

```
>> a:=1:5
{1,2,3,4,5}
```

سپس با نوشتن یک تابع ادامه می‌دهیم.

```
>> function MySqr input Real x; output Real y; algorithm y:=x*x; end MySqr;
Ok
```

حالا تابع را فراخوانی می‌کنیم.

```
>> b:=MySqr(2)
4.0
```

مقدار متغیر a را نمایش می‌دهیم.

```
>> a
{1,2,3,4,5}
```

مقدار نوع متغیر a را نمایش می‌دهیم.

```
>> typeOf(a)
"Integer[]"
```

مقدار نوع متغیر b را نمایش می‌دهیم.

```
>> typeOf(b)
"Real"
```

نوع MySqr چیست؟ در حال حاضر (در ویرایش OpenModelica 1.5.0) امکان تشخیص نوع تابع وجود ندارد.

پیاده‌سازی نشده است.

```
>> typeOf(MySqr)
Error evaluating expr.
```

لیست متغیرهای موجود:

```
>> listVariables()
{currentSimulationResult, a, b}
```

پاک کردن دوباره:

```
>> clear()
true
```

۱۰-۷-۱۰) دریافت اطلاعات در خصوص علت خطاها

برای نمایش داده های بیشتر از علت خطا در صورت شکست شبیه سازی از تابع getErrorString استفاده نمایید:

```
getErrorString()
```

۱۰-۷-۱۱) سایر قالب های خروجی شبیه سازی

برای ذخیره نتایج شبیه‌سازی قالبهای خروجی مختلفی را می‌توانید مشخص نمایید. پیش فرض "plt" است، در ویرایش فعلی نرم افزار فقط این قالب توانایی پشتیبانی توابع val() و plot() را دارد. قالب csv[1] در شبیه‌سازیهای سنگین تقریباً دو برابر سریعتر است، با استفاده از این قالب نیازی نیست در هنگام شبیه‌سازی همه متغیرها dدر RAM ذخیره شود. انتخاب empty و ذخیره نکردن هیچ متغیری سریعترین روش است:

```
simulate(... , outputFormat="csv")
simulate(... , outputFormat="plt")
simulate(... , outputFormat="empty")
```

۱۰-۷-۱۲) استفاده از توابع خارجی

در ادامه با استفاده از یک مثال کوچک (ExternalLibraries.mo) روش استفاده از توابع خارجی را نشان خواهیم داد. فرض کنید در بخش از شبیه سازی نیاز دارید بخشی از مدلها را در یک تابع خارجی بنویسید یا این که بخواهید از مدلی که قبلاً نوشته شده است استفاده نمایید. در مثال بعدی از زبان برنامه‌نویسی C برای نوشتن توابع خارجی استفاده می‌گردد، اما محیط openModelica امکان استفاده از Java، Fortran و Pyton را نیز در مثالهایش به نمایش گذاشته است:

```
model ExternalLibraries
```

[1] comma separated values

```
        Real x(start=1.0),y(start=2.0);
equation
        der(x)=-ExternalFunc1(x);
        der(y)=-ExternalFunc2(y);
end ExternalLibraries;
function ExternalFunc1
        input Real x;
        output Real y;
external
        y=ExternalFunc1_ext(x) annotation(Library="libExternalFunc1_ext.o",
        Include="#include \"ExternalFunc1_ext.h\"");
end ExternalFunc1;
function ExternalFunc2
        input Real x;
        output Real y;
external "C" annotation(Library="libExternalFunc2.a",
        Include="#include \"ExternalFunc2.h\"");
end ExternalFunc2;
```

برای این کار، فایل‌های C(.c) و سرآیند (.h) زیر لازم می‌باشند:

```
/* file: ExternalFunc1.c */
double ExternalFunc1_ext(double x)
{
        double res;
        res = x+2.0*x*x;
        return res;
}

/* Header file ExternalFunc1_ext.h for ExternalFunc1 function */
double ExternalFunc1_ext(double);

/* file: ExternalFunc2.c */
double ExternalFunc2(double x)
{
        double res;
        res = (x-1.0)*(x+2.0);
        return res;
}

/* Header file ExternalFunc2.h for ExternalFunc2 */
double ExternalFunc2(double);
```

به شرط آنکه gcc در مسیر مورد نظر نصب شده باشد، فایل ExternalLibraries.mos تمام وظایف لازم را انجام خواهد داد.

نرم‌افزار OMShell را اجرا نمایید. چون فایلهای مورد نیاز این مثال در پوشه testModels قرار دارد لازم است ابتدا مسیر پیش فرض را به این پوشه تغییر بدهید، برای این کار از دستور زیر استفاده نمایید (با فرض این که OpenModelica در مسیر C:\openModelica1.5.0\ نصب شده باشد):

```
>> cd("C:/openModelica1.5.0/testModels")
```

دستورات زیر را یکی پس از دیگری وارد کرده و اجرا نمایید یا کد دسته‌ای که شامل تمام دستورات زیر می‌باشد را اجرا کنید:

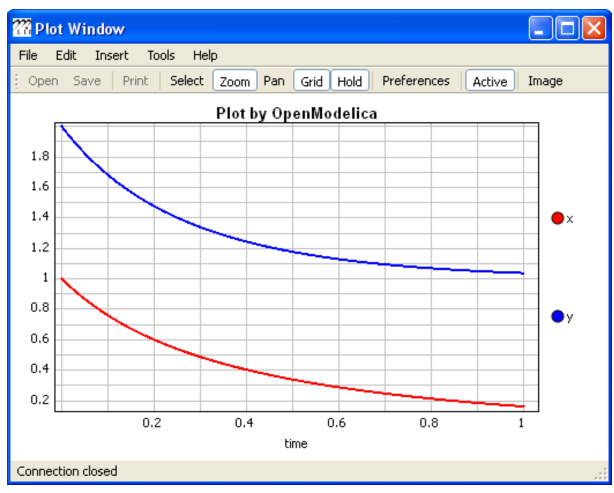
```
loadFile("ExternalLibraries.mo");
system("gcc -c -o libExternalFunc1_ext.o ExternalFunc1.c");
system("gcc -c -o libExternalFunc2.a ExternalFunc2.c");
simulate(ExternalLibraries);
```

کد دسته‌ای را با دستور زیر اجرا کنید:

```
>> runScript("ExternalLibraries.mos");
```

و نتایج را رسم می‌کنیم که در شکل ۱۱-۱۰ نشان داده شده است.

```
>> plot({x,y});
```



شکل ۱۱-۱۰.رسم نتایج با استفاده از توابع خارجی.

۱۰-۷-۱۳) فراخوانی Model Query و Manipulation API

در اسناد مربوط به سیستم OpenModelica یک API[1] یا واسطه شرح داده شده است. این رابط اطلاعاتی مدل و امکان دستکاری مدل را برمی‌گرداند. فراخوانی این تابع از طریق نرم‌افزارهای

[1] Application Programming Interface

سرویس‌گیرنده مانند OpenModelica MDT Eclipse، OMNotebook، MathModelica Lite graphic model editor می‌باشند. برای افزایش کارایی، این تابع بدون نوع می‌باشد. یعنی در هنگام فراخوانی هیچ گونه کنترلی بر روی انواع ورودی انجام نمی‌شود و بررسی خطا حداقل می‌باشد. نتیجه فراخوانی این تابع، رشته متنی با ساختار گرامری Modelica است که سرویس گیرنده باید آن را ترجمه کند. یک مترجم به زبان C++ در کد مرجع OMNotebook وجود دارد و یک مترجم دیگر به زبان Java در MDT Eclipse plugin موجود می‌باشد. در ادامه تعدادی از توابع این کتابخانه برای مدل BouncingBall فراخوانی خواهد شده است. اطلاعات کامل این API در مستندات سیستم در دسترس می‌باشد. ابتدا برای نشان دادن ساختار مدل، بار دیگر آن را بار گذاری و لیست می‌کنیم:

```
>>loadFile("C:/OpenModelica1.5.0/testmodels/BouncingBall.mo")
true
>>list(BouncingBall)
"model BouncingBall
	parameter Real e=0.7 "coefficient of restitution";
	parameter Real g=9.81 "gravity acceleration";
	Real h(start=1) "height of ball";
	Real v "velocity of ball";
	Boolean flying(start=true) "true, if ball is flying";
	Boolean impact;
	Real v_new;
equation
	impact=h <= 0.0;
	der(v)=if flying then -g else 0;
	der(h)=v;
	when {h <= 0.0 and v <= 0.0,impact} then
		v_new=if edge(impact) then -e*pre(v) else 0;
		flying=v_new > 0;
		reinit(v, v_new);
	end when;
end BouncingBall;"
```

روش‌های مختلف فراخوانی به همراه مقادیر بازگردانده شده در زیر آمده است:

```
>>getClassRestriction(BouncingBall)
"model"
>>getClassInformation(BouncingBall)
{"model","","",{false,false,false},{"writable",1,1,18,17}}
>>isFunction(BouncingBall)
false
>>existClass(BouncingBall)
true
>>getComponents(BouncingBall)
```

```
{{Real,e,"coefficient of restitution", "public", false, false, false,
"parameter", "none", "unspecified"},
{Real,g,"gravity acceleration",
"public", false, false, false, "parameter", "none", "unspecified"},
{Real,h,"height of ball", "public", false, false, false,
"unspecified", "none", "unspecified"},
{Real,v,"velocity of ball",
"public", false, false, false, "unspecified", "none", "unspecified"},
{Boolean,flying,"true, if ball is flying", "public", false, false,
false, "unspecified", "none", "unspecified"},
{Boolean,impact,"",
"public", false, false, false, "unspecified", "none", "unspecified"},
{Real,v_new,"", "public", false, false, false, "unspecified", "none",
"unspecified"}}
>>getConnectionCount(BouncingBall)
0
>>getInheritanceCount(BouncingBall)
0
>>getComponentModifierValue(BouncingBall,e)
0.7
>>getComponentModifierNames(BouncingBall,e)
{}
>>getClassRestriction(BouncingBall)
"model"
>>getVersion() // Version of the currently running OMC
"1.5.0"
```

۱۰-۷-۱۴) خروج از OpenModelica

با اجرای دستور زیر از محیط OpenModelica خارج می‌شویم.

```
>> quit()
```

۱۰-۸) تولید XML مدل

دستور dumpXMLDAE، فایل XML مدل را تولید می‌کند. این دستور گزینه‌های مختلفی دارد.

```
dumpXMLDAE(modelname[,asInSimulationCode=<Boolean>]
[,filePrefix=<String>]
[,storeInTemp=<Boolean>] [,addMathMLCode =<Boolean>])
```

این دستور نمایش ریاضی مدل را با قالب XML تولید خواهد نمود. قالب خروجی را می‌توان با انتخاب گزینه‌های مختلف کنترل نمود. برای مثال دستورات زیر را برای ایجاد قالب XML مدل BouncingBall اجرا نمایید:

```
loadFile("BouncingBall.mo");
dumpXMLDAE(BouncingBall);
```

فایل XML تولید شده را می‌توانید در پوشه tmp بیابید.

۱۰-۹) خروجی مدل در قالب Matlab

این دستور نمایش مدل را به قالب Matlab تولید خواهد نمود. برای مثال دستورات زیر را برای تولید ترجمه مدل BouncingBall اجرا نمایید.

```
loadFile("BouncingBall.mo");
exportDAEtoMatlab(BouncingBall);
```

فایل Matlab تولید شده را می‌توانید در پوشه tmp بیابید.

۱۰-۱۰) دستوراتی برای Interactive Session Handler

در ادامه لیست کاملی از دستورات موجود در interactive session hander در جدول ۱-۱۰ ارائه شده است:

جدول ۱-۱۰. دستورات موجود در interactive session hander.

simulate(modelname)	مدلی با نام modelname را ترجمه و شبیه‌سازی می‌کند.
simulate(modelname) [,startTime=<Real>] [,stopTime=<Real>] [,numberOfIntervals=<Integer>])	یک مدل را بر اساس پارمترهای اختیاری زمان شروع، زمان پایان، تعداد بازه‌های شبیه‌سازی یا گام‌هایی که لازم است به ازای آنها نتایج شبیه‌سازی محاسبه شود، ترجمه و شبیه-سازی می‌کند. تعداد پیش فرض بازه‌ها ۵۰۰ می‌باشد.
plot(vars)	متغیرهای برداری یا عددی داده شده را رسم می‌کند. برای مثال plot({x1,x2}) یا plot(x1)
plotParametric(var1, var2)	var2 را نسبت به var1 رسم می‌کند. برای مثال plotParametric(x,y)
cd()	دایرکتوری جاری را برمی‌گرداند.
cd(dir)	دایرکتوری را به رشته داده شده (dir) تغییر می‌دهد.
clear()	تمامی تعاریف بار گذاری شده را پاک می‌کند.
clearVariables()	تمامی متغیرهای تعریف شده را پاک می‌کند.

instantiateModel(modelname)	کد یک دست مربوط به مدل را به صورت یک رشته برمی‌گرداند.
list()	یک رشته شامل تعاریف تمامی کلاس‌های بار گذاری شده را برمی‌گرداند.
list(modelname)	یک رشته شامل تعریف کلاس modelname را برمی‌گرداند.
listVariables()	یک بردار شامل نام متغیرهای تعریف شده جاری برمی‌گرداند.
loadModel(classname)	مدل یا بسته‌ای با نام classname که در مسیر مشخص شده توسط متغیر OPENMODELICALIBRARY قرار دارد را بار گذاری می‌کند.
loadFile(str)	فایل Modelica (.mo) با نام داده شده در str را بار گذاری می‌کند.
readFile(str)	فایل داده شده با نام str را بار گذاری می‌کند و رشته محتوی فایل را برمی‌گرداند.
runScript(str)	فایل دستورات با نام داده شده str را اجرا می‌کند.
system(str)	دستور سیستم عامل ارائه شده در str را اجرا می‌کند و یک عدد صحیح بیانگر موفقیت اجرا برمی‌گرداند.
timing(expr)	عبارت expr را محاسبه می‌کند و مقدار زمان مورد نیاز برای این محاسبه را برمی‌گرداند.
typeOf(variable)	نوع متغیر variable را به صورت یک رشته برمی‌گرداند.
saveModel(str,modelname)	مدل با نام modelname را در فایل مشخص شده توسط رشته str ذخیره می‌کند.
help()	متن راهنما را نمایش می‌دهد.
quit()	از محیط OpenModelica خارج می‌شود.

فصل ۱۱ گرافیک

در این فصل به بررسی رسم نمودارهای دو بعدی در نرم افزارهای OMNotebook و OMShell خواهیم پرداخت. در حال حاضر پویانمایی سه بعدی فقط در OMNotebook قابل استفاده است.

در ویرایش جدید OpenModelica رسم نمودارها برمبنای کتابخانه گرافیکی Qt قابل استفاده است. استفاده از این کتابخانه امکانات بیشتری نسبت به روش کلاسیک، رسم نمودار را برای کاربر فراهم نموده است. در فصلهای گذشته بدون این که بدانید از این کتابخانه نیز استفاده نموده‌اید. در این فصل خیلی عمیق‌تر به روش رسم نمودارها و استفاده از پویانمایی خواهیم پرداخت. امکانات فراهم شده توسط این کتابخانه عبارتند از:

- ارتباط متقابل با OMNotebook، این کتابخانه گرافیکی به گونه‌ای طراحی شده است که کاملاً با محیط OMNotebook سازگار است و تمامی امکانات این کتابخانه در این محیط قابل استفاده است. حتی ترسیمات ویرایشهای گذشته را نیز به راحتی می‌توان با این کتابخانه رسم نمود.
- استفاده از کتابخانه بدون استفاده از OMNotebook، اگر از امکانات این کتابخانه در خارج از محیط OMNotebook استفاده گردد، پنجره ای برای رسم نمودار با باز خواهد شد.
- استفاده از محورهای لگاریتمی، پاسخ برخی از شبیه‌سازیها در بازه بسیار گسترده‌ای قرار دارد در این حالت رسم نمودار با محورهای لگاریتمی می‌تواند نمودار را به صورت خواناتری نمایش دهد.
- بزرگنمایی.
- پشتیبانی از برنامه‌نویسی گرافیکی، برای داشتن مدلهای Modelica که امکان رسم نمودار و نمایش دیاگرام را داشته باشند، به غیر از این که می‌توان از توابع API خارجی OMC استفاده نمود، امکان برنامه‌نویسی گرافیکی از داخل مدل نیز فراهم شده است.
- توابع API قابل برنامه‌نویسی، توابع فراهم شده قابل برنامه‌نویسی در کتابخانه Modelica.Graphics.Plot قرار دارند. این توابع برای دسترسی به امکانات توابع گرافیکی از طریق سایر برنامه‌های ویندوز طراحی شده است.

توابع گرافیکی قابل برنامه نویسی فراهم شده عبارتند از:

- تابع plot(x) یک نمودار را به صورت دو بعدی رسم می‌کند.

- تابع plotParametric(x,y) یک نمودار دو بعدی پارامتری از y را به صورت تابعی از x رسم می‌کند.
- تابع plotTable([x1, .., y1, .. ; xn, .., yn]) یک نمودار دو بعدی پارامتری از y را به صورت تابعی از x رسم می‌کند.
- تابع drawRect(x1, x2, y1, y2) برای رسم مستطیل از نقطه (x1, y1) تا نقطه (x2, y2).
- تابع drawEllipse(x1, x2, y1, y2) برای رسم بیضی در محیط یک مستطیل فرضی از نقطه (x1, y1) تا نقطه (x2, y2).
- تابع drawLine(x1, x2, y1, y2) برای رسم خطی از نقطه (x1, y1) تا نقطه (x2, y2).

۱-۱۱) رسم نمودار ساده دوبعدی

برای رسم نمودار ساده از مثال HelloWorld استفاده خواهیم نمود. برای کاهش تعداد داده‌ها، شبیه‌سازی را با استفاده از گزینه numberOfIntervals=10 برای ده نقطه زمانی انجام می‌دهیم. دستور شبیه‌سازی به شکل زیر خواهد بود:

```
simulate(HelloWorld, startTime=0, stopTime=4, numberOfIntervals=10);
```

نتایج شبیه‌سازی پس از پایان شبیه‌سازی در فایل HelloWorld_res.plt قابل مشاهده است. این فایل را می‌توانید با استفاده از هر برنامه ویرایشگر متنی مشاهده نمایید. محتوای این فایل در بخش مقدار x به شکل زیر خواهد بود.

```
0                       1
4.440892098500626e-013  0.9999999999995559
0.4444444444444444      0.6411803884299349
0.8888888888888888      0.411112290507163
1.333333333333333       0.2635971381157249
1.777777777777778       0.1690133154060587
2.222222222222222       0.1083680232218813
2.666666666666667       0.06948345122279623
3.111111111111112       0.04455142624447787
3.555555555555556       0.02856550078454138
4                       0.01831563888872685
```

برای رسم نتایج به صورت نمودار، دستور plot(x) را اجرا نمایید. همانطور که در نمودار می‌بینید، نمودار به صورت قطعه قطعه است. اگر تعداد نقاط را زیاد کنید این مشکل رفع خواهد شد. پیش فرض ۵۰۰ نقطه است. با راست تلیک کردن روی نمودار می‌توانید به امکانات زیادی برای تنظیم، بزرگنمایی و ... دسترسی پیدا کنید.

۲-۱۱) همه توابع رسم و گزینه های آنها

توابع رسم نمودار در Modelica در جدول ۱-۱۱نشان داده شده است.

جدول ۱-۱۱.توابع رسم نمودار در Modelica.

دستور	شرح
plot(x)	متغیر x را از شبیه سازی قبل رسم می‌کند
plot({x,y,..., z})	مانند دستور قبل، برای رسم تعداد زیادی متغیر
plot(model, x)	متغیر x را از شبیه سازی model رسم می‌کند
plot(model, {x,y,..., z})	مانند دستور قبل، برای رسم تعداد زیادی متغیر
plotParametric(x, y)	رسم نمودار پارامتری از آخرین شبیه‌سازی
plotParametric(model, x, y)	مانند دستور بالا، نتایج را از شبیه‌سازی model رسم می-کند
plotAll()	همه متغیرهای آخرین شبیه‌سازی را رسم می‌کند
plotAll(model)	همه متغیرهای شبیه‌سازی model را رسم می‌کند

همه این دستورات دارای گزینه‌هایی برای تنظیم نمودار رسم شده هستند. گزینه‌های موجود و مقدار پیش فرض آنها در جدول ۲-۱۱ بیان شده است.

جدول ۲-۱۱. گزینه‌های موجود برای تنظیم نمودار رسم شده.

گزینه	پیش فرض	شرح
grid	true	شبکه بندی رسم شود؟
title	"Plot by OpenModelica "	عنوان نمودار
interpolation	linear	روش میانیابی نقاط نمودار را مشخص می‌کند: • گزینه linear به عنوان میانیابی خطی • گزینه constant مقدار داده قبل را تا رسیدن به داده جدید رسم می‌کند • گزینه none فقط نقاط داده‌ها رسم می‌گردد.
Legend	true	شرح نمودار نمایش داده شود؟
points	true	نقاط مشخص گردد؟
logX	false	محور افقی لگاریتمی باشد؟
Logy	false	محور عمودی لگاریتمی باشد؟
xRange	{0, 0}	محدوده محور افقی {۰و۰} برای تعیین خودکار محدوده
yRange	{0, 0}	محدوده محور عمودی {۰و۰} برای تعیین خودکار محدوده

antiAliasing	false	فیلتر گرافیکی برای صافی تصویر اعمال گردد؟
vTitle	“”	عنوان محور عمودی
hTitle	“time”	عنوان محور افقی

۳-۱۱) بزرگنمایی

می‌توانید برای بزرگنمایی بخشی از نمودار از تلیک چپ موشی روی همان بخش دلخواه استفاده نمایید. همچنین امکان استفاده از گزینه xRange و yRange در دستور رسم وجود دارد.

۴-۱۱) رسم نمودار در حین شبیه سازی

وقتی یک شبیه‌سازی به طول می‌انجامد یا زمانی که می‌خواهید نمودار بدون استفاده از دستورات plot یا plotParametric رسم شود، امکان استفاده از انتقال داده‌ها در حین شبیه‌سازی وجود دارد. این امکان با استفاده از دستور زیر مهیا می گردد:

```
enableSendData(true)
```

همین دستور با گزینه false برای از کار انداختن این گزینه استفاده می‌گردد. با توجه به این که انتقال داده زمان بر است با استفاده از این گزینه زمان شبیه‌سازی طولانی تر خواهد بود. برای انتخاب داده‌های مورد نظر جهت ذخیره می‌توانید از دستور setVariableFilter استفاده نمایید. برای مثال اگر بخواهیم نتایج متغیر x از شبیه‌سازی HelloWorld را با این روش ذخیره نماییم، شبیه‌سازی را با دستورات زیر انجام می‌دهیم.

```
class HelloWorld
        Real x(start = 1);
        parameter Real a = 1;
equation
    der(x) = - a * x;
end HelloWorld;
enableSendData(true);
setVariableFilter({x});
simulate(HelloWorld, startTime=0, stopTime=25);
```

وقتی انتقال داده‌ها انجام گرفت یک کلید D در سمت راست بخش ورودی نمایان می‌گردد. با فشار این کلید پنجره Simulation data نمایش داده خواهد شد که با استفاده از آن می‌توانید نتایج شبیه‌سازی را رسم نمایید.

فصل ۱۲ استفاده از ویرایشگر گرافیکی مدل

برای استفاده از این نرم افزار لازم است که نرم‌افزار JDK (Java Development Kit) در سیستم شما نصب شده باشد.

در این فصل مثال ساده‌ای برای استفاده از محیط گرافیکی SimForge را شرح خواهیم داد. نرم‌افزار SimForge در ویرایش 1.5.0 به عنوان یک بخش از نرم‌افزار openModelica ارائه شده است. پس از نصب نرم‌افزار OpenModelica برای دسترسی به SimForge کافی است به پوشه محل نصب برنامه رفته، پوشه SimForge را در آنجا بیابید. برای نصب فایل SimForge-0.9.0RC1.jar را اجرا نمایید. پس از نصب برنامه به پوشه محل نصب آن رفته و از فایل SimForge.jar روی صفحه اصلی یک میانبر ایجاد نمایید. این فایل را برای ورود به محیط نرم افزار اجرا نمایید.

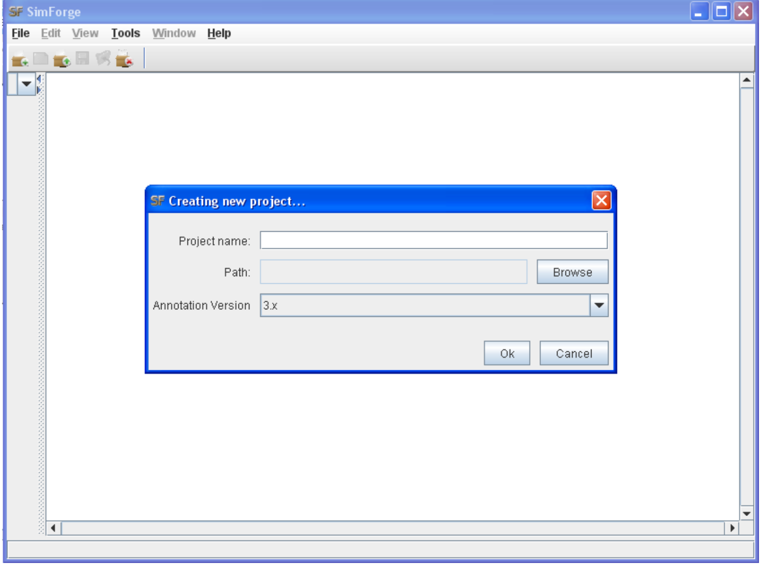
قبل از این که قادر به استفاده از این نرم‌افزار باشید لازم است آدرسهای زیر را برای نرم‌افزار مشخص نمایید (برای مشخص نمودن آدرس از منو Tools گزینه Setting را اجرا کنید):

```
OPENMODELICAHOME    => C:\OpenModelica1.5.0\
OPENMODELICALIBRARY => C:\OpenModelica1.5.0\ModelicaLibrary
```

تنظیمات را ذخیره نمایید. اگر همه چیز درست باشد هنگام ذخیره نمودن آدرسها پنجره‌های تأیید صحت عملیات نمایش داده خواهد شد.

۱-۱۲) ایجاد یک پروژه جدید

از منو File گزینه New project را انتخاب کنید. برای پروژه یک نام انتخاب نمایید. اسم پوشه‌ای که همه فایلهای پروژه را نگه خواهد داشت نیز همین نام است. همچنین لازم است مسیری برای ایجاد پوشه انتخاب کنید که مراحل کار در شکل ۱-۱۲ نشان داده شده است.



شکل ۱-۱۲. نشان دادن مراحل ایجاد یک پروژه جدید.

نرم افزار SimForge پوشه‌ای را با همین نام ایجاد خواهد نمود. در این پوشه، پوشه‌های IEC61131، پوشه Modelica برای کدهای Modelica، پوشه نتایج برای نگهداری نتایج شبیه-سازی و پوشه Temp برای نگهداری فایلهای موقت و یک فایل properties.xml برای نگهداری ویژگیهای پروژه است. درخت Modelica شامل سه گروه است:

- کتابخانه‌های خارجی مدل: شامل تمام کتابخانه‌هایی است که در پروژه استفاده شده‌اند اما در پوشه پروژه قرار ندارند. مثلاً قطعات کتابخانه استاندارد Modelica یا سایر کتابخانه‌های تجاری دیگر که می‌توانـد در پروژه کاربرد داشته باشند.
- کلاسهای Modelica: شاخه‌ای شامل همه مدلها و کتابخانه‌های تعریف شده در پروژه می‌باشد، مدلهای موجود در این شاخه هم در محیط متنی و هم در محیط گرافیکی قابل ویرایش هستند.
- فایلهای Modelica: شامل همه فایلهای با پسوند .mo است که در پروژه قرار دارند. برای بررسی فایلها قابل استفاده است و برای دسته بندی محتوای این فایلها مناسب است. ویرایش این شاخه فقط در حالت متنی ممکن است.

پس از این مرحله، نرم افزار نظر شما را در خصوص اضافه نمودن کتابخانه استاندارد Modelica به پروژه خواهد پرسید، گزینه Yes را انتخاب نمایید که پیغام آن در شکل ۲-۱۲ نشان داده شده

است. پس از این مرحله در صورت پاسخ مثبت باید ویرایش کتابخانه Modelica را انتخاب نمایید. گزینه 3.1 را انتخاب نمایید.

Choose Modelica Library
Do you want to use default Modelica Library? (STRONGLY RECCOMENDED)
Yes No

شکل ۲-۱۲.پیغام برنامه در خصوص اضافه نمودن کتابخانه استاندارد.

برای نگه داشتن همه مدلهای مربوط به هم یک بسته بسازید، برای این کار روی پوشه Modelica در پنجره سمت چپ دو بار تلیک کنید تا این پنجره باز شود. روی پوشه Modelica Classes در این پوشه راست تلیک کنید و برای ایجاد یک بسته گزینه Add class را انتخاب نمایید. از پنجره باز شده گزینه package با نام test را انتخاب نمایید. توجه نمایید که گزینه Structured entity (directory) انتخاب نشده باشد تا بسته ایجاد شده در یک فایل ذخیره گردد. در صورتی که این گزینه انتخاب شده باشد به ازاء هر مدل اضافه شده به این بسته یک فایل با پسوند mo. ساخته خواهد شد. مراحل ذخیره فایل در شکل ۳-۱۲ نمایش داده شده است.

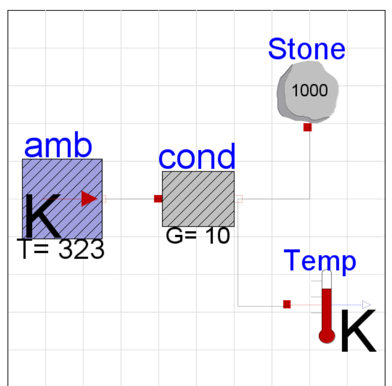
SF Add class
Name: test
Type: Package
Structured entity (directory)
Ok Cancel

شکل ۳-۱۲. مراحل ذخیره فایل.

پوشه Modelica classes را باز کنید. بسته ایجاد شده به نام test را به صورت یک پوشه در این بخش خواهید دید. برای اضافه نمودن یک مدل به این بسته روی بسته test تلیک راست کرده و مانند مثال بالا یک مدل به نام model1 را به این پوشه اضافه نمایید که در شکل ۴-۱۲ نشان داده شده است.

شکل ۴-۱۲. مراحل اضافه نمودن یک مدل به پوشه.

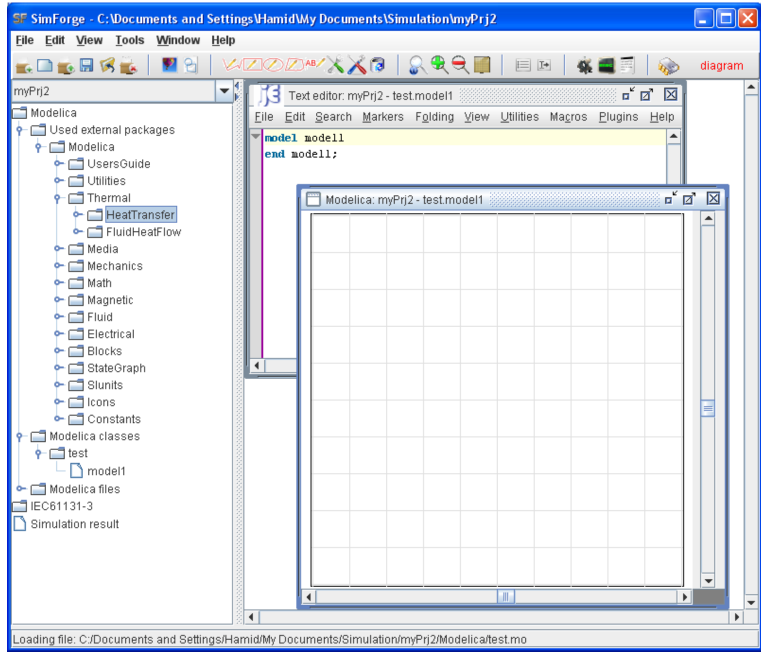
اگر روی model1 دو بار تلیک کنید، پنجره‌های ویرایش مدل شامل پنجره Modelica و textEditor باز خواهد شد. در بالای این پنجره ها نام کامل و آدرس آنها نمایش داده شده است. حال می‌توانید مدل جدید خود را ایجاد نمایید. اجازه بدهید یک مدل از گرم شدن یک جرم را با استفاده از کتابخانه Modelica مدلسازی نماییم. برای این کار کتابخانه Modelica را باز کنید، این کتابخانه در آدرس Used external package>Modelica قرار دارد. از اجزاء موجود در کتابخانه HeatTransfer در کتابخانه Thermal استفاده نموده و مدل شکل ۵-۱۲ را بسازید که پس از کشیدن هر جزء به پنجره Modelica فوراً پنجره‌ای برای دریافت نام قطعه باز خواهد شد.



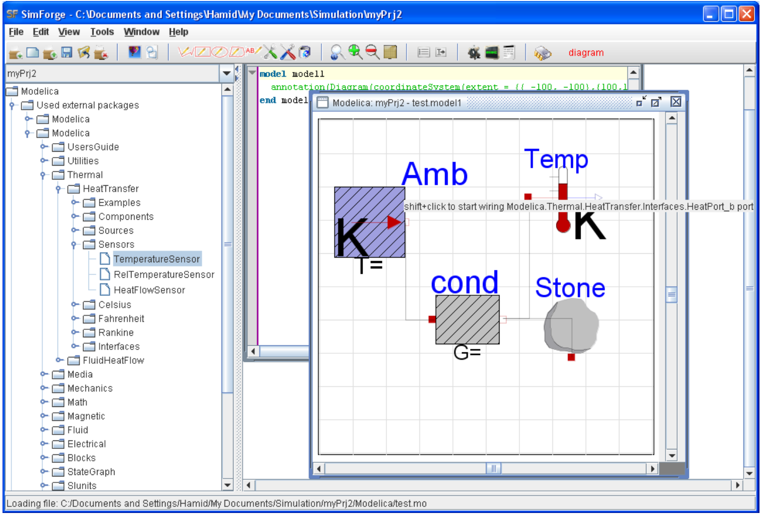
شکل ۵-۱۲.مراحل ساخت مدل گرم شدن یک جرم.

برای اتصال قطعات مختلف، موشی را روی درگاه هر قطعه قرار دهید. اگر موشی دقیقاً روی درگاه قرار داشته باشد با پس از گذشت حدود ۲ ثانیه توضیحی ظاهر می گردد که به شما اعلام می کند در صورتی که می‌خواهید این درگاه را به درگاه دیگری اتصال بدهید لازم است از ترکیب Shift+Click استفاده نمایید. حال کلید shift را نگه دارید و روی درگاه تلیک کنید. کلید shift را رها کنید، موشی را روی درگاه دوم ببرید و روی این درگاه نیز تلیک کنید تا بین دو درگاه اتصال

برقرار گردد. به همین ترتیب اتصال بین همه قطعات را برقرار کنید. در صورتی که یک اتصال اشتباه به وجود آمد آن را انتخاب و سپس پاک کنید و اتصال جدید ایجاد نمایید.

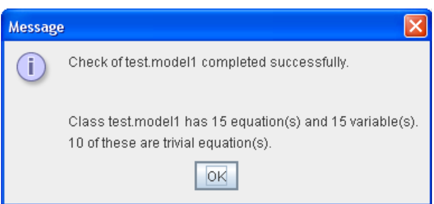


شکل ۶-۱۲. محیط **SimForge**.



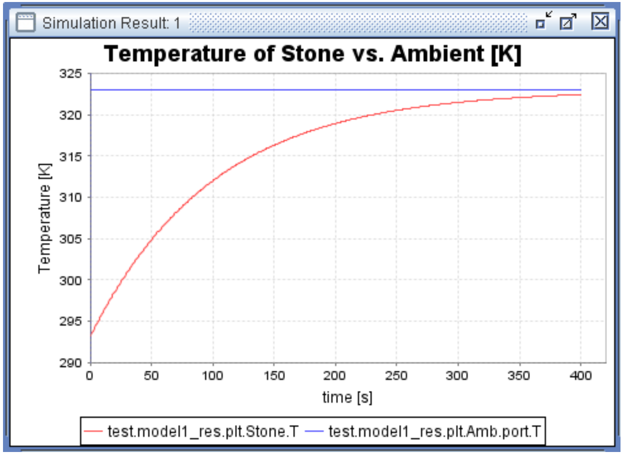
شکل ۷-۱۲. مدل پیاده سازی شده.

برای مشخص نمودن مقادیر پارامترهای قطعات، روی هر قطعه راست تلیک کرده و گزینه properties را انتخاب نمایید. در پنجره خواص ظاهر شده، برگ پارامترها را انتخاب کنید و مقادیر پارامترها را وارد نمایید. مدل را ذخیره نمایید. حال با تلیک روی آیکون مدل را از نظر تعداد معادلات و مجهولات بررسی نمایید. اگر همه چیز درست باشد، پنجره‌ای برای تأیید این مطلب نمایش داده خواهد شد که در شکل ۸-۱۲ نشان داده شده است.



شکل ۸-۱۲.نمایش پیغام تایید.

روی آیکون شبیه‌سازی تلیک کنید. در پنجره نمایش داده شده زمان شروع را ۰ و زمان پایان شبیه‌سازی را برابر ۴۰۰ وارد نمایید و برای انجام شبیه‌سازی روی کلید OK تلیک کنید. برای نمایش نتایج پوشه Simulation result را باز کنید و در آنجا از پوشه Stone گزینه T را برای نمایش دمای سنگ و از پوشه Amb، پوشه port گزینه T را برای نمایش دمای هوای محیط انتخاب کنید. همچنین با تلیک راست روی محورها و انتخاب گزینه properties می‌توانید خواص مختلف را تنظیم نموده و نمودار خواناتری ایجاد کنید که نتیجه در شکل ۹-۱۲ نشان داده شده است.



شکل ۹-۱۲. نمودار نمایش تغییرات دمای سنگ با دمای هوا.

فصل ۱۳ ویرایشگر ارتباطیOpenModelica

۱۳-۱) درباره OMEdit

ویرایشگر گرافیکی Open Modelica یک رابط گرافیکی جدید برای کاربر می باشد که برای ویرایش مدل گرافیکی در Open Modelica بکار می رود. در این ویرایشگر با استفاده از Qt 4.7یک کتابخانه گرافیکی در C++ ایجاد شده است. این فصل مقدمه ای بر OMEdit می باشد و همچنین نشان می دهد که چگونه می توان مدل DCmotor را در آن ایجاد کرد.

OMEdit دارای خصوصیات زیر است که باعث می شود کاربر به راحتی با آن کار کند.

- مدلسازی: ساخت مدلهای ساده برای مدلهای Modelica.
- مدلهای از قبل تعریف شده: برای نشان دادن کتابخانه استاندارد Modelica برای دسترسی به مدلهای تهیه شده.
- مدلهای تعریف شده توسط کاربر: کاربر می تواند مدل مورد استفاده خود را تعریف کند که متعاقبا مورد استفاده قرار می گیرد.
- رابط های اجزا: ویرایشگر هوشمند برای رسم و ویرایش درگاهها بین رابطهای مدل.
- شبیه‌سازی: یک زیر سیستم برای اجرای شبیه‌سازی و تعیین کردن پارامترهای شبیه-سازی با شروع و خاتمه زمان.
- رسم کردن: رابطی برای رسم متغیرها از مدلهای شبیه سازی شده.

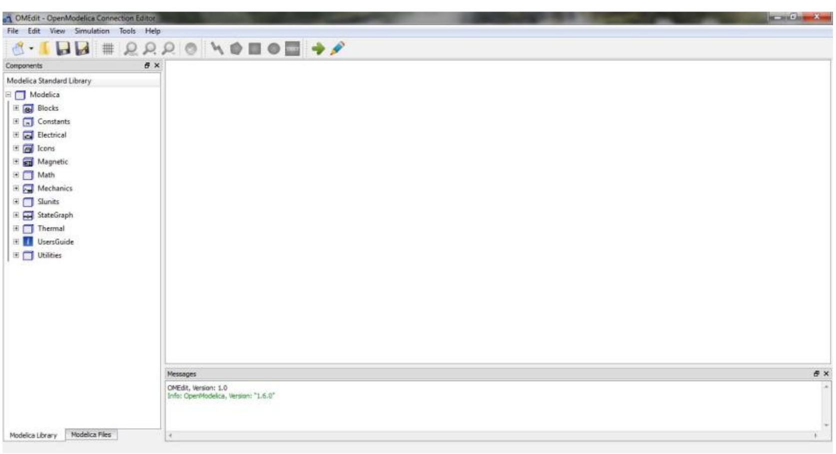
۱۳-۲) چگونه OMEdit را شروع کنیم؟

OMEdit می تواند با انتخاب OpenModelica > Open Modelica Connection Editor از منوی start ویندوز شروع شود. سپس صفحه ای مطابق با شکل ۱-۱۳ که نشان دهنده اجرای OMEdit است پدیدار می شود. در ادامه صفحه ای مطابق با شکل ۲-۱۳ بر روی صفحه نشان داده می شود.

شکل ۱-۱۳. صفحه اعلام شروع OMEdit.

۱۳-۳) مدل مقدماتی در OMEdit

در این قسمت به تشریح ساخت مدلهای Modelica در OMEdit می پردازیم. به طور مثال مدل DCmotor که در ادامه به آن می‌پردازیم.

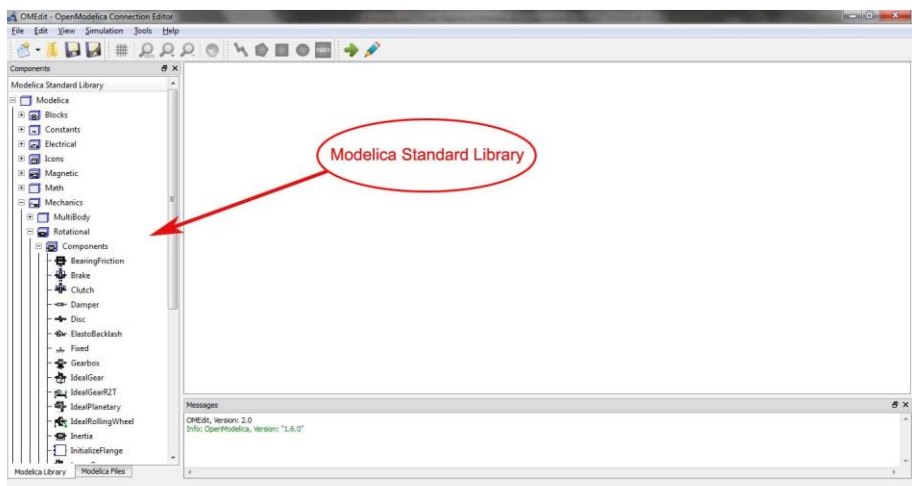


شکل ۲-۱۳. پنجره اصلی OMEdit.

۱۳-۳-۱) ساخت فایل جدید

ساخت یک فایل یا مدل جدید در OMEdit خیلی راحت است. در OMEdit فایل جدید می تواند یکی از انواع model، class، connector، record، block، function و package باشد. کاربر میتواند بسته به دلخواه خود هرکدام از انواع فایل را از منوی File > New انتخاب کند. همچنین کاربر میتواند با تلیک بر روی دکمه کشویی کناری new icon واقع در نوار ابزار بالایی انواع فایل را انتخاب نماید. مراحل در شکل ۴-۱۳ نشان داده شده است.

برای شرح این مثال مقدماتی یک مدل جدید بنام DCmotor می سازیم. با پیش فرض اینکه مدل جدید ساخته شده از منوی view قابل مشاهده است و از طریق Designer Window نیز قابل مشاهده می باشد.



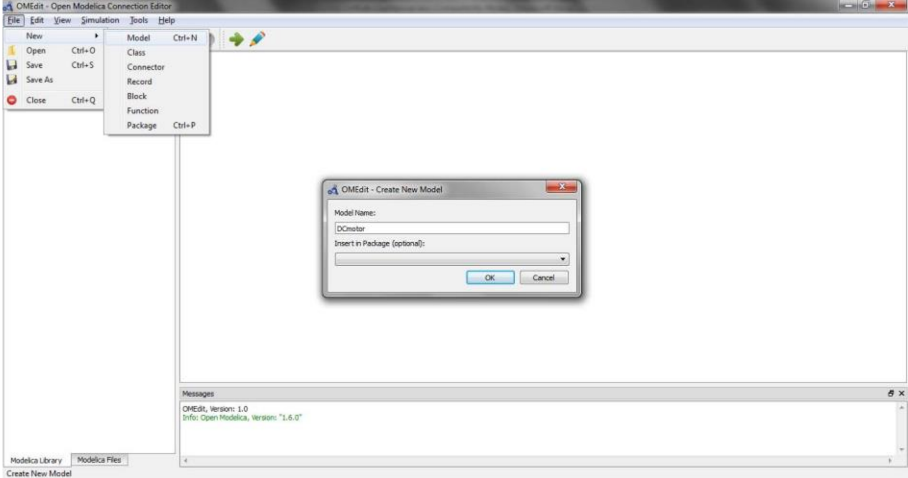
شکل ۳-۱۳. کتابخانه استاندارد Modelica.

۲-۳-۱۳) اضافه کردن اجزای مدلها

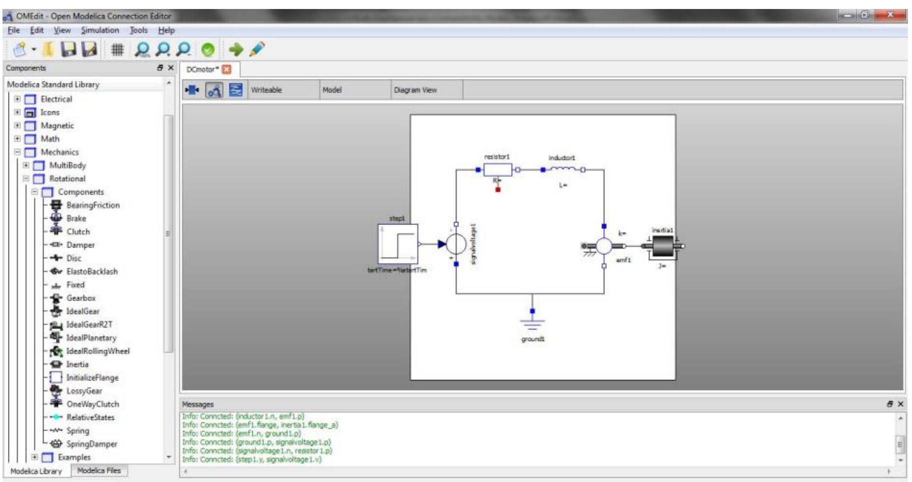
کتابخانه استاندارد Modelica به طور اتوماتیک بار گذاری شده و در سمت چپ پنجره قابل دسترس است. مدل اجزا در کتابخانه استاندارد Modelica به وسیله کشیدن و گذاشتن در پنجره کتابخانه قابل اضافه شدن به مدلها می‌باشد.

۳-۳-۱۳) برقراری ارتباط بین دو مدل

برای ارتباط مدل یک جزء با جزء دیگر، کاربر باید بر روی هرکدام از درگاه‌های مدل تلیک کرده و سپس نشانگر را به سمت درگاه دیگر هدایت کند. به این ترتیب ارتباط بین دو مدل برقرار می‌شود.



شکل ۴-۱۳. ساخت یک مدل جدید.



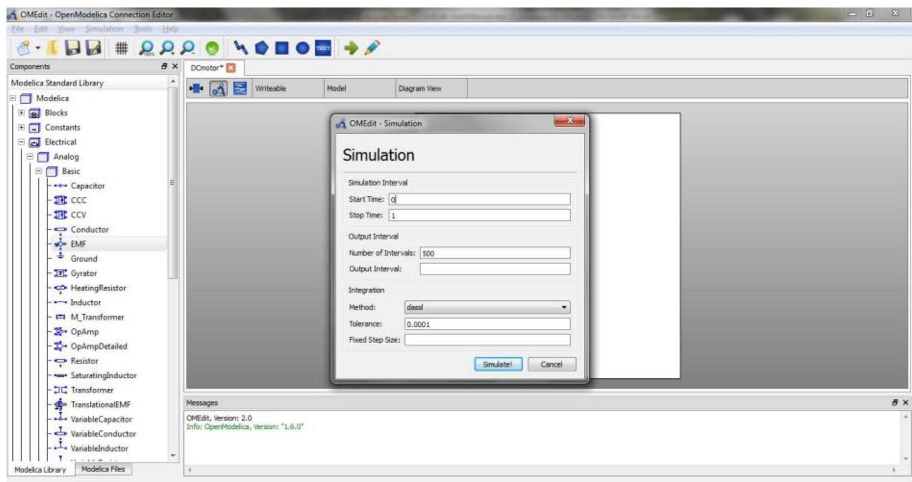
شکل ۵-۱۳. مدل **DCmotor**.

۴-۳-۱۳) شبیه سازی مدل

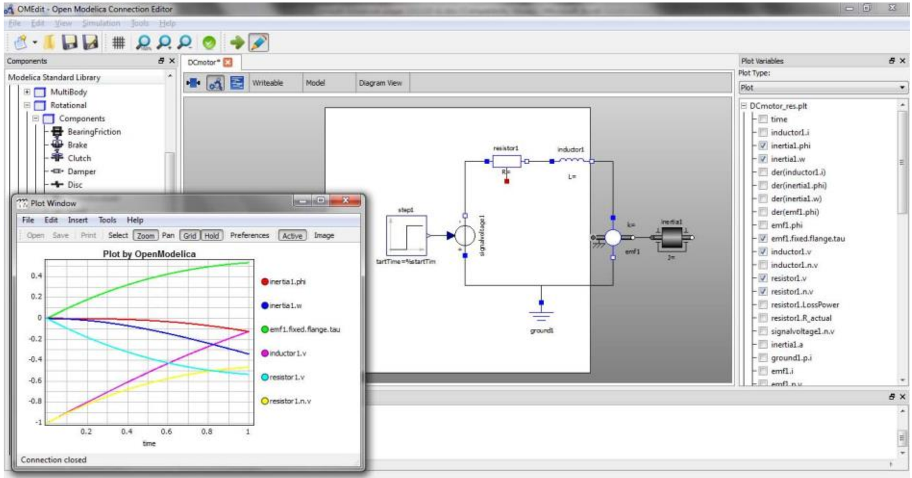
در OmEdit، شبیه سازی از طریق انتخاب گزینه Simulate در Simulation یا تلیک بر روی آیکون Simulate در نوار ابزار قابل دسترسی است. وقتی که کاربر بر روی آیکون مذکور تلیک کند فرایند شبیه سازی مدل شروع شده و در انتهای شبیه سازی پنجره رسم متغیرها در سمت راست نمایش داده می‌شود. شکل ۶-۱۳ پنجره شبیه سازی را نمایش می‌دهد.

۵-۳-۱۳) رسم متغیرها از مدلهای شبیه سازی شده

متغیرهای دخیل در رسم نمودار، در سمت راست صفحه نمایش در قالب پنجره‌ای نمایش داده می‌شوند. این پنجره به طور خودکار در هنگام شبیه سازی مدل قابل دسترس می‌باشد. همچنین کاربر می‌تواند از طریق Simulation>Plot variables یا تلیک بر روی آیکون Plot از نوار ابزار این پنجره را فعال نماید. این پنجره شامل لیست کلیه متغیرهایی است که امکان رسم نمودار آن فراهم می‌باشد که کاربر با انتخاب هرکدام از متغیرها نمودار آن را مشاهده خواهد کرد.



شکل ۶-۱۳. پنجره شبیه‌سازی.



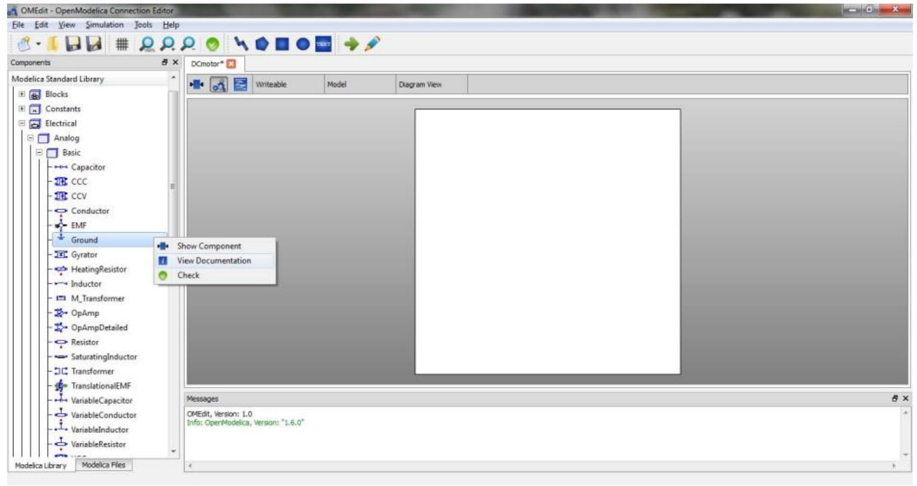
شکل ۷-۱۳. متغیرهای ترسیم شده.

۴-۱۳) پنجره ها

OMEdit از تعدادی پنجره که نماهای متفاوتی را به کاربر نشان می‌دهد تشکیل شده است.

۱-۴-۱۳) پنجره کتابخانه

مهمترین پنجره، پنجره کتابخانه می‌باشد. کتابخانه استاندارد Modelica به طور خودکار در OMEdit بارگذاری شده و در پنجره سمت چپ واقع شده است. پنجره کتابخانه این امکان را به کاربر می‌دهد که پس از ساخته شدن مدل، اجزا را از کتابخانه به مدل انتقال دهد. با راست تلیک کردن بر روی مدل در پنجره کتابخانه و انتخاب نمایش مستندات پنجره مربوطه باز می‌شود که در شکل ۸-۱۳ نمایش داده شده است.



شکل ۸-۱۳. پنجره نمایش اجزای مدل.

۲-۴-۱۳) پنجره طراحی

پنجره طراحی، پنجره اصلی در OMEdit می‌باشد. این پنجره از سه قسمت تشکیل شده است:

- نمایش آیکون: مدل نمایش آیکون را نشان می‌دهد.
- نمایش دیاگرام: دیاگرام مدل ساخته شده توسط کاربر را نشان می‌دهد.
- نمایش متن Modelica: مدل متنی Modelica را نمایش می‌دهد.

۱۳-۴-۳) پنجره رسم

پنجره رسم یک نمودار درختی است که فهرست متغیرهای استخراج شده از نتایج شبیه سازی را نشان می‌دهد. هر مورد از این نمودار درختی دارای قابلیت انتخاب توسط کاربر را دارد که به وسیله آن پنجره ترسیم نمودار نمایش داده می‌شود. کاربر می‌تواند متغیرهایی را به این نمودار اضافه یا کسر کند.

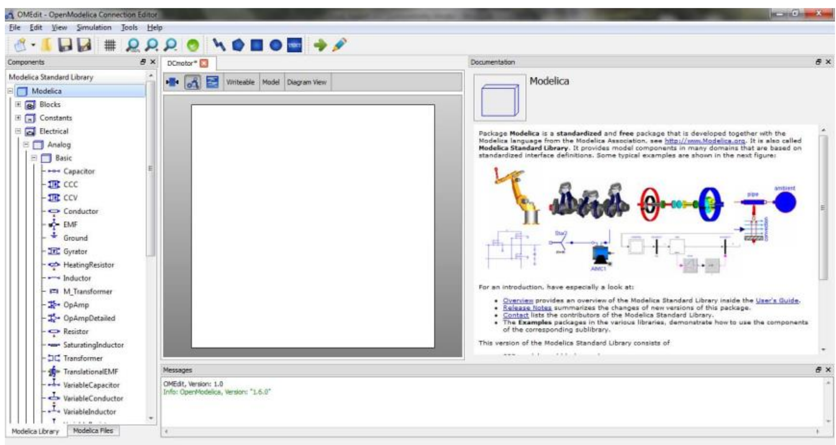
۱۳-۴-۴) پنجره پیغام

پنجره پیغام در پایین صفحه قرار دارد. این پنجره شامل چهار نوع پیغام می‌باشد:

- پیغامهای عمومی که با رنگ مشکی نمایش داده می‌شود.
- پیغامهای اخباری که با رنگ سبز نمایش داده می‌شود.
- پیغامهای اخطاری که با رنگ نارنجی نمایش داده می‌شود.
- پیغامهای خطا که با رنگ قرمز نمایش داده می‌شود.

۱۳-۴-۵) پنجره مستندات

این پنجره در هنگام راست تلیک کردن کاربر بر روی اجزای کتابخانه و انتخاب گزینه "نمایش مستندات" نشان داده می‌شود. شکل ۹-۱۳ پنجره مستندات را نمایش می‌دهد.



شکل ۹-۱۳. پنجره مستندات.

۱۳-۵) محاوره

محاوره نوعی زیر پنجره است که به طور قراردادی قابل مشاهده نمی‌باشد. کاربر برای فعال کردن این گزینه باید اقداماتی انجام دهد.

۱۳-۵-۱) محاوره جدید

با انتخاب از منوی File>New>Modal Type می‌توان محاوره جدید را ایجاد کرد. نوع مدل می‌تواند به صورت مدل، کلاس، رابط، رکورد، تابع و پکیج باشد.

۱۳-۵-۲) محاوره شبیه سازی

با انتخاب از منوی Simulation>Simulate یا با تلیک بر روی دکمه Simulate در نوار ابزار می‌توان محاوره شبیه‌سازی را اجرا نمود. شکل ۶-۱۳ یک محاوره شبیه‌سازی را نمایش می‌دهد. محاوره شبیه‌سازی شامل متغیرهای شبیه‌سازی می‌باشد که کاربر می‌تواند با توجه به نیاز خود نسبت به قراردادن مقدار هر متغیر اقدام کند. متغیرهای شبیه‌سازی عبارتند از:

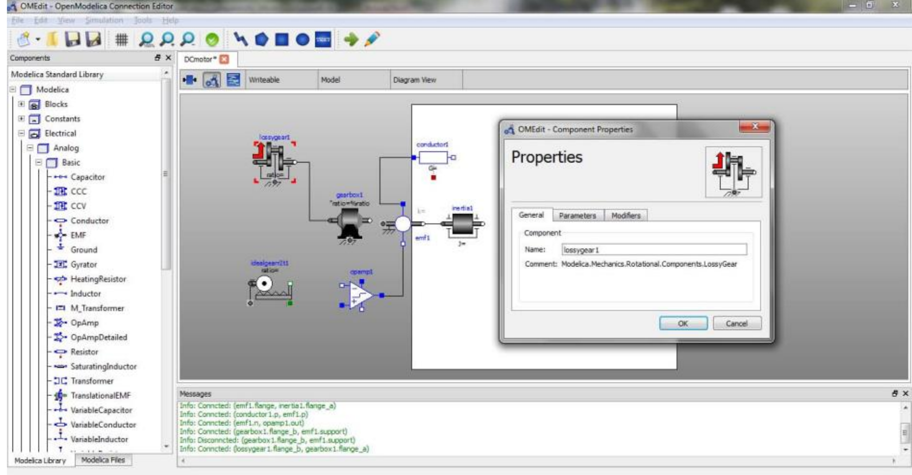
۱- بازه شبیه‌سازی
- زمان شروع
- زمان پایان

۲- بازه خروجی
- تعداد بازه
- بازه خروجی

۳- اجماع
- روش
- دقت
- اندازه گام

۱۳-۵-۳) محاوره خواص مدل

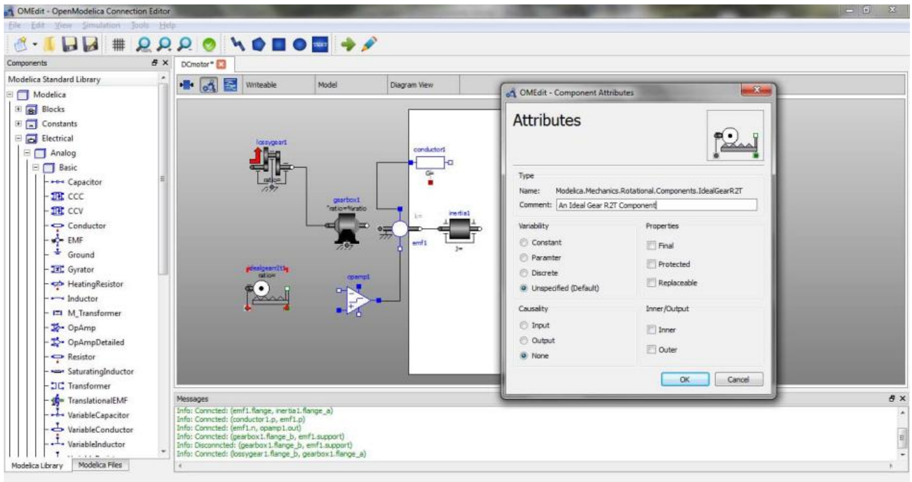
مدلهایی که در پنجره طراحی قرار دارند با تغییر خواص آن می‌توانند تعریف شوند. با راست تلیک کردن بر روی یک مدل خاص و انتخاب گزینه Properties می‌توان به محاوره خواص مدل دسترسی پیدا کرد که در شکل ۱۰-۱۳ نمایش داده شده است.



شکل ۱۰-۱۳. پنجره خواص.

۴-۵-۱۳) پنجره خواص مدل

با راست تلیک کردن بر روی مدل و پنجره طراحی و انتخاب گزینه Attributes این پنجره فعال می‌شود. شکل ۱۱-۱۳این پنجره را نشان می‌دهد.



شکل ۱۱-۱۳. پنجره خواص.

بخش سوم
آموزش زبان Modelica

MODELICA

فصل ۱۴ مروری کوتاه برModelica

در این بخش با امکانات زبان قدرتمند Modelica آشنا خواهیم شد. برخی از این امکانات را در دو بخش پیشین مورد بررسی قرار داده‌ایم و با آن آشنایی داریم؛ در این بخش بر زبان Modelicaتمرکز خواهیم کرد و امکانات و ساختار این زبان را بررسی خواهیم نمود. مثالهای داده شده در اغلب محیطهای Modelica قابل استفاده است و در این فصل تأکید بر مشترکان و مطالب پرکاربرد این زبان است که لازم است شما آنها را بدانید.

این یک مرور کوتاه بر زبان مدل‌سازیModelica است که شما را قادر می‌سازدتا کد نویسی Modelica را به سرعت شروع کنید. زبانModelica یک زبان مدل‌سازی است که امکان پیاده‌سازی مدل ریاضی سیستم‌های فیزیکی و شبیه‌سازی کامپیوتری آنها را فراهم می‌کند. این زبان یک زبان برنامه‌نویسی شیءگرا بر اساس معادلات است، که برای کاربردهای محاسباتی با پیچیدگی زیاد که نیازمند عملکرد بالایی می‌باشند، خلق شده است.

- زبان Modelica به جای اینکه براساس عبارات دستوری باشد،بر اساس معادلات است. این مساله امکان استفاده از مدل‌سازی غیرسببی را فراهم می‌کند. از آنجایی که در این روش جهت مشخصی برای انتقال اطلاعات در معادلات، مشخص نمی‌شود، امکان استفاده مجدد از کلاس‌ها بهتر فراهم می‌گردد. بنابراین کلاس‌های Modelica می‌توانند چیزی بیشتر از توابعی تک جهته باشند.
- زبان Modelica دارای قابلیت مدل‌سازی سیستم‌های فیزیکی مختلف (چند دامنه‌ای) به طور توام می‌باشد. یک مدل می‌تواند از قطعات سیستم‌های مختلف فیزیکی مانند الکتریکی، مکانیکی، ترمودینامیکی، هیدرولیکی، بیولوژیکی و کنترلی تشکیل شده باشد، که به یکدیگر متصل شده‌اند.
- زبان Modelica شیءگرا بوده و از مفهوم عمومی کلاس‌ها استفاده می‌کند که موجب بهبود امکان استفاده مجدد کلاس‌ها و ارزیابی مدل می‌گردد.
- زبان Modelica دارای کتابخانه استاندارد قطعات متداول، به همراه ابزارهایی برای ساخت قطعات جدید و اتصال قطعات می‌باشد. بنابراین Modelica زبان بسیار مناسبی برای

تعریف ساختاری سیستم‌های پیچیده فیزیکی است و تا حدودی هم می‌تواند برای سیستم-های نرم‌افزاری مورد استفاده قرار گیرد.

این فصل امکان مرور مختصری بر نکات اصلی زبان بدون درگیر کردن خواننده با جزییات کامل آن را فراهم می‌آورد.

۱۴-۱) شروع

برنامه‌های Modelicaبا استفاده از کلاس‌ها ساخته می‌شوند. از یک کلاس، می‌توان تعدادی نمونه ایجاد کرد که آنها را شیء می‌نامیم. می‌توان به کلاس‌ها به عنوان مجموعه‌ای از نقشه‌ها که در کارخانه برای درست کردن اشیاء استفاده می‌شود، نگاه کرد. در این حالت مترجم Modelica و برنامه اجرایی مثل کارخانه عمل می‌کنند.

یک کلاس Modelica شامل دو جزء اساسی است که عبارتند ازتعریف متغیرها و تعریف معادلات. متغیرها، اطلاعات مربوط به نمونه‌هایکلاس را نگهداری می‌کنند و وضعیت نمونه را تعیین می‌کنند. معادلات یک کلاس،رفتار نمونه‌های آن کلاس را مشخص می‌کنند.

طبق یک سنت قدیمی، اغلب آموزش هر زبان برنامه‌نویسی را با نوشتن برنامه‌ای ساده برای نمایش رشته "Hello World"شروع می‌کنند؛ اما از آنجایی که Modelica زبانی برای شبیه‌سازی است و براساس معادلات بنا شده است، چاپ کردن یک رشته، کار معقولی به نظر نمی‌رسد. پس مناسب‌تر است که برنامه Hello World ما یک معادله دیفرانسیل ساده را حل کند. در فصلهای گذشته یادگیری زبان Modelica را با حل یک معادله دیفرانسیل ساده شروع کردیم، در اینجا در خصوص همان مثال توضیحات بیشتری ارائه خواهیم داد.

$$\dot{x} = -a * x \qquad (۱۴-۱)$$

متغیر x در این معادله متغیر حالت است که می‌تواند با زمان تغییر کند. $\dot{x}$مشتق زمانی x است.

از آنجایی که اغلب برنامه‌های Modelicaکه مدل نامیده می‌شوند به عنوان یک کلاس تعریف می-گردند، ما نیز اولین برنامه خود را به عنوان یک کلاس تعریف می‌کنیم. کلاس ساختار بنیادی زبان modelica است، در واقع در modelica همه چیز کلاس است:

```
class HelloWorld
	Real x(start = 1);
	parameter Real a = 1;
equation
	der(x) = -a*x;
end HelloWorld;
```

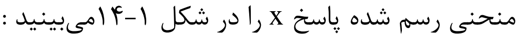
برای نوشتن کدهای Modelica می‌توانید از برنامه‌های ویرایشگر متنی دلخواه خود و یا از محیط‌های برنامه نویسی Modelicaاستفاده کنید. در بخش اول و دوم این کتاب با محیط‌های MathModelica و OpenModelica آشنا شدید. مثال‌های این بخش را می‌توانید در هر یک از محیط‌ها پیاده‌سازی و اجرا نمایید. پیاده‌سازی زبان Modelica در محیط‌های مختلف تقریباً مشابه است و اغلب مدل‌های نوشته شده در یک محیط را می‌توان با اندکی تغییر به محیط دیگر انتقال داد. مثال‌های این بخش قبلاً در DrModelica پیاده‌سازی شده است و می‌توانید به راحتی آنها را بیابید و به جای نوشتن مثال‌ها، وقت خود را به یادگیری امکانات زبان Modelica اختصاص دهید. پس از نوشتن کد برنامه دستور شبیه‌سازی را در محیط برنامه‌نویسی خود فراخوانید. این کار باعث ترجمه کد Modelica برنامه شما به یک کد میانی، معمولاً C، می‌گردد. این کد میانی نیز به کد ماشین ترجمه شده و با یک حل‌کننده عددی ODE (معادلات دیفرانسیل) یا DAE (معادلات جبری-دیفرانسیل) اجرا می‌گردد تا پاسخ x به عنوان تابعی از زمان پیدا شود.

در محیط OpenModelica دستور زیر شبیه سازی را برای زمان بین صفر تا چهار ثانیه انجام خواهد داد:

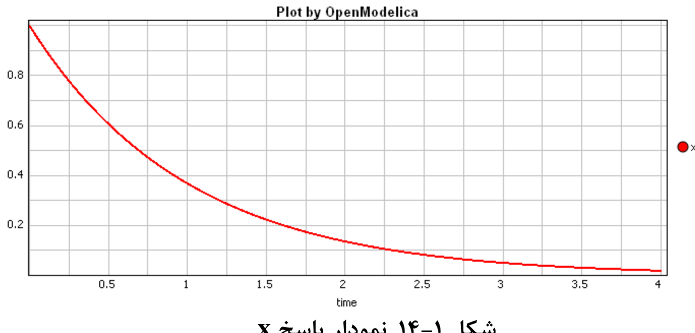
```
simulate( HelloWorld, startTime=0, stopTime=4 )
```

از آنجایی که x تابعی از زمان است، می‌توان توسط دستور رسم نمودار، آن را رسم کرد:

```
plot2(x)
```



منحنی رسم شده پاسخ x را در شکل ۱-۱۴می‌بینید :



شکل ۱-۱۴. نمودار پاسخ x.

حالا یک مدل کوچک Modelica داریم که کاری انجام می‌دهد، اما واقعاً چه کاری؟

برنامه شامل تعریف کلاس HelloWorld به شکل دو فیلد تعریف متغیر و یک معادله می‌باشد. فیلد اول متغیر x را تعریف می‌نماید که مقدار اولیه‌اش برابر ۱ قرار داده شده است (مقدار متغیر در

شروع شبیه سازی). همه متغیرها در Modelica یک مقدار اولیه دارند که به طور پیش‌فرض این مقدار صفر است. اختصاص یک مقدار اولیه متفاوت توسط عبارتی که اصلاح کننده خوانده می‌شود داخل پرانتزهایی بعد از نام متغیر انجام می‌شود؛ برای مثال اصلاح کننده start=1که مقدار اولیه متغیر را برابر یک قرار می‌دهد.

فیلد دوم پارامتر a را به عنوان یک ثابت تعریف می‌کند و در شروع شبیه‌سازی به آن مقدار ۱ نسبت داده می‌شود. ثابت‌ها در زبان Modelica توسط کلمه کلیدی parameter تعریف می‌گردند. مقدار یک ثابت در طول زمان شبیه‌سازی ثابت است اما یکی از پارامترهای شبیه‌سازی است که می‌تواند بین شبیه‌سازی‌ها مقدار آنها را تغییر داد. تغییر مقدار ثابت‌ها با استفاده از دستوراتی در محیط شبیه‌سازی امکانپذیر است، برای مثال ما می‌توانیم شبیه‌سازی را به ازای مقدار متفاوت a بار دیگر اجرا کنیم.

همچنین توجه کنید که هر متغیر نوع خاصی دارد که باید هنگام تعریف متغیر قبل از نام متغیر مشخص می‌گردد. در اولین برنامه هر دو پارامتر x و a از نوع Real تعریف شده‌اند.

تک معادله موجود در این مثال مشخص می‌کند که مشتق x برابر a- ضربدر در x می‌باشد. در Modelica علامت مساوی = همیشه معنی برابر می‌دهد، علامت مساوی در Modelica مانند سایر زبانهای برنامه‌نویسی نیست و نشان دهنده یک معادله است، نه یک عبارت دستوری برای انتساب یک مقدار به متغیر. مشتق زمانی یک متغیر با شبه-تابع der() نمایش داده می‌شود.

۱۴-۲) متغیرها

مثال بعدی یک مدل نسبتا پیچیده‌تر را نمایش می‌دهد، که یک نوسانگر Van der Pol است. این مثال همچنین شامل اعلان متغیرهای اضافی برای نشان دادن داده‌های پیاده سازی شده اولیه در Modelica است. توجه داشته باشید در اینجا از کلمه کلیدی model به جای class و با همان کاربرد استفاده شده است.

```
model VanDerPol "Van der Pol oscillator model"
	Real x(start = 1) "Descriptive string for x";//x starts at 1
	Real y(start = 1) "y coordinate"; //y starts at 1
	parameter Real lambda = 0.3;
	Boolean bb = true;
	String dummy = "dummy string";
	Integer fooint = 0;
equation
	der(x) = y; // This is the first equation
	der(y) = -x + lambda*(1 - x*x)*y; /*This is the second
	differential equation */
end VanDerPol;
```

این مثال شامل تعریف دو متغیر حالت x و y که هر دوی آنها از نوع Real و در زمان شروع شبیه‌سازی معمولاً در زمان صفر دارای مقدار اولیه۱ هستند، می‌باشد. در ادامه تعریف پارامتر ثابت lambda که پارامتر شبیه‌سازی نامیده می‌شود را می‌بینید.

کلمه کلیدی parameter مشخص می‌کندکه پارامتر در خلال مدت شبیه‌سازی ثابت است، ولی می‌تواند قبل از اجرای شبیه‌سازی و یا در بین شبیه‌سازی‌ها مقدار دهی اولیه شود. یعنی parameter نوع خاصی از ثابت‌ها است که به عنوان متغیر ایستا تعریف می‌گردد که یکبار مقدار دهی اولیه شده است در حین شبیه‌سازی مقدارش تغییر نمی‌کند. یک parameter یک مقدار ثابت است که اصلاح رفتار مدل را برای کاربر ساده می‌کند. برای مثال امکان تغییر مقدار lamda که تأثیر شدیدی بر رفتار نوسانگر Van der Pol دارد.

برعکس، در Modelica مقدار یک ثابت که با پیشوند constant تعریف می‌گردد، هیچ وقت عوض نمی‌شود. مترجم هنگام ترجمه برنامه مقدار عددی این ثابتها را در جایش قرار می‌دهد و برنامه را ترجمه می‌کند. بنابراین برای تغییر مقدار یک ثابت بعد از اعمال تغییر باید برنامه را بار دیگر ترجمه نمود و برعکس پارامترها، تغییر مقدار ثابت در هر شبیه‌سازی ممکن نیست.

در نهایت سه متغیر اضافی تعریف کرده‌ایم که واقعاً در مدل نوسانگر به آنها نیاز داریم: متغیر منطقی[1]bb که اگر مقدارش مشخص نشده باشد مانند همه متغیرهای منطقی زبان Modelica به صورت پیش فرض دارای مقدار اولیه false می‌باشد، متغیر رشته‌ای dummy و متغیر صحیح fooint.

زبانModelica به طور ذاتی از انواع داده‌های اعشاری، صحیح، منطقی و رشته‌ها پشتیبانی می‌کند. این انواع اولیه شامل داده‌هایی است که Modelica به طور مستقیم آنها را درک می‌کند و نیازی به تعریف نوع از طرف کاربر نیست. نوع متغیرهای دیگر باید به طور واضح تعریف شود. انواع اصلی داده‌های Modelica عبارتند از:

Boolean Either true or false
Integer Corresponding to the C int data type, usually 32-bit two's complement
Real Corresponding to the C double data type, usually 64-bit floating-point
String String of 8-bit characters

در پایان این برنامه، یک قسمت معادلات وجود دارد که با کلمه کلیدی equation شروع می‌شود و شامل دو معادله به هم وابسته است که دینامیک مدل را تعریف می‌کنند.

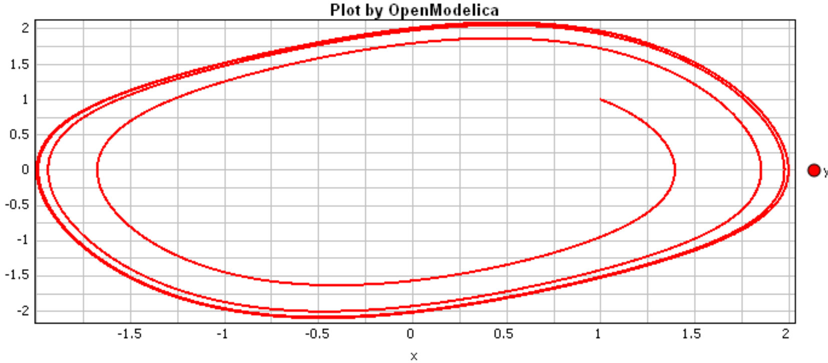
[1] boolean

برای روشن شدن رفتار مدل ، دستور شبیه سازی نوسانگر Van der Pol را به مدت ۲۵ ثانیه از زمان صفر وارد می‌کنیم:

```
simulate(VanDerPol, startTime=0, stopTime=25 )
```

سپس نمودار پارامتری متغیرهای حالت مدل نوسانگر Van der Pol را با استفاده از دستور زیر رسم می‌کنیم:

```
plotParametric( x, y )
```



شکل ۲-۱۴. نمودار مدل Van der pol.

۱۴-۳) توضیحات

توضیحات متنی برای خوانایی بیشتر کدهای نرم‌افزاری به آنها اضافه می‌گردند. وجود این توضیحات اختیاری است. اما با توجه به اهمیت مستندسازی در برنامه‌نویسی، مناسب است که توضیحاتی به هر بخش از برنامه خود بیفزایید. زبانModelica دارای سه نمونه توضیحات است، در مثال بالا از هر سه نوع آنها استفاده شده است. بخش توضیحات به کد برنامه شما خوانایی بالایی می‌دهد تا فهم کدهای شما برای برنامه نویسی‌هایی که برنامه شما را در آینده می‌خوانند، آسان شود. آن برنامه‌نویس ممکن است خود شما در یک ماه، یا یک سال آینده باشید! با این کار از به زحمت افتادن در آینده جلوگیری می‌کنید. همچنین نوشتن توضیحات به شما کمک می‌کند تا برنامه خود را بهتر درک کنید و چون ممکن است مجبور شوید بارها در مورد ساختار برنامه خود فکر کنید، از اشتباهات شما خواهد کاست.

اولین نوع توضیحات نوع رشته‌ای است که درون گیومه قرار می‌گیرد مثل "a comment"، معمولاً به صورت اختیاری بعد از تعریف متغیر یا در شروع تعریف کلاس قرار می‌گیرد. این توضیحات توسط محیط برنامه‌نویسی Modelica برای ایجاد منو یا تولید فایل کمک استفاده می‌شوند. از نقطه نظر دستوری با توجه به اینکه این توضیحات بخشی از ساختار عبارات هستند نمی‌توان آن‌ها را جزء توضیحات به حساب آورد. برای مثال در برنامه نوشته شده قبلی توضیحات کلاس Van der Pol و متغیرهای x وy را ببینید.

دو نوع توضیح دیگر توسط مترجم Modelica نادیده گرفته می‌شوند و فقط برای برنامه‌نویسان Modelica سودمند می‌باشند. توضیحات پس از علامت // و متن بین /*and the next*/ توسط مترجم نادیده گرفته می‌شوند و در برنامه ترجمه شده، دیده نخواهند شد.

۱۴-۴) ثابت‌ها

ثابت در Modelica می‌تواند مقادیر صحیح مثل 4,753,078 ؛ مقادیر اعشاری مثل 3.14159 0.5, 2.735E-10 و مقادیر رشته‌ای مثل "hello world", "red" باشند.

ثابت‌های نامگذاری شده به دو دلیل توسط برنامه‌نویس‌هااستفاده می‌شوند. دلیل اول این است که به کار بردن اسم ثابت‌ها اگر مناسب انتخاب گردد می‌تواند جایگاه و کاربرد ثابت‌ها در برنامه را بهتر مشخص کند که یک نوع مستندسازی مناسب می‌باشد و توضیح می‌دهد مقدار مورد نظر به چه منظور استفاده شده است. دلیل مهم‌تر این است که، وقتی مقدار ثابت (نام گذاری شده) جایی در برنامه تعریف می‌گرددر صورتی که نیاز باشد مقدار ثابت را در کل برنامه تغییر داده یا تصحیح کنیم، فقط کافی است در یک جا (در تعریف ثابت) مقدار آن را تغییر دهیم. بنابراین تعریف ثابت، نگهداری برنامه را ساده‌تر می‌کند.

ثابت‌های با نام در زبان Modelica با استفاده از کلمات پیشوندی parameter یا constant ایجاد و تعریف می‌شوند. برای مثال به چند نمونه از عبارت‌های به کار رفته برای تعریف انواع ثابت‌ها توجه نمایید.

```
constant     Real       PI = 3.141592653589793;
constant     String     redcolor = "red";
constant     Integer    one = 1;
parameter    Real       mass = 22.5;
```

از آنجایی که مقدار پارامترها را می‌توان در ابتدای شبیه‌سازی تعیین نمود یا از یک فایل ورودی خواند، مقداردهی اولیه آنها ضروری نیست؛ هرچند اینکار قابل انجام است. برای مثال:

```
parameter Real mass, gravity, length;
```

اگر مقدار یک متغیر عددی در هنگام تعریف مشخص نباشد، در آغاز شبیه‌سازی مقدار اولیه صفر به آن تعلق می‌گیرد. مقدار اولیه پیش فرض برای متغیرهای منطقی مقدار false است و اگر در هنگام تعریف متغیر رشته‌ای، مقدارش ذکر نشده باشد، خالی فرض می‌گردد. استثناء این قوانین در نتایج توابع و متغیرهای محلی است، که مقادیرشان به صورت پیش‌فرض، تعریف نشده است.

۱۴-۵) مدلسازی شیءگرای ریاضی

زبان‌های برنامه‌نویسی شیءگرای سنتی مثل Simula، ++C، Java و Smalltalk و همچنین زبان‌های رویه‌ای مثل فرترن و C از انجام عملیات بر روی حالت پشتیبانی می‌کنند. حالت یک برنامه شامل مقادیر متغیرها و داده‌های اشیاء می‌باشد. در یک برنامه معمولا تعداد اشیاء به صورت پویا تغییر می‌کند.

دیدگاهModelica بر شیءگرا بودن متفاوت است. Modelica بر مدل‌سازی ریاضی ساختاریافته تاکید دارد. شیءگرا بودن یک مفهوم ساختاری است که برای کار با سیستم‌های بزرگ و پیچیده به ما کمک می‌کند. یک مدل در زبان Modelica اصولاً یک تعریف ریاضی برای تشریح رفتار یک سیستم است. مشخصه‌های دینامیک سیستم به صورت معادلات ریاضی بیان می‌گردند.

در این زبان مفهوم برنامه‌نویسی بر پایه تعریف معادلات سیستم بنا شده است. این معادلات رفتار حاکم بر سیستم را به صورت ریاضی مدل می‌کنند. در مقابل زبان‌های برنامه‌نویسی رویه‌ای قرار دارند. در آنها الگوریتم برنامه به صورت گام به گام پاسخ مطلوب سیستم را ایجاد می‌کند.

بنابراین، دیدگاه شیءگرای Modelica، از نقطه نظر مدلسازی شیءگرای ریاضی می‌تواند به صورت زیر خلاصه شود:

- مشخصات مدل دینامیک (بر اساس ساختار) به وسیله معادلات ریاضی بیان می‌شوند و قابلیت استفاده دوباره از معادلات را فراهم می‌کند.
- شیء مجموعه‌ای از متغیرهاو معادلات می‌باشد که رفتار سیستم را مدل می‌کند.
- شیءگرایی در این زبان فراتر از ارسال داده‌ها بین اجزاء است.

توصیف یک سیستم و رفتارش (به روش شیءگرا) در زبان Modelica نسبت به زبان‌های برنامه-نویسی شیءگرای دیگر دارای سطح بالاتری از خلاصه‌سازی می‌باشد. زیرا برخی از جزئیات پیاده‌سازی حذف می‌گردند. برای مثال برای انتقال داده‌ها بین اشیاء نیازی به کد نویسی نیست. این کدبر اساس معادلات داده شده به طور خودکار توسط زبان Modelica ایجاد می‌گردد.

درست مانندسایر زبان‌های شیءگرا، کلاس‌ها الگویی برای ایجاد اشیا هستند. هم متغیرها و هم معادلات می‌توانند بین کلاس‌ها موروثی باشند. تعریف توابع نیز می‌تواند موروثی باشد. در زبان

Modelica رفتار سیستم به جای روش‌ها (methods) به وسیله معادلات تعریف می‌شوند. همچنیندر زبان Modelica امکاناتی برای تعریف کدهایی مانند توابع نیز فراهم شده است. توابع این زبان، ساختاری مشابه با توابع زبان‌های برنامه نویسی سنتی دارند.

۱۴-۶) کلاس‌ها و نمونه‌ها

زبان Modelica مثل هر زبان کامپیوتری شیءگرا، مفهوم کلاس و اشیاء (که نمونه هم خوانده می‌شود) را به عنوان ابزارهایی برای حل مسائل برنامه‌نویسی و مدلسازی فراهم نموده است. هر شیء در Modelica از یک کلاس به وجود می‌آید که اطلاعات و رفتار آن شیء را تعریف می‌کند. هر کلاس می‌تواند دارای سه نوع عضو باشد:

- فیلدها: شامل اطلاعات متغیرهاییک کلاس و نمونه‌های آن می‌باشند. فیلدها نتیجه محاسباتی که از حل معادلات یک کلاس به همراه معادلات دیگر کلاس‌ها حاصل می‌شوندرا در خود ذخیره می‌کنند. در خلال حل مسائل وابسته به زمان، فیلدها نتایج حل (در زمان جاری) را ذخیره می‌کنند.
- معادلات: رفتار یک کلاس را مشخص می‌کنند. روشی که معادلات با معادلات کلاس‌های دیگر در تعامل‌اند، فرآیند حل را تعیین می‌کند؛ یعنی اجرای برنامه.
- کلاس‌ها: خود نیز می‌توانند عضو یک کلاس دیگر باشند.

در اینجا تعریف یک کلاس ساده برای مدل‌سازی یک نقطه در فضای سه بعدی را می‌بینید:

```
class Point "point in a three-dimensional space"
          public Real x;
          Real y, z;
end Point;
```

کلاس Point دارای سه فیلد است که مختصات x، y و z یک نقطه را بیان می‌کنند و این کلاس (هنوز) معادله‌ای ندارد. اعلان کلاس شبیه به این مانند یک الگو عمل می‌کند. این الگو تعیین می‌کند که نمونه‌های ایجاد شده از این کلاس چه شکلی خواهند داشت.

عضوهای یک کلاس می‌تواند در دو سطح قابل رؤیت باشند. تعریف public قبل از متغیر x، به این معنی است که هر کدی که به نمونه Point دسترسی داشته باشد می‌تواند مقدار x را بخواند یا آن را به روز کند، این حالت پیش فرض هر متغیر است، یعنی اگر متغیر از نوع دیگری تعریف نگردد، برای آن متغیر این حالت در نظر گرفته خواهد شد. سطح بعدی محدوده رؤیت توسط کلمه کلیدی protected مشخص می‌شود، به این معنا که فقط کدهای داخل کلاس و همینطور کدهایی از این کلاس ارث می‌برند، قابلیت دسترسی به این متغیر را خواهند داشت.

توجه داشته باشید که وقتی یکی از کلمات کلیدی protected یا public ذکر می‌شود، همه متغیرهایی که بعد از آن تعریف می‌شوند از این سطح رویت پیروی می‌کنند. مگر اینکه یکی دیگر از این کلمات کلیدی بیاید.

۱۴-۶-۱) ایجاد نمونه‌ها

در Modelica، با اعلان نمونه‌ای از کلاس‌ها، اشیا به طور ضمنی ایجاد می‌شوند. این روش با زبان‌های شیءگرای دیگر مثل جاوا و C++ ، که ایجاد اشیا جدید با کلمه کلیدی new انجام می‌شود، تفاوت دارد. برای مثال، به منظور ایجاد ۳ نمونه از کلاس Point ما فقط سه متغیر از نوع Point اعلان می‌کنیم:

```
class mypoints
        Point point1;
        Point point2;
        Point point3;
end mypoints;
```

بدین ترتیب با استفاده از سه شیء تعریف شده بر اساس کلاس Point یک کلاس جدید به نام mypoints به وجود آمده است. از آنجا که با استفاده از سه نقطه می‌توان یک صفحه را در فضای سه بعدی تعریف نمود، با استفاده از کلاس mypoints می‌توان اشیایی به وجود آورد که هر کدام بیانگر یک صفحه در فضا می‌باشند.

حال می‌خواهیم بدانیم که کجا می‌توان اشیایی از کلاس mypointsرا ایجاد نمود؟ برای پاسخ به این سوال باید توجه داشت که در زبان Modelica یک کلاس اصلی با عنوان "main" وجود دارد که همیشه نمونه‌ای از آن ایجاد می‌شود. بنابراین پاسخ بدین صورت است اگر لازم باشد با اشیای ایجاد شده از نوع mypoints کلاس جدیدی تعریف کنیم، این اشیاء در آن کلاس تعریف می‌شوند. در غیر این صورت کلاس "main" مکانی مناسبی برای تعریف و استفاده از اشیاء مختلف برای مدل‌سازی یک سیستم یک پارچه می‌باشد.

وقتی نمونه‌ای ازکلاس اصلی ایجاد می‌گردد، نمونه‌ای از همه اعضاء (فیلدهای) این کلاس ساخته می‌شود و همین طور نمونه‌هایی از فیلدهای هر یک از این اعضا ایجاد می‌گردد و به همین ترتیب. برای ایجاد نمونه‌ای از mypoints، می‌توان کلاس mypoints را به عنوان یک فیلد در کلاس اصلی تعریف کرد. در مثالی که در ادامه می‌آید، نمونه‌ای ازهر دو کلاس mypoints و foo1 ایجاد شده است.

```
class foo1
        ...
end foo1;
```

```
class foo2
        ...
end foo2;
...

class mypoints
        Point point1;
        Point point2;
        Point point3;
end mypoints;

class main
        mypoints pts;
        foo1 f1;
end main;
```

فیلدهای کلاس‌های Modelica در هر شئ نمونه‌سازی می‌شوند، یعنی یک فیلد در یک شئ، به صورت مجزا از فیلدهای همنام در اشیاء دیگر از همان کلاس نمونه‌سازی می‌گردد. اعلان متغیری در یک کلاس به صورتی که در تمام نمونه‌های ایجاد شده از کلاس مشترک و قابل دسترس باشد تاکنون در زبان Modelica پیاده‌سازی نشده است[این موضوع شاید در ویرایش‌های جدید رفع گردد].

۱۴-۶-۲) مقدار دهی اولیه

مسأله دیگری که وجود دارد مقدار دهی اولیه به فیلدها است. اگر چیزی مشخص نشود مقدار پیش فرض تمامی متغیرهای عددی صفر خواهد بود، به جز نتایج توابع یا متغیرهای محلی که مقدار آنها به صورت پیش‌فرض مشخص نشده (undefined) هستند. همانطور که در گذشته گفته شد مقادیر اولیه دیگری را می‌توان با نوشتن خاصیت مقدار اولیه در اعلان متغیر مشخص نمود. در مثال زیر برای یکی از فیلدهای کلاس mypoints مقدار اولیه مشخص می‌کنیم:

```
class mypoints
        Point point1(start={1,2,3});
        Point point2;
        Point point3;
end mypoints;
```

روش دیگر، مقدار دهی اولیه point1 در هنگام ایجاد نمونه از کلاس mypoints می‌باشد (همانطور که در کد زیر می‌بینید):

```
class main
        mypoints pts(point1.start={1,2,3});
```

```
    foo1 f1;
end main;
```

۱۴-۶-۳) کلاسهای محدود

مفهوم کلاس در زبان Modelica یک مفهوم بنیادی است و برای اهداف مختلفی استفاده می‌شود. تقریباً هر چیزی در Modelica یک کلاس است. به منظور بالا بردن خوانایی و امکان نگهداری کدها، کلمات کلیدی ویژه‌ای برای استفاده از مفهوم کلاس در کاربردها مختلف تعریف شده است. کلمات کلیدی model, connector, record, block و type می‌توانند در شرایط مناسب به جای class استفاده شوند. به عنوان مثال record کلاسی است که برای نگهداری ساختار رکورد داده‌ها (مجموعه‌ای از داده‌ها) استفاده می‌شود و نمی‌تواند شامل معادلات باشد. یک block کلاس با سببیت مشخص است، یعنی دارای ورودی و خروجی مشخص است و همیشه خروجی با استفاده از مقدار ورودی و انجام محاسبات مشخص به دست می‌آید. یک connector کلاسی برای تعریف ساختار نقاط اتصال است و نباید شامل معادلات باشد؛ type کلاسی است که می‌تواند برای تعریف یک نام مستعار برای نوع از پیش تعریف شده، آرایه یا رکورد یا توسعه آنها استفاده گردد. برای مثال:

```
type vector = Real[3];

record person
    Real age;
    String name;
end person;
```

ایده استفاده از کلاس‌های محدود با توجه به این که کاربر مجبور نیست مفاهیم مختلف را یاد بگیرد مفید می‌باشد. کاربر فقط یک مفهوم کلاس را می‌آموزد. تمام مشخصات کلاس‌های محدود مانند دستور زبان و روش تعریف، ایجاد نمونه، توارث و مشخصات عمومی برای همه انواع کلاس‌های محدود یکسان هستند. علاوه بر این ساختمان مترجم Modelica ساده می‌گردد، زیرا فقط لازم است ساختار و دستور زبان لازم برای ایجاد کلاس و مواردی برای کنترل محدودیت‌های کلاس ایجاد شود.

مفهوم بسته (در بخش‌های گذشته کلمه کتابخانه را به جای بسته استفاده نمودیم) و تابع در Modelica دارای اشتراک زیادی با مفهوم کلاس است ولی از آنجا که این مفاهیم دارای دستور زبان خاص خودشان می‌باشند، جزء کلاس‌های محدود محسوب نمی‌شوند.

۱۴-۶-۴) استفاده دوباره از کلاس‌های اصلاح شده

در زبان Modelica، مفهوم کلاس امکان استفاده مجدد از اشیاء را فراهم نموده است. در این زبان مقررات و روش‌هایی برای تطبیق یا تغییرات کلاس‌ها وجود دارد که استفاده مجدد از کلاس‌ها و کتابخانه‌های موجود را آسان می‌کند. برای این کار از دستوراتی که اصلاح‌گر (modifier) نامیده

می‌شوند، استفاده می‌شود. برای مثال فرض کنید، می‌خواهیم دو مدل فیلتر را به صورت سری به هم وصل کنیم. به جای ایجاد دو کلاس مجزا برای این فیلترها بهتر است که یک کلاس مشترک ایجاد کرده و دو نمونه از این کلاس را با اعمال تغییرات لازم بسازیم؛ و آنها را به هم متصل کنیم. در مثال زیر ابتدا یک کلاس فیلتر ساخته خواهد شد و سپس دو نمونه فیلتر پایین گذر با تغییر کلاس اصلی ایجاد می‌گردد و این دو فیلتر را به هم متصل می‌کنیم:

```
model LowPassFilter
	parameter Real T = 1;
	Real u, y(start = 1);
equation
	T*der(y) + y = u;
end LowPassFilter;
```

می‌توان از این کلاس (مدل) برای ایجاد دو نمونه فیلتر با ثابت زمانی متفاوت استفاده و توسط معادله F2.u=F1.y آنها را به هم متصل کرد.

```
model FiltersInSeries
	LowPassFilter F1(T = 2), F2(T = 3);
equation
	F1.u = sin(time);
	F2.u = F1.y;
end FiltersInSeries;
```

توجه کنید که در اینجا چگونه از اصلاح کننده‌ها استفاده شد، همچنین به استفاده از معادلات مشخصه T=2 و T=3 برای اصلاح ثابت زمانی فیلترها در زمان ایجاد نمونه‌های F1 و F2 از کلاس فیلتر توجه نمایید. متغیر مستقل زمان، با کلمه کلیدی time مشخص می‌گردد. حال اگر بخواهیم مدل FilterInSeries را در سطح بالاتری استفاده نماییم، برای مثال F12، هنوز می‌توان ثابت زمانی هر یک از فیلترها را با رعایت سلسله مراتب قرارگیری، اصلاح نمود؛ همانطوری که برای F1 و F2 در ادامه خواهید دید:

```
model ModifiedFiltersInSeries
	FiltersInSeries F12(F1(T = 6), F2(T = 11));
end ModifiedFiltersInSeries;
```

۱۴-۶-۵) کلاس‌های داخلی

نوع در کلاس‌های داخلی (کلاسهایی که به صورت ذاتی در Modelica وجود دارند)، شامل انواع اولیه Real، Integer، Boolean و String است. این کلاس‌ها نیز اکثر مشخصات کلاس را دارا هستند. برای مثال می‌توانند ارث ببرند، اصلاح شوند. مقدار آنها (یکی از خاصیت‌های این نوع کلاس‌ها است) در زمان اجرا قابل تغییر بوده و از طریق نام متغیر قابل دسترسی است. مقدار این کلاس‌ها با استفاده از علامت نقطه قابل دسترس نیست. برای مثال وقتی متغیری به نام x تعریف می‌کنید، مقدار این متغیر با استفاده از x (نام متغیر) در دسترس است و نمی‌توان از x.values برای دسترسی به مقدار آن استفاده کرد. سایر مشخصات این کلاس‌ها از طریق علامت نقطه قابل دسترس است برای مثل x.unit را می‌توان برای دسترسی به واحد اندازه‌گیری متغیر استفاده نمود.

برای مثال یک متغیر Real دارای مجموعه‌ای از مشخصات پیش‌فرض مثل واحد اندازه‌گیری، مقدار اولیه، مقدار ماکزیمم و مینیمم می‌باشد. این مشخصات را می‌توان در اعلان کلاس جدید تغییر داد، برای مثال:

```
class Voltage = Real(unit = "V", min = -220.0, max = 220.0);
```

۱۴-۷) توارث

یکی از بزرگترین منافع شیءگرا بودن، امکان توسعه دادن رفتار و مشخصات کلاس موجود است. کلاس اصلی، که به عنوان سوپرکلاس شناخته می‌شود، برای ایجاد نسخه‌های خاصی از آن کلاس که به عنوان زیرکلاس شناخته می‌شوند به کار خواهد رفت. در این فرآیند، رفتار و مشخصات کلاس اصلی یعنی فیلدها، معادلات و دیگر محتویات آن دوباره توسط زیرکلاس استفاده می‌گردد یا به اصطلاح به زیرکلاس به ارث می‌رسد.

اجازه بدهید به توسعه یک کلاس ساده در زبان Modelica به وسیله توارث نگاهی بیاندازیم، برای مثال کلاس Point که قبلاً معرفی شده است را توسعه خواهیم داد. در ابتدا ما دو کلاس Color و ColorData را تعریف می‌کنیم؛ کلاس ColorData اطلاعات رنگ را نگهداری می‌کند و یک ساختار داده‌ای است. کلاس Color، کلاس ColorData را به ارث می‌برد به عبارت دیگر کلاس Color برای توسعه دادن کلاس ColorData استفاده شده است و از آن برای نگهداری داده‌های رنگ استفاده می‌کند و همچنین یک معادله را به عنوان محدودیت به آن اضافه می‌کند. کلاس جدید ColoredPoint از چندین کلاس ارث می‌برد، فیلدهای موقعیت را از کلاس Point و فیلد رنگ‌ها و معادلات را ازکلاس Color به ارث می‌برد.

```
record ColorData
	Real red;
	Real blue;
```

```
        Real green;
end ColorData;

class Color
        extends ColorData;
equation
        red + blue + green = 1;
end Color;

class Point
        public Real x;
        Real y, z;
end Point;

class ColoredPoint
        extends Point;
        extends Color;
end ColoredPoint;
```

۱۴-۸) کلاس‌های عمومی

در خیلی از مواقع سودمند است که برای مدل‌ها و برنامه‌هایی که به صورت عمومی استفاده می‌گردند، الگوهای عمومی در نظر بگیریم. این کار باعث صرفه‌جویی در نوشتن قطعات زیادی از کدهای مشابه با ساختار ذاتی یکسان می‌گردد و کدنویسی و نگهداری نرم‌افزار را ساده‌تر می‌کند.

چنین ساختارهایی در زبان‌های مختلف برنامه‌نویسی وجود دارد. برای مثال الگوهای زبان Ada،C++ و انواع پارامترها در زبان‌های تابعی مثل Haskell یا Standard M را می‌توان نام برد. در Modelica ساختار کلاس‌ها به قدر کافی برای مدل‌سازی عمومی و برنامه‌نویسی قدرتمند است.

ذاتاً دوحالت از کلاس‌های کلی در Modelica وجود دارد: پارامترهای نمونه و پارامترهای نوع، توجه داشته باشید در این بخش از متن منظور ما از پارامترهای کلاس، پارامترهای شبیه‌سازی که با کلمه کلیدی parameter در کلاس تعریف می‌گردد، نیست (گرچه این متغیرها نیز می‌توانند پارامترهای کلاس باشند). پارامترهای قابل جایگزینی با پیشوند replaceable مشخص می‌گردند. ساختارهای virtual در برخی از زبانهای برنامه‌نویسی شیءگرا تقریباً معادل حالت خاصی از توابع قابل جایگزینی است.

۱۴-۸-۱) اشیاء به عنوان پارامترهای قابل جایگزینی کلاس

در ابتدا حالتی را ارائه می‌کنیم که پارامترهای کلاس نمونه‌ها و اشیاء هستند و می‌خواهیم آنها را به نمونه‌های دیگری تبدیل کنیم. در مثال زیر کلاس C دارای سه پارامتر است که با کلمه کلیدی

replaceable علامتگذاری شده‌اند. این پارامترها که اجزاء کلاس C هستند به صورتی تعریف شده‌اند که به ترتیب دارای نوع پیش‌فرض GreenClass، YellowClass و GreenClass باشند.

```
class C
	replaceable GreenClass obj1(p1=5);
	replaceable YellowClass obj2;
	replaceable GreenClass obj3;
equation
	...
end C;
```

حالا می‌خواهیم کلاس C2را با تعریف دو آرگومان حقیقی Obj1 و Obj2 برای کلاس C به گونه‌ای تعریف کنیم که به جای مقادیر پیش فرض GreenClass و YellowClass این دو آرگومان به ترتیب از نوع RedClass و GreenClass باشند. برای این که از تغییر نوع ناخواسته پارامترهای قابل تغییر جلوگیری شود باید کلمه کلیدی redeclare را قبل از آرگومان حقیقی به کار برد. اجزاء کلاس در صورتی که به صورت replaceable تعریف نشده باشد، قابل جایگذاری و تغییر نوع نیست.

```
class C2 = C(redeclare RedClass obj1,redeclare GreenClass obj2);
```

دستور بالا مشابه تعریف کردن کلاس C2 به صورت زیر است:

```
class C2
	RedClass obj1(p1=5);
	GreenClass obj2;
	GreenClass obj3;
equation
	...
end C2;
```

۱۴-۸-۲) انواع به عنوان پارامترهای قابل جایگزینی کلاس

پارامترهای کلاس می‌تواند انواع مختلف داده‌ها (type) باشد، این انواع را نیز می‌توان جایگزین نمود. برای مثال با فراهم کردن یک پارامتر نوع به نام ColouredClass در کلاس C، تغییر رنگ همه اجزاء از نوع ColouredClass بسیار آسان می‌گردد.

```
class C
	replaceable class ColouredClass = GreenClass;
	ColouredClass		obj1(p1=5);
```

```
        replaceable YellowClass          obj2;
        ColouredClass           obj3;
equation
        ...
end C;
```

حال کلاس C2 را با جایگزاری پارامتر ColouredClass کلاس C به مقدار BlueCalss ایجاد می‌کنیم.

```
class C2 =
C(redeclare class ColouredClass = BlueClass);
```

این کار با تعریف کلاس C2 به شکل زیر هم ارز است :

```
class C2
        BlueClass obj1(p1=5);
        YellowClass obj2;
        BlueClass obj3;
equation
        ...
end C2;
```

۹-۱۴) معادلات

همان طوری که قبلاً بیان کردیم، زبان Modelica اصولاً زبانی برپایه معادلات است. در مقابل زبان‌های برنامه‌نویسی دیگر که در آنها مقدار یک عبارت را به یک متغیر تخصیص می‌دهیم (دستورات جایگزین ساده)، معادلات انعطاف پذیری بیشتری به زبان Modelica می‌دهند. چون معادلات (برعکس دستورات جایگزین ساده) یک جهت مشخص جریان اطلاعات را تعیین نمی‌کنند. این موضوع کلید قابلیت مدل‌سازی فیزیکی و افزایش قابلیت استفاده مجدد کلاس‌های Modelica است.

تفکر در مورد مدلسازی به وسیله معادلات برای بیشتر برنامه‌نویس‌ها کمی غیر معمول است.

در زبان Modelica موارد زیر صادق است:

- دستورات جایگزین در زبان‌های سنتی در Modelica معمولاً به صورت معادلات ارائه می‌شوند.
- دستورات جایگزینی برای صفات توسط معادلات ارائه می‌شوند.
- ارتباط بین اشیاء باعث تولید معادلات می‌گردد.

برای آن که بهتر درک کنیم که چرا معادلات قدرتمندتر از دستورات جایگزینی هستند، به یک مثال میزنیم. معادله یک مقاومت را درنظر بگیرید که مقاومت R ضربدر جریان i برابر با ولتاژ v است:

```
R*i=v
```

اگر بخواهیم از دستورات جایگزین استفاده کنیم، با توجه به این که مقدار کدام متغیرها مشخص است، این معادله می‌تواند به سه شکل استفاده شود: محاسبه جریان از ولتاژ و مقاومت، محاسبه ولتاژ از مقاومت و جریان، یا محاسبه مقاومت از جریان و ولتاژ. این سه دستور به شکل دستورات جایگزینی زیر است:

```
i := v/R;
v := R*i;
R := v/i;
```

در Modelica، معادلات را می‌توان به سه گروه مختلف دسته‌بندی کرد، این دسته‌بندی از تقسیمات رسمی این زبان نیست و ما برای ایجاد درک بهتر به معادلات به این شکل نگاه می‌کنیم، معیار این دسته‌بندی ساختار دستوری معادلات است:

- معادلات معمولی که در قسمت معادله نوشته می‌شوند، دستورات اتصال نیز شکل خاصی از این معادلات هستند.
- معادلات تعریف، که قسمتی از تعریف متغیرها یا ثابت‌ها هستند.
- معادلات اصلاحگر که معمولا برای اصلاح مشخصات استفاده می‌شوند.

در خصوص تعیین صفات پارامترها قبلاً مثال‌هایی برای چگونه تغییر دادن مقدار اولیه را ارائه دادیم:

```
Real y(start=1);
```

معادلات تعریف معمولا به عنوان قسمتی از تعریف پارامترها یا ثابت‌ها نوشته می‌شوند، برای مثال:

```
constant Integer one = 1;
parameter Real mass = 22.5;
```

معادلات تعریف همیشه صادق هستند، مثلاً جرم در مثال بالا در زمان شبیه‌سازی هیچ وقت تغییر نمی‌کند. برای متغیرهای حالت هم می‌توان از معادلات تعریف استفاده کرد و به آنها مقداری اختصاص داد.

```
Real speed = 72.4;
```

با این حال معقول نیست که این کار را بکنیم! چون باعث می‌گردد در حین محاسبات، متغیر را مجبور به داشتن مقدار مشخصی‌کرده و متغیر نقش یک ثابت را بازی خواهد کرد. اگر لازم است برای متغیری مقدار اولیه (مقدار متغیر در شروع محاسبات) مشخص گردد، باید از معادله اصلاحگر برای تعیین مقدار اولیه که یکی از مشخصات متغیر است، استفاده نمایید.

```
Real speed (start = 72.4);
```

۱۴-۹-۱) ساختار معادلات تکراری

بعضی وقت‌ها لازم است مجموعه‌ای از معادلات مشابه را تعریف نماییم. اغلب می‌توان آنها رابه صورت آرایه‌ای از معادلات بیان کرد. معمولاًبرای دسترسی به اعضاء آرایه از براکت استفاه می‌کنیم. زبان Modelica برای معادلات مشابه ساختار حلقه را معرفی کرده است. توجه داشته باشید که این حلقه مانند حلقه‌های موجود در الگوریتم‌های برنامه‌نویسی نیست. این حلقه نمادی از دسته معادلات مشابه است که با این کار تولید و نوشتن آنها بسیار آسان‌تر شده است.

برای مثال عبارت چند جمله‌ای زیر را در نظر بگیرید:

```
y = a[1]+a[2]*x + a[2]*x^1 + ... + a[n]*x^n
```

عبارت چند جمله‌ای را می‌توان به عنوان مجموعه‌ای از معادلات با ساختار معمولی در Modelica اعلان کرد، y برابر است با حاصل ضرب اسکالر بردارهای a و xpowers:

```
xpowers[1] = 1;
xpowers[2] = xpowers[1]*x;
xpowers[3] = xpowers[2]*x;
...
xpowers[n+1] = xpowers[n]*x;
y = a * xpowers;
```

مجموعه منظم معادلات شامل xpowers را می‌توان با استفاده از نماد حلقه‌راحت‌تر بیان کرد:

```
for i in 1:n loop
	xpowers[i+1] = xpowers[i]*x;
end for;
```

در این حالت خاص معادلات برداری،تعریف فشرده‌تری ایجاد می‌کند:

xpowers[2:n+1] = xpowers[1:n]*x;

۱۴-۱۰) مدلسازی فیزیکی غیرسببی[1]

قبل از ادامه لازم می‌دانم کلمه سببی را تعبیر نمایم، این کلمه یعنی داشتن رابطه علت و معلول مشخص، مثلاً وقتی یک تابع ریاضی را به صورت y=f(x) بیان می‌کنیم، x را به عنوان ورودی در نظر می‌گیریم و با داشتن x، y را محاسبه می‌کنیم. اگر x را متغیر مستقل بگیریم و y را متغیر وابسته آنگاه x علت و y معلول آن است. در مقابل غیرسببی قرار دارد که روابط علت و معلولی مشخصی بین متغیرها قابل تشخیص نیست، مثلاً فرض کنید رابطه‌ای به شکل $v^2 + u^2 = 1$ باشد که نمی‌توان متغیر خاصی را به عنوان متغیر مستقل در نظر گرفت، در مدلسازی سیستم‌های فیزیکی نوشتن معادلات به صورت غیرسببی مدلسازی را بسیار آسانتر و بسیار قویتر می‌کند.

در مدلسازی غیرسببی به جای دستورات جایگزینی (منظور دستوراتی است که مقدار یک عبارت را محاسبه نموده و آنرا به یک متغیر اختصاص می‌دهند)، مدلسازی براساس معادلات بنا شده است. معادلات مشخص نمی‌کنند کدام متغیرها ورودی‌اند و کدام خروجی (نتایج). در حالی که در دستورات جایگزین متغیرها در سمت چپ همیشه خروجی هستند و متغیرها در سمت راست همیشه ورودی هستند. بنابراین در مدلسازی غیرسببی روابط سببی بین معادلات مشخص نیست و بعد از حل معادلات مشخص خواهد شد. ذات مدلسازی فیزیکی نشان می‌دهد که مدلسازی غیرسببی برای ارائه ساختار سیستم‌های فیزیکی بسیار مناسب تر است. مثلاً یک شیر صنعتی را در نظر بگیرید. جریان ورودی و خروجی آن را نمی‌توان فقط با دیدن شیر در یک سیستم صنعتی مشخص نمود، چون ممکن است با توجه به فشار جریان برعکس تصور ما وجود داشته باشد.

مزیت اصلی مدلسازی غیرسببی در سازگاری بین جهت حل معادلات باجریان اطلاعات در سیستم تحت مدلسازی است. مفهوم جریان اطلاعات به وسیله تعیین متغیرهای ورودی و خروجی در سیستم مشخص می‌گردد. کتابخانه Modelica مجموعه گسترده‌ای از قطعات غیر سببی است. غیرسببی بودن کتابخانه کلاس‌های Modelica باعث می‌شود قابلیت استفاده مجدد آنها به نسبت انواع کلاس‌هایی که با دستورات جایگزینی نوشته می‌شوند و ورودی و خروجی آنها مشخص است (روابط سببی مشخص)، بسیار بالاتر برود.

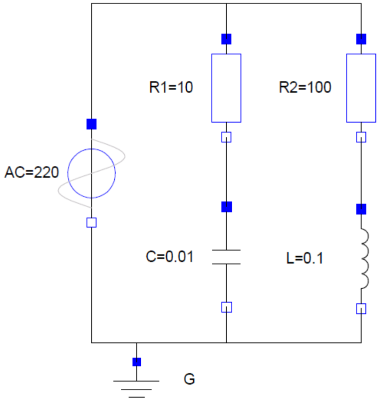
۱۴-۱۰-۱) مدلسازی فیزیکی در مقابل مدلسازی سببی

برای روشن ساختن ایده مدلسازی فیزیکی غیرسببی، مثالی از یک مدار ساده الکتریکی ارائه می‌گردد، شکل ۱۴-۳ **Error! Reference source not found.**را ببینید. دیاگرام اتصال مدار الکتریکی نشان می‌دهد که قطعات چگونه به هم وصل شده‌اند و تقریبا متناظر با آرایش فیزیکی مدار

---

[1] Acausal physical modeling

الکتریکی بر روی یک بردچاپی است. اتصالات فیزیکی در مدار واقعی مشابه اتصالات در دیاگرام است. بنابراین عبارت مدلسازی فیزیکی کاملاً مناسب است.

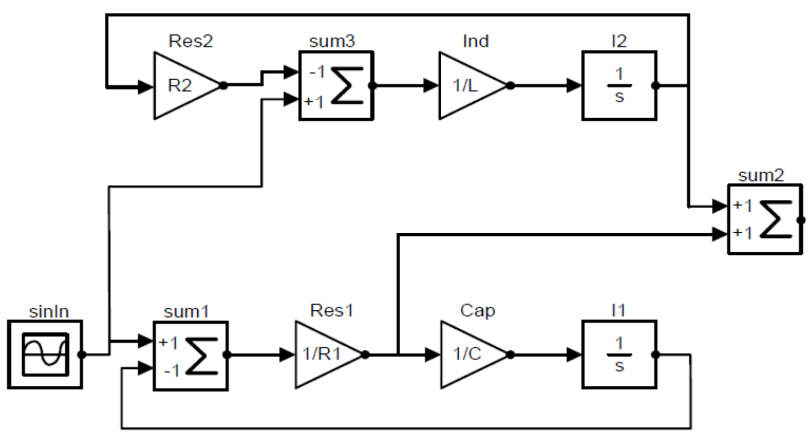


شکل ۳-۱۴. دیاگرام اتصالات مدل غیرسببی مدار ساده الکتریکی.

مدل زیر به زبان Modelica و متناظر با مدار نشان داده شده در شکل ۳-۱۴ می‌باشد. هر شئ گرافیکی در دیاگرام متناظر با نمونه اعلان شده در مدل مدار است. از آنجایی که در این مدل جریان سیگنال (اثر علت و معلولی) مشخص نشده است، این مدل غیرسببی است. اتصال بین قطعات توسط عبارت connect انجام شده است، این عبارت شکل دستوری خاصی از معادلات است که بعداً در مورد آن بیشتر خواهیم گفت.

```
model SimpleCircuit
	Resistor R1(R = 10);
	Capacitor C(C = 0.01);
	Resistor R2(R = 100);
	Inductor L(L = 0.1);
	VsourceAC AC;
	Ground G;
equation
	connect(AC.p, R1.p); // Capacitor circuit
	connect(R1.n, C.p);
	connect(C.n, AC.n);
	connect(R1.p, R2.p); // Inductor circuit
	connect(R2.n, L.p);
	connect(L.n, C.n);
	connect(AC.n, G.p); // Ground
end SimpleCircuit;
```

برای مقایسه، همین مدار را با استفاده از مدلسازی سببی به صورت بلوکی در شکل ۱۴-۴ نشان داده‌ایم. پیکربندی فیزیکی در اینجا قابل تشخیص نیست (ساختار دیاگرام با ساختار فیزیک برد مدار به سادگی قابل انطباق نیست). از آنجایی که جریان سیگنال مشخص و در دیاگرام نمایش داده شده، این مدل سببی است. حتی برای این مثال ساده، تحلیل و عملیات لازم برای تبدیل مدل فیزیکی به مدل سببی بلوکی برای غیرمتخصصین کار دشواری است. یکی دیگر از معایب این روش در نمایش مقاومتهای مدار است که وابسته به شرایط قرارگیری آنهاست و ممکن است بلوکهای متفاوتی برای آنها به کار برود. برای مثال مقاومت R1 و R2 تعاریف متفاوتی دارند. که استفاده مجدد از کتابخانه مدل را سخت می‌کند. علاوه بر این، نگهداری مدل این گونه سیستم‌ها معمولاًبسیار دشوار است، زیرا اعمال تغییرات کوچک در ساختار فیزیکی مدار ممکن است باعث تغییرات متناظر بزرگی در مدل بلوکی سیستم شود.



شکل ۴-۱۴.مدل مدار ساده الکتریکی با استفاده از مدلسازی سببی بلوکی.

۱۴-۱۱) قابلیت مدلسازی جزء به جزء

برای مدت طولانی نرم‌افزار نویسان به سازندگان صنعت سخت‌افزار غبطه می‌خوردند. در سخت-افزار استفاده مجدد از تجهیزات بسیار ساده است، می‌توان این تجهیزات را به راحتی برای ساختن سیستمهای پیچیده کنار یکدیگر قرارداد. اما به نظر می‌رسید در نرم‌افزار برای ایجاد نرم افزارهای کمی متفاوت‌تر به جای استفاده مجدد از کدها، دوباره باید از صفر شروع کرد. تلاش‌های اولیه برای رفع این مشکل استفاده از کتابخانه رویه‌های بود، که متأسفانه دارای محدودیت قابلیت اجرا و انعطاف-پذیری کم می‌باشد. ظهور برنامه‌نویسی شیءگرا باعث ایجاد چارچوبهایی شامل کتابخانه اجزا مثل CORBA, theMicrosoftCOM/DCOM component objectmodel و JavaBeans گردید. این چارچوبها در حیطه‌های خاصی موفقیت قابل توجهی به دست آوردند اما هنوز راه طولانی برای

دستیابی به سطح استفاده مجدد و استانداردسازی اجزا که در صنعت سخت افزار یافت می‌شود، در پیش روست.

خواننده ممکن است تعجب کند تمام این مطالب چه ارتباطی با Modelica دارد. در واقع Modelica مدل نرم‌افزاری قدرتمندی از اجزا را ارائه می‌کند که از لحاظ انعطاف پذیری برابری با مولفه‌های سیستم سخت‌افزاری دارند و پتانسیل قابلیت استفاده مجدد را داراست. کلید افزایش انعطاف پذیری در کلاس‌های Modelica نهفته است، چون این کلاسها بر پایه معادلات ایجاد می-گردند.

مدل نرم‌افزاری اجزا چیست؟ مفهومی است که Modelica ارائه کرده است و شما در هر مدلسازی از این سیستم استفاده خواهید نمود. به خصوص زمانی که لازم باشد کتابخانه یا قطعات جدیدی بسازید، از این مفاهیم استفاده خواهید نمود و باید شامل سه مورد زیرباشد:

- اجزاء
- مکانیزم اتصال
- چارچوب اجزاء

اجزا از طریق مکانیزم اتصال به هم متصل می‌شوند، که در دیاگرام اتصالات قابل مشاهده هستند. چارچوب اجزا، مولفه‌ها و اتصالات را درک می‌کند و این اطمینان را می‌دهد که ارتباط تمام اتصالات برقرار است. برای سیستم‌های تشکیل شده از مولفه غیرسببی، جهت جریان اطلاعات یعنی سببیت، به طور خودکار توسط مترجم در زمان ترکیب استخراج می‌گردد.

۱۴-۱۱-۱) اجزاء

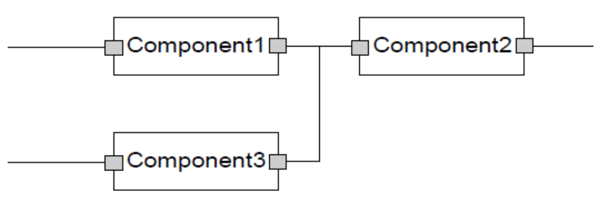
اجزاء فقط نمونه‌هایی از کلاس‌های Modelica هستند. این کلاس‌ها باید دارای ارتباط تعریف شده خوبی با دنیای خارج باشند، این ارتباط از طریق درگاه‌ها میسر می‌گردد که در بخش‌های قبل با آنها آشنا شدیم. درگاه یا کانکتور در Modelica برای ارتباط بین اجزاء و دنیای بیرون به کار خواهند رفت.

یک جزء به صورت مستقل از محیطی که در آن استفاده میشود، مدل خواهد شد. رعایت این نکته برای استفاده مجدد ضروری است. یعنی هر جزء در زمان تعریفش فقط باید شامل معادلات، متغیرهای محلی و متغیرهای ارتباطی باشد. وجود هیچ وسیله ارتباطی بین جزء و بقیه سیستم به غیر از درگاه‌های تعریف شده در جزء، مجاز نیست. یک جزء ممکن است خودش از ترکیبی از اجزاء دیگر که به یکدیگر متصل شده‌اند تشکیل گردد، یعنی مدل سازی سلسله مراتبی.

در هنگام خلق اجزاء جدید بعد از ساختن مدل، مناسب است فرض کنید کاربر قصد دارد مدل شما را دچار مشکل نماید و با این دیدگاه مدل را آزمایش نمایید و مدل را به گونه‌ای تعریف نمایید که کاربر این مدل به هیچ وجه نتواند مدل شما را دچار مشکل سازد.

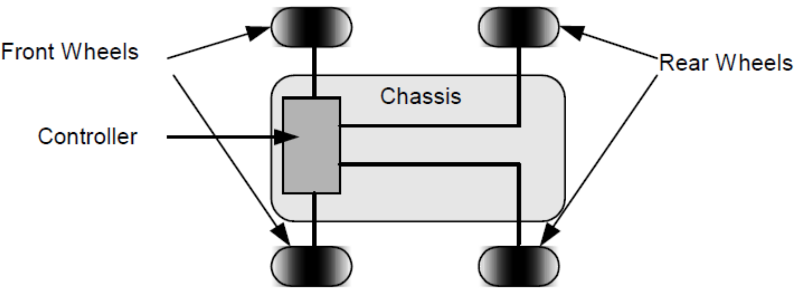
۱۴-۱۱-۲) دیاگرام اتصالات

سیستم‌های پیچیده معمولاً شامل تعداد زیادی از قطعات به هم متصل شده می‌باشند که خیلی از این قطعات می‌توانند به صورت سلسله مراتبی خودشان از قطعات دیگری تشکیل شده باشند. برای درک این پیچیده‌گی نمایش تصویری اجزاء و اتصالات بسیار مهم است. این چنین نمایش دیداری، دیاگرام اتصالات نامیده می‌شود، یک مثال شماتیک در شکل ۵-۱۴ نشان داده شده است. قبلاًهم دیاگرام اتصال رابرای یک مدار ساده نشان داده‌ایم (شکل ۳-۱۴).



شکل ۵-۱۴.نمایش شماتیک اجزاء و اتصالات.

هر مستطیل در نمودار نماینده یک قطعه فیزیکی است برای مثال یک مقاومت، خازن، ترانزیستور، چرخ دنده مکانیکی، شیر و ....اتصالات بین مستطیل‌ها با اتصالات فیزیکی واقعی بین قطعات متناظر است و در نمودار با خط نشان داده می‌شود. برای مثال اتصالات را می‌توان معادل سیم‌های الکتریکی، اتصالات مکانیکی، لوله‌ها برای سیالات، اتصالات لازم برای انتقال حرارت بین مولفه‌ها در نظر گرفت. درگاه‌ها یعنی نقطه اتصال هر جزء با محیط، به صورت یک مربع کوچک در نمودار نشان داده شده است. متغیرها در این درگاه‌ها تعامل بین اجزاء را تعریف می‌کنند.



شکل ۶-۱۴.دیاگرام اتصال برای مدل ساده یک اتومبیل.

به عنوان کاربرد نمودار اتصال در مکانیک، یک مثال ساده از اتومبیل در شکل ۶-۱۴ نشان داده شده است.

مدل ساده اتومبیل شامل متغیرهایی برای اجزاء زیرمجموعه‌اش مثل چرخ‌ها، شاسی، واحد کنترل است. چرخ‌ها هم به شاسی و هم به کنترل کننده متصل شده‌اند. عبارت‌هایی برای اتصالات تعریف می‌گردد، ولی در مثال خلاصه زیر نمایش داده نشده است.

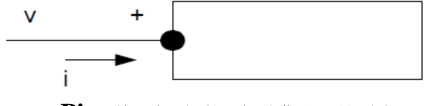
```
class Car "A car class to combine car components"
        Wheel w1,w2,w3,w4 "Four wheels;
        Chassis chassis "Chassis;
        CarController controller "Car controller;
        ...
end Car;
```

۱۴-۱۱-۳) درگاه ها و کلاس‌های اتصال دهنده

درگاه‌های Modelica نمونه‌ای از کلاس‌های اتصالات هستند، این درگاه‌ها متغیرهای مربوط به خود را دارند و برای ارتباط بین مدل و محیط به کار می‌روند. بنابراین درگاه‌ها ارتباط خارجی را برای مدلها ایجاد می‌کنند.

برای مثال، Pin یک درگاه است و به عنوان یک کلاس اتصال دهنده (connector) تعریف شده است. این درگاه می‌تواند برای ارتباط خارجی مولفه‌های الکتریکی استفاده شود. متغیرهای Voltage و Current که درکلاس Pin استفاده می‌شوند، هر دو از جنس Real هستند. ولی چون واحدهای متفاوتی دارند به صورت انواع جدید تعریف شده‌اند. از دید زبان Modelica متغیرهایVoltage وCurrent دقیقاً Real هستند. ارزیابی کردن سازگاری واحدها در معادلات اختیاری است و توسط ابزار ویژه این کار انجام می‌پذیرد.

```
type Voltage = Real(Unit="V");
type Current = Real(Unit="A");
```



شکل ۷-۱۴. قطعه‌ای با یک اتصال Pin

```
connector Pin
        Voltage         v;
        flow Current    i;
end Pin;
```

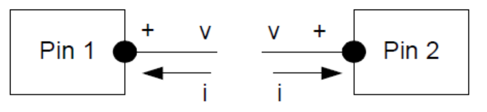
۱۴-۱۱-۴) اتصالات

اتصال بین قطعات می‌تواند بین درگاه‌ها با درگاه‌های هم ارز آنها برقرار شود. زبان Modelica از اتصالات غیرسببی بر پایه معادلات پشتیبانی می‌کند، یعنی اتصالات به عنوان معادلات شناخته می‌شوند.در اتصالات غیرسببی (اتصالات معمول modelica)، به معلوم بودن جهت جریان اطلاعات نیازی نیست. زبان modelca از اتصالات سببی نیز پشتیبانی می‌نماید. اتصالات سببی را می‌توان با وصل کردن یک درگاه ورودی به یک درگاه خروجی انجام داد.

متغیر اتصال می‌تواند به صورت جریانی یا عادل تعریف گردد. متغیرهای جریانی با استفاده از پیشوند flow تعریف می‌گردند. با توجه به این که متغیرهای اتصال از نوع عادی (پیش فرض) یا از نوع جریانی تعریف باشند، دو نوع جفت شدن[1]اتصالات قابل تشخیص است:

- جفت شدن برابر، برای متغیرهای عادی.
- جفت شدن با جمع برابر صفر، برای متغیرهای جریانی که از قانون کیرشهف تبعیت می‌کند.

برای مثال کلمه کلیدی flow برای متغیر i از نوع Current در کلاس اتصال‌دهنده Pin، نشان می‌دهد که بر طبق قانون کیرشهف، جمع تمامی جریانهای متصل به Pin، برابر صفر است.



شکل ۸-۱۴. اتصال دو قطعه که پینهای الکتریکی دارند.

برای وصل کردن دو مدل ازعبارات اتصال استفاده می‌شود. یک عبارت اتصال مثل connect(Pin1,Pin2)، دو درگاه Pin1 و Pin2 را به هم وصل می‌کند و یک گره را تشکیل می‌دهند. این کار باعث ایجاد دو معادله می‌گردد:

$$Pin1.v = Pin2.v$$
$$Pin1.i + Pin2.i = 0$$

معادله اول می‌گوید که ولتاژ انتهای اتصالات برابر هستند. معادله دوم متناظر با قانون کیرشهف است که می‌گوید، مجموع جریان‌های ورودی و خروجی به یک گره برابر صفر است (علامت جریان ورودی به قطعه را مثبت فرض کنید). معادلات جمع برابر صفر وقتی تولید می‌شوند که از پیشوند flow استفاده شود. قانون مشابهی برای جریان در شبکه‌های لوله و نیرو و گشتاوردر سیستم‌های مکانیکی به کار می‌رود.

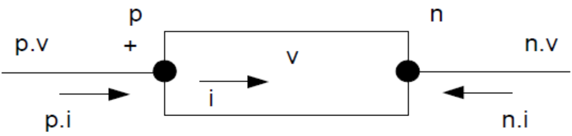
[1] coupling

۱۴-۱۲) کلاس‌های جزیی مشخصات عمومی را بیان می‌کنند

بسیاری از قطعات الکتریکی دارای ۲ پین (اتصال) هستند، این مورد را به عنوان یک خاصیت مشترک در نظر گرفته و یک کلاس الگو برای ساخت همه قطعات الکتریکی تعریف می‌کنیم. کلاس الگویی که این ویژگی مشترک را داراست، TwoPin می‌نامیم. از آنجایی که این کلاس شامل معادلات کافی برای تعریف رفتار فیزیکی قطعه نیست، یک کلاس جزیی بوده و آن را با پیشوند partial تعریف می‌کنیم. کلاسهای جزیی معمولاً به عنوان کلاس‌های خلاصه[1] در دیگر زبان‌های شیءگرا شناخته می‌شوند.

```
partial class TwoPin "Superclass of elements with two electrical pins"
        Pin p, n;
        Voltage v;
        Current i;
equation
        v = p.v - n.v;
        p.i + n.i = 0;
        i = p.i;
end TwoPin;
```

کلاس TwoPin دارای دو پین به نامهای p و n است و همچنین یک کمیت v، که افت ولتاژ در قطعه را نشان می‌دهد و کمیت i که جریان خروجی از پین n و ورودی به پین p به داخل قطعه را تعریف می‌کند. شکل ۹-۱۴را ببینید.



شکل ۹-۱۴. کلاس عمومی TwoPin که ساختار کلی قطعات ساده الکتریکی با دو اتصال را تعریف می‌کند.

معادلات روابط کلی بین کمیت‌های یک قطعه الکتریکی ساده را تعریف می‌کنند. به منظور استفاده از این قطعه باید معادلاتی که رفتار فیزیکی خاص قطعه را نشان می‌دهد به این مدل اضافه گردد.

---

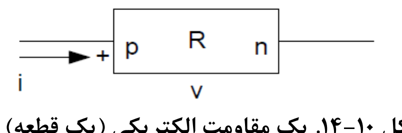
[1] abstract

۱۴-۱۲-۱) استفاده مجدد از کلاس‌های جزیی

کلاس‌جزئیTwoPinرا به صورت کلی تعریف نمودیم. می‌توان با استفاده از این کلاس قطعات الکتریکی مختلف را ساخت. از مدلسازی مقاومت الکتریکی شروع می‌کنیم. برای ایجاد مدل مقاومت به این کلاس جزئی، معادلات سازنده مقاومت را اضافه می‌کنیم.

```
R*i = v;
```

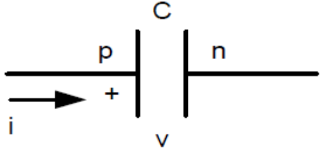
این معادله رفتار فیزیکی (مشخصه فیزیکی) یک مقاومت الکتریکی را با رابطه بین ولتاژ و جریان نشان می‌دهد.



شکل ۱۰-۱۴. یک مقاومت الکتریکی (یک قطعه)

```
class Resistor "Ideal electrical resistor"
        extends TwoPin;
        parameter Real R(Unit = "Ohm") "Resistance";
equation
        R*i = v;
end Resistor;
```

به روش مشابهی می‌توان با استفاده مجدد از کلاس جزئی TwoPin و با اضافه کردن معادلات فیزیکی حکم بر یک خازن، یک کلاس خازن الکتریکی ساخت.



شکل ۱۱-۱۴. یک خازن الکتریکی.

```
class Capacitor "Ideal electrical capacitor"
        extends TwoPin;
        parameter Real C(Unit="F") "Capacitance";
equation
        C*der(v) = i;
end Capacitor;
```

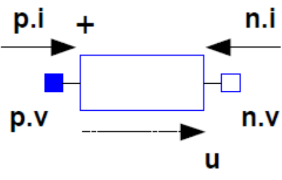
در طول مدت شبیه‌سازی متغیرهای i و v که در قطعات بالا مشخص شده‌اند، به عنوان تابعی از زمان در نظر گرفته خواهند شد. حل‌کننده معادلات دیفرانسیلی مقادیر i(t) و v(t) را محاسبه می‌کند (t زمان است) به طوریکه برای تمام مقادیر t معادله تشکیل دهنده خازن را ارضاء نماید.

۱۴-۱۳) کتابخانه قطعات الکتریکی

کلاس‌های دیگر قطعات الکتریکی را مشابه روش ایجادمقاومت و خازن، ساخته و یک کتابخانه ساده قطعات الکتریکی تشکیل می‌دهیم. اینچنین کتابخانه‌ای از قطعات با قابلیت استفاده مجدد، کلید مدلسازی موثر سیستم‌های پیچیده است. در ادامه کتابخانه کوچکی از قطعات الکتریکی به همراه معادلاتشان نشان داده شده است، این قطعات را برای مدل کردن مدار الکتریکی ساده‌ای استفاده خواهیم کرد. برای درک بهتر روش شبیه‌سازی و حل معادلات توسط محیط modelica، معادلات قطعات را استخراج نموده و خودمان آنها را بررسی می‌نماییم.

۱۴-۱۳-۱) مقاومت

از مدل مقاومت که در شکل ۱۲-۱۴ نشان داده شده است، چهار معادله قابل استخراج است.

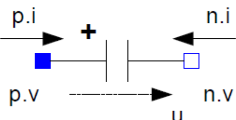


شکل ۱۲-۱۴.مقاومت الکتریکی.

سه معادله اول از کلاس TwoPin و معادله آخر معادله تشکیل دهنده مقاومت است.

$0 = p.i + n.i$
$v = p.v - n.v$
$i = p.i$
$v = R*i$

۱۴-۱۳-۲) خازن

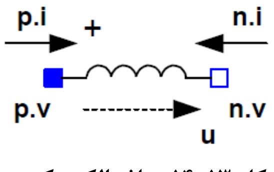


شکل ۱-۱۴. خازن الکتریکی.

چهار معادله زیر از مدل خازن به دست آمده‌اند، که معادله آخر معادله سازنده خازن است.

```
0 = p.i + n.i
v = p.v - n.v
i = p.i
i = C*der(v)
```

۱۴-۱۳-۳) القاگر(سلف)



شکل ۱۳-۱۴. سلف الکتریکی.

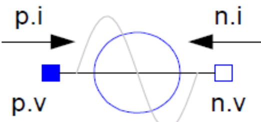
کلاس نمایش داده شده زیر یک مدل ایده‌آل سلف را نشان می‌دهد.

```
class Inductor "Ideal electrical inductor"
        extends TwoPin;
        parameter Real L(Unit = "H") "Inductance";
equation
        v = L*der(i);
end Inductor;
```

از کلاس سلف می‌توان چهار معادله استخراج کردکه طبق معمول سه تای اول از کلاس TwoPin نشأت می‌گیرد و آخرین معادله معادله سازنده سلف است.

```
0 = p.i + n.i
v = p.v - n.v
i = p.i
v = L*der(i)
```

۱۴-۱۳-۴) منبع ولتاژ



شکل ۱۴-۱۴. منبع ولتاژ الکتریکی VsourceAC که در آن u(t)=VA*sin(2*Pi*f*time).

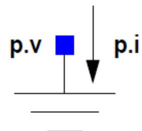
یک مدل برای منبع ولتاژی با موج سینوسی خواهیم ساخت. این منبع برای ساخت مدار الکتریکی استفاده خواهد شد. آن را کلاس VsourceAC می‌نامیم. این کلاس به شکل زیر خواهد بود. این مدل نیز مانند سایر مدل‌های Modelica به عنوان تابعی از زمان رفتار می‌کند. توجه داشته باشید متغیر زمان (از پیش تعریف شده) در طول مدت شبیه‌سازی رو به جلو پیش می‌رود.

```
class VsourceAC "Sin-wave voltage source"
        extends TwoPin;
        parameter Voltage VA = 220 "Amplitude";
        parameter Real f(unit = "Hz") = 50 "Frequency";
        constant Real PI = 3.141592653589793;
        input Voltage u;
equation
        v = u;
        u = VA*sin(2*PI*f*time);
end VsourceAC;
```

در این مدل دو پایه‌ای که بر اساس کلاس جزئی TwoPin ساخته شده است، ۵ معادله قابل استخراج است.

```
0 = p.i + n.i
v = p.v - n.v
i = p.i
v=u
u = VA*sin(2*PI*f*time)
```

۱۴-۱۳-۵) زمین



شکل ۱۵-۱۴. زمین مدار.

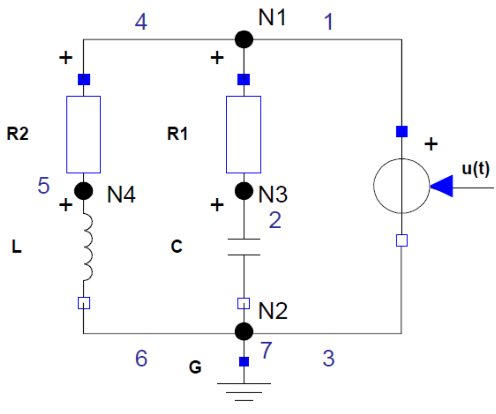
در آخر ما یک کلاس برای زمین تعریف می‌کنیم که می‌تواند به عنوان مرجعی برای سطح ولتاژ در مدار الکتریکی معرفی گردد. این کلاس فقط یک اتصال دارد.

```
class Ground "Ground"
	Pin p;
equation
	p.v = 0;
end Ground;
```

تنها یک معادله را می‌توان از کلاس Ground استخراج نمود.

```
p.v = 0
```

۱۴-۱۴) مدل مدار ساده



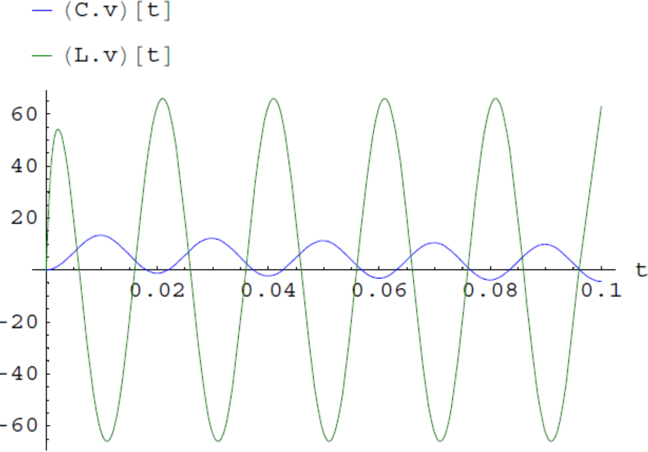
شکل ۱۶-۱۴. مدل مدار ساده الکتریکی.

پس از گردآوری یک کتابخانه کوچک از قطعات الکتریکی ساده می‌توانیم مدار الکتریکی ساده‌ای که در شکل ۱۶-۱۴ نشان داده شده است راسر هم کنیم. دو مقاومت R1 و R2 به همراه معادلات اصلاحی برای اصلاح مقدار مقاومت هر کدامشان تعریف شده است.به همین ترتیب یک خازن C و یک سلف L همراه با معادلات اصلاحیشان برای اصلاح ظرفیت خازن و خودالقایی سلف تعریف شده‌اند. منبع ولتاژ AC و زمین G هیچ گونه معادله اصلاحی ندارند. عبارات اتصال، ارتباطات لازم بین قطعات را فراهم می‌کنند.

```
class SimpleCircuit
	Resistor R1(R = 10);
	Capacitor C(C = 0.01);
	Resistor R2(R = 100);
	Inductor L(L = 0.1);
	VsourceAC AC;
```

```
        Ground G;
equation
        connect(AC.p, R1.p); // 1, Capacitor circuit
        connect(R1.n, C.p); // Wire 2
        connect(C.n, AC.n); // Wire 3
        connect(R1.p, R2.p); // 2, Inductor circuit
        connect(R2.n, L.p); // Wire 5
        connect(L.n, C.n); // Wire 6
        connect(AC.n, G.p); // 7, Ground
end SimpleCircuit;
```

در ادامه برای این مدل (مدار)، شبیه‌سازی انجام می‌شود و بعد نمودار ولتاژ خازن (آبی) و سلف (سبز) رسم می‌گردد.



شکل ۱۷-۱۴. نمودار ولتاژ خازن و سلف.

۱۴-۱۵) آرایه‌ها

آرایه مجموعه‌ای از متغیرها از نوع مشابه است. عناصر یک آرایه از طریق ایندیس‌های آن قابل دسترس هستند. اندیسهای آرایه همیشه اعداد صحیح است. محدوه یک آرایه از کران پایین، از ۱ شروع می‌گردد و تا کران بالا که به اندازه و ابعاد آرایه مربوط است، ادامه می‌یابد. متغیر آرایه را می‌تواند به وسیله نوشتن ابعاد در براکت بعد از اسم کلاس یا اسم متغیر، تعریف نمود. برای مثال:

```
Real[3] positionvector = {1,2,3};
Real[3,3]        identitymatrix = {{1,0,0}, {0,1,0}, {0,0,1}};
Real[3,3,3] arr3d;
```

تعریف بالا،یک بردار موقعیت سه بعدی (یک آرایه تک بعدی با سه عضو)، یک ماتریس تبدیل (یک آرایه دو بعدی با ۹ عضو) و یک آرایه سه بعدی را مشخص می‌کند. با استفاده از دستورالعمل زیر نیز می‌توان با مشخص نمودن ابعاد بعد از اسم متغیر، به همان نتیجه بالا رسید:

```
Real positionvector[3] = {1,2,3};
Real identitymatrix[3,3] = {{1,0,0}, {0,1,0}, {0,0,1}};
Real arr3d[3,3,3];
```

در تعریف دو آرایه اول، از سازنده آرایه {} برای مقداردهی به آرایه‌ها و مشخص نمودن بردار موقعیت و ماتریس همانی استفاده شده است. اعضاء آرایه A بانوشتن A[i,j,...] قابل دسترسی هستند. کران پایین اندیسهای آرایه ۱ و کران بالا برای k امین بعد (A,k) است. زیرماتریس‌ها را می‌توان با به کار بردن علامت : در محدوده اندیسها مشخص نمود، برای مثال A[i1:i2,j1:j2]. عبارات شامل آرایه‌ها را می‌توان با استفاده از عملگرهای محاسباتی +، - ، * و / ایجاد نمود. این عملگرها هم بر روی اسکالرها و هم بر روی بردارها، ماتریس‌ها یا آرایه‌های چند بعدی (وقتی که از نظر ریاضی امکانپذیر باشد) با عناصری از نوع Real و Integer قابل استفاده است. عملگر * وقتی بین بردارها استفاده می‌شود، ضرب اسکالر، وقتی بین ماتریسها استفاده شود، ضرب ماتریسی و زمانی که بین یک اسکالر و یک آرایه استفاده گردد به عنوان ضرب المان به المان عمل خواهد کرد. به عنوان یک مثال ضرب positionvector در ۲:

```
positionvector*2
```

که حاصل آن:

```
{2,4,6}
```

تعدادی از توابع از پیش تعریف شده در Modelica در دسترس است، که تعداد کمی از آنها در جدول ۱-۱۴ نشان داده شده است.

جدول ۱-۱۴. توابع از پیش تعریف شده در Modelica.

transpose(A)	دو بعد اول آرایه A را جابه‌جا می‌کند (ترانزپوز)
zero(n1, n2, n3, …)	آرایه ای پر از صفر را باز می‌گرداند (آرایه صحیح)
ones(n1, n2, n3, …)	آرایه ای پر از یک را باز می‌گرداند (آرایه صحیح)
fill(s, n1, n2, n3, …)	آرایه ای پر از مقدار عددی s را باز می‌گرداند (آرایه صحیح)
min(A)	کوچکترین المان آرایه A را باز می‌گرداند
max(A)	بزرگترین المان آرایه A را باز می‌گرداند

sum(A)	مجموع همه المانهای آرایه A را باز می‌گرداند

یک تابع اسکالر می‌تواند به صورت المان به المان روی آرایه به کار برده شود. برای مثال اگر A برداریاز اعداد حقیقی باشد، cos(A) یک بردار است که هر المانش نتیجه اعمال کردن تابع cos بر روی المان متناظرش در A می‌باشد. برای مثال:

cos({1, 2, 3}) = {cos(1), cos(2), cos(3)}

ترکیب عمومی آرایه‌ها از طریق عملگر الحاقی cat(k,A,B,C,…) که آرایه‌های A,B,C,… رادر امتداد بعد kام به هم پیوند می‌دهد، امکانپذیر است.

حالت خاص الحاق در امتداد اولین و دومین بعد به ترتیب از طریق دستورات خاص [A;B;C;...] و [A,B,C,...] پشتیبانی می‌شود. هر دو دستور را می‌توان با هم ترکیب نمود. به منظور سازگاری با دستورات آرایه Matlab ، استاندارد de facto، آرگومان‌های اسکالر و برداری قبل از الحاق در این عملگرها به ماتریس تبدیل می‌گردند. این کار باعث می‌شود که ایجاد ماتریس از عبارات اسکالر با جدا کردن ردیف‌ها توسط سمیکالن ؛ و ستون‌ها به وسیله کاما ، امکان پذیر گردد. مثال زیر یک ماتریس m در n را ایجاد می‌کند:

```
[expr11, expr12, ... expr1n;
expr21, expr22, ... expr2n;
 …
exprm1, exprm2, ... exprmn]
```

ایجاد یک ماتریس از عبارات اسکالر با استفاده از عملگرهای بالا می‌تواند آموزنده باشد. برای مثال:

[1,2;3,4]

ابتدا هر آرگومان اسکالر به ماتریس تبدیل می‌گردد (ارتقاء می‌یابد):

```
[{{1}}, {{2}};
 {{3}}, {{4}}]
```

از آنجایی که اولویت عملگر [... , ...]که در طول بعد دوم ماتریسها را به هم ملحق می‌کند، بالاتر از اولویت عملگر [...;...] است که در طول بعد اول ماتریسها را ملحق می‌کند، پس از اولین قدم الحاق، نتیجه به شکل زیر خواهد بود:

```
[{{1,2}};
{{3,4}}]
```

در آخر، ردیف‌های ماتریس به هم می‌پیوندند و ماتریس مطلوب ۲در۲ را به ما می‌دهد:

```
{{1, 2},
```

{3, 4}}

حالت خاص یک آرگومان اسکالر را می‌توان جهت تولید ماتریس ۱در ۱ به کار برد:

[1]

که ماتریس زیر را می‌دهد:

{{1}}

۱۴-۱۶) ساختار الگوریتمی

اگرچه معادلات برای مدلسازی سیستم‌های فیزیکی توانایی زیادی دارند، برای برخی دیگر از کارها، مواقعی پیش‌می‌آید که ساختار الگوریتم (مانند الگوریتم‌هایی که در برنامه‌نویسی سنتی به کار می‌روند و محاسبات در آنها به صورت ترتیبی و با جایگزینی عبارتها در متغیرها انجام می‌گردد) نیاز است. مثلاً برای آن محاسبه عددی مقدار یک انتگرال معین در یک لحظه مدلسازی به الگوریتم مرحله به مرحله ترتیبی نیاز داریم.

۱۴-۱۶-۱) الگوریتم‌ها

در Modelica، برای استفاده از الگوریتم، عبارات ترتیبی تنها می‌توانند در بخش‌های که با کلمه کلیدی algorithm شروع می‌شوند، قرار بگیرد. بخش الگوریتم‌ها با ظاهر شدن برخی از کلمات کلیدی دیگر مثل equation, public, protected یا وقتی به عنوان بدنه توابع استفاده می‌شود با کلمه کلیدی end، به پایان می‌رسد.

```
algorithm
        ...
        <statements>
        ...
<some other keyword>
```

یک بخش الگوریتم که در میان بخش معادلات قرار گرفته است، می‌تواند به فرم زیر ظاهر شود، در این مثال بخش الگوریتم شامل سه دستور جایگزینی است. دستورات جایگزینی موجود در الگوریتم با =: مشخص می‌گردد و استفاده از = در الگوریتم مجاز نیست.

```
equation
        x = y*2;
        z=w;
algorithm
        x1 := z+x;
```

```
        x2 := y-5;
        x1 := x2+y;
equation
        u = x1+x2;
        …
```

توجه داشته باشید کدی که در بخش الگوریتم قرار دارد، بعضی وقت‌ها معادلات الگوریتم خوانده می‌شود و از مقادیر متغیرهای مشخصی از خارج الگوریتم استفاده می‌کند. این متغیرها، مانند x، y و z در مثال بالا، متغیر ورودی به الگوریتم نامیده می‌شوند. متغیرهایی که مقدارشان توسط تعریف الگوریتم اختصاص می‌یابد، مانند x1 و x2 در مثال بالا و با رابطه جایگزینی مشخص می‌گردند خروجی الگوریتم خوانده می‌شوند. این شکل نوشتن بخش الگوریتم شبیه به یک تابع با بخش الگوریتم در بدنه‌اش و پارامترهای ورودی و خروجی است.

۱۴-۱۶-۲) دستورات

علاوه بر دستورات جایگزینی، که در مثال قبلی استفاده شد، سه نوع از عبارت‌های الگوریتمی دیگر در Modelica در دسترس هستند: دستوراتif-then-else، حلقه‌های for و حلقه‌های while. استفاده از این دستورات بسیار به کاربرد آنها در برنامه نویسی کلاسیک است. جمع زیر از دستور if و حلقه while استفاده می‌کند، size(a,1) اندازه بعد اول آرایه a را باز می‌گرداند. قسمت elseif و بخش else از دستور if اختیاری است.

```
sum := 0;
n := size(a,1);
while n>0 loop
        if a[n]>0 then
                sum := sum + a[n];
        elseif a[n] > -1 then
                sum := sum - a[n] -1;
        else
                sum := sum - a[n];
        end if;
end while;
```

بار دیگر محاسبات چندجمله‌ای با ساختار منظم را در نظر بگیرید.

```
y := a[1]+a[2]*x + a[2]*x^1 + ... + a[n]*x^n;
```

وقتی از معادلات برای مدل کردن محاسبات یک عبارت چندجمله‌ای استفاده می‌کنید، لازم است که بردار کمکی xpowers رابرای ذخیره کردن توان‌های مختلف x تعریف کرد. همین کار را می‌توان با استفاده حلقه for بدون نیاز به بردار اضافی انجام داد، فقط کافیست از متغیر اسکالر xpower برای ذخیره توان x استفاده کنیم.

```
algorithm
	y := 0;
	xpower := 1;
	for i in 1:n+1 loop
		y := y + a[i]*xpower;
		xpower := xpower*x;
	end for;
	...
```

۱۴-۱۶-۳) توابع

توابع یک بخش طبیعی از هر مدل ریاضی هستند. تعدادی از توابع ریاضی مثل sin، cos، exp و ... در Modelica از پیش تعریف شده‌اند. عملگرهای حسابی +- * / می‌توانند به عنوان یک تابع در نظر گرفته شوند که به راحتی از طریق ساختار یک عملگر قابل استفاده هستند. استفاده از توابع ریاضی تعریف شده توسط کاربر در زبان Modelica امری طبیعی است. بدنه یک تابع Modelica یک بخش الگوریتم است که شامل کد الگوریتمی می‌باشد که زمان فراخوانی تابع به صورت ترتیبی اجرا می‌گردد. پارامترهای ورودی با استفاده از کلمه کلیدی input و نتایج خروجی توسط کلمه کلیدی output مشخص می‌گردد. این کار باعث می‌گردد ساختار دستوری تابع به تعریف کلاس در زبان Modelica نزدیک شود.

توابع Modelica توابع ریاضی هستند. یک تابع Modelica همیشه با داشتن ورودی مشخص، نتیجه مشخصی متناظر با ورودی را باز می‌گرداند. در ادامه کد تابع PolynomialEvaluator برای ارزیابی چند جمله‌ای را آورده‌ایم.

```
function PolynomialEvaluator
	input Real a[:];		// array, size defined at function call time
	input Real x = 1.0;	// default value 1.0 for x
	output Real y;
protected
	Real xpower;
algorithm
	y:=0;
	xpower := 1;
	for i in 1:size(a,1) loop
		y := y + a[i]*xpower;
```

```
            xpower := xpower*x;
        end for;
end PolynomialEvaluator;
```

توابع معمولاً با رعایت مکان نسبی آرگومان حقیقی (آرگومانهای زمان فراخوانی) نسبت به پارامترهای قراردادی (آرگومانهای زمان تعریف) فرا خوانده می‌شوند. برای مثال، در فراخوانی زیر آرگومان حقیقی {۴، ۳، ۲، ۱} به عنوان مقدار بردار a در نظر گرفته می‌شود و ۲۱ به عنوان مقدار پارامتر قراردادی x قرار می‌گیرد. پارامترهای تابع Modelica فقط خواندنی هستند، یعنی ممکن نیست در کد تابع مقادیری به آنها تخصیص داده شود. وقتی تابعی فراخوانده می‌شود مکان نسبی آرگومان و تعداد آرگومان‌های واقعی و پارامترهای قراردادی باید یکسان باشد. نوع آرگومان‌های ارسالی به تابع نیز باید با نوع پارامترهای تابع سازگار باشد.

```
p = PolynomialEvaluator({1, 2, 3, 4}, 21);
```

همان گونه که در مثال بعدی می‌بینید به جای عبارت بالا می‌توان تابع PolynomialEvaluator را با استفاده از نام آرگومان‌های واقعی فراخوانی نمود. فایده این کار، مستندسازی بهتر کد است و باعث می‌گردد کد برای به روز رسانی انعطاف‌پذیری بیشتری داشته باشد. همچنین امکان اشتباه در تخصیص مقدار به پارامتر ها کاهش می‌یابد.

برای مثال اگر فراخوانی تابع PolynomialEvaluator با استفاده از پارامترهای با نام صورت گیرد، ترتیب پارامترهای قراردادی a و x قابل تغییر است و می‌توان پارامترهای قراردادی جدیدی با مقادیر پیش‌فرض در تعریف تابع اضافه نمود بدون اینکه منجر به خطای مترجمی در زمان فراخوانی شود. نیازی به مشخص نمودن مقدار پارامترهای قراردادی که دارای مقادیر پیش‌فرض هستند، نیست، مگر اینکه قرار باشد مقدار متفاوتی به این پارامترها اختصاص داده شود.

```
p = PolynomialEvaluator(a = {1, 2, 3, 4}, x = 21);
```

برعکس زبانهای برنامه‌نویسی سنتی، توابع زبان modelica می‌توانند چندین جواب داشته باشند. برای مثال تابع f که در پایین آمده است دارای سه نتیجه است که به صورت سه پارامتر قراردادی خروجی r1، r2 و r3 تعریف شده است.

```
function f
        input Real x;
        input Real y;
        output Real r1;
        output Real r2;
        output Real r3;
        ...
end f;
```

فراخوانی توابع چند پاسخی در بخش کد الگوریتمیک فقط با استفاده از دستورات جایگزین خاص امکان‌پذیر است، همان طور که در ادامه نشان داده شده، نتایج توابع به متغیرهای سمت چپ اختصاص داده شده‌اند.

```
(a, b, c) := f(1.0, 2.0);
```

دستورالعمل مشابهی در بخش معادلات قابل استفاده است:

```
(a, b, c) = f (1.0, 2.0);
```

۱۴-۱۶-۴) توابع خارجی

فراخوانی توابعی که خارج از زبان Modelica، به زبان C یا Fortran نوشته شده‌اند نیز امکان-پذیر است. اگر زبان تابع خارجی مشخص نشده باشد به طور پیش فرض، زبان C به عنوان زبان تابع در نظر گرفته می‌شود. بدنه تابع خارجی با کلمه کلیدی external در بخش تعریف توابع خارجی در Modelica مشخص می‌گردد. برای مثال تعریف زیر یک تابع خارجی به نام log را مشخص می‌کند که دارای یک پارامتر ورودی و یک پارامتر خروجی است. به جای بدنه تابع از کلمه external استفاده شده است و چون زبانی برای این تابع خارجی مشخص نشده است زبان آن به صورت پیش فرض C در نظر گرفته خواهد شد.

```
function log
	input Real x;
	output Real y;
	external;
end log;
```

رابط توابع خارجی از امکانات پیشرفته‌ای پشتیبانی می‌کند، امکاناتی مانند پارامترهای ورودی-خروجی،آرایه‌های قابل استفاده محلی، ترتیب آرگومانهای تابع خارجی، نقشه حافظه آرایه به صورت سطری و ستونی و... . برای مثال پارامتر قراردادی Ares در کد زیر یک پارامتر ورودی-خروجی در تابع خارجی LeastSquares است که دارای مقدار ورودی پیش‌فرض A و مقدار متفاوتی به عنوان نتیجه است. کنترل ترتیب و استفاده از پارامترهای تابع خارجی Modelica امکان‌پذیر است. در مثال زیر از این روش برای ارسال اندازه ابعاد آرایه به تابع خارجی در هنگام فراخوانی روال dgels، که به زبان Fortran نوشته شده است، استفاده می‌گردد. بعضی از روال‌های قدیمی فرترن مثل dgels نیاز به آرایه‌های فعال محلی دارند، آرایه‌هایی که در برنامه اصلی توسط متغیرهای محلی تعریف شده باشد و بعد از کلمه کلیدی protected تعریف می‌گردد. تخصیص حافظه به این آرایه محلی در برنامه اصلی صورت می‌گیرد و تابع خارجی از این حافظه استفاده خواهد نمود.

```
function LeastSquares "Solves a linear least squares problem"
        input Real A[ :, :];
        input Real B[: ,:];
        output Real Ares[size(A,1),size(A,2)]:=A;   // Factorization is returned
        in Ares for later use
        output Real x[size(A,2),size(B,2)];
protected
        Integer lwork = min(size(A,1),size(A,2))+
        max(max(size(A,1),size(A,2)),size(B,2))*32;
        Real work[lwork];
        Integer info;
        String transposed="NNNN";// Workaround for passing CHARACTER
                                              //data to Fortran routine
        external "Fortran 77"
        dgels(transposed, 100, size(A,1), size(A,2), size(B,2), Ares, size(A,1),
        B, size(B,1), work, lwork, info);
end LeastSquares;
```

۱۴-۱۶-۵) نگاه به الگوریتم‌ها به عنوان تابع

زمانی که از ساختار و مفهوم برنامه‌نویسی صحبت می‌گردد، تابع به عنوان بلوک ساختمانی (سازنده اصلی) برنامه مطرح است. تلاش برای درک ساختار الگوریتم در زبان Modelica مخصوصاً به شکل تابع بسیار سودمند است. برای مثال، بخش الگوریتم زیر را در نظر بگیرید که در متن یک معادله اتفاق می‌افتد:

```
algorithm
        y:=x;
        z := 2*y;
        y := z+y;
...
```

همان‌طور که در مثال زیر دیده می‌شود، این الگوریتم را می‌توان به یک تابع و یک معادله تبدیل کرد، بدون این که معنی آن تغییر کند. معادله، متغیر خروجی از الگوریتم قبلی را با نتیجه تابع fبرابر هم قرار می‌دهد. پارامترهای ورودی قراردادی تابع f به عنوان ورودی الگوریتم و پارامترهای قراردادی خروجی تابع به عنوان نتایج الگوریتم عمل خواهند کرد. کد بخش الگوریتم به عنوان بدنه تابع f آمده است.

```
(y,z) = f(x);
...
function f
        input Real x;
```

```
	output Real y,z;
algorithm
	y:=x;
	z := 2*y;
	y := z+y;
end f;
```

۱۴-۱۷) مدل کردن ترکیبی

از دید کلان، سیستم‌های فیزیکی عموماً به عنوان تابع پیوسته‌ای از زمان در نظر گرفته می‌شوند که از قوانین فیزیک تبعیت می‌کنند. برای مثال حرکت سیستمهای مکانیکی، سطح ولتاژ و جریان در سیستم‌های الکتریکی، واکنش‌های شیمیایی و ... . این گونه سیستم‌ها دارای دینامیک پیوسته هستند.

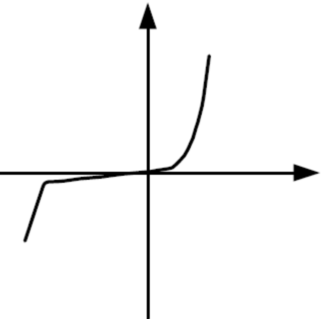
از طرف دیگر، بعضی وقت‌ها مفید است فرض کنیم که برخی از اجزاء سیستم‌های خاصی، رفتار ناپیوسته دارند. یعنی تغییر در مقدار متغیرهای سیستم ممکن است خیلی سریع و ناپیوسته اتفاق بیافتد. در سیستم‌های فیزیکی واقعی تغییرات می‌توانند خیلی سریع باشند ولی آنی نیستند. برای مثال برخورد در سیستم‌های مکانیکی، توپ جهنده که تقریباً به طور آنی تغییر جهت می‌دهد، کلیدها در مدارهای الکتریکی با تغییر سریع در سطح ولتاژ، شیرها و پمپ‌ها در کارخانه‌های شیمیایی و ... . منظورمان قطعات با دینامیک ناپیوسته است. دلیل استفاده از تقریب گسسته (رفتار دارای ناپیوستگی)، داشتن مدل ریاضی ساده‌تر است. این کار کمک می‌کند تا مدل ما قابل کنترل‌تر شود که خود باعث می‌شود شبیه‌سازی چندین بار سریعترگردد.

ازآنجایی که تقریب گسسته فقط برای بخشهای خاصی از سیستم حاکم است، با مدلی از سیستم روبه‌رو خواهیم بود که هم قطعات با رفتار پیوسته و هم قطعاتی با رفتار گسسته را شامل می‌گردد. این چنین سیستمی را سیستم ترکیبی می‌نامند و روش‌های مربوط به مدلسازی این سیستمها را مدلسازی ترکیبی می‌نامند. معرفی مدل‌های ترکیبی ریاضی مشکلات جدیدی در روش یافتن راه حل آنها ایجاد کرده است، اما معایب این نوع مدلسازی در مقابل فواید آن ناچیز است.

زبان Modelica برای بیان مدلهای ترکیبی دو نوع ساختار معرفی می‌نماید: عبارات شرطی یا معادلات برای تشریح مدل‌های گسسته و شرطی؛ و عبارات when برای بیان معادلاتی که فقط در ناپیوستگی‌ها نافذ هستند مانند وقتی شرایط خاصی اتفاق می‌افتد. برای مثال، عبارات شرطی if-then-else امکان استفاده ازمقادیر متفاوت در نواحی مختلف کاری را برای مدل کردن پدیده‌هایی که در نواحی مختلف رفتار متفاوت دارند، را فراهم نموده است. همانطور که در معادله پایین یک محدود کننده نشان داده شده است.

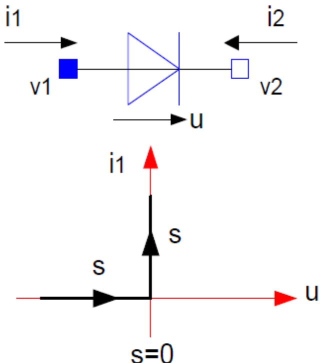
```
y = if u > limit then limit else u;
```

مثال کاملتری از یک مدل شرطی، مدل دیود ایده‌آل است. مشخصات واقعی یک دیود در شکل ۱۸-۱۴ نمایش داده شده است.



شکل ۱۸-۱۴.مشخصه یک دیود واقعی.

مشخصات دیود ایده‌آل به شکل پارامتری در شکل ۱۹-۱۴ آمده است.



شکل ۱۹-۱۴ .مشخصات دیود ایده‌آل.

از آنجایی که در نمودار ولتاژ- جریان، سطح ولتاژ دیود ایده‌آل به سمت بینهایت می‌رود، استفاده از تعرف پارامتری مناسب‌تر است، به این شکل که ولتاژ v و جریان i هر دو تابعی از پارامتر s هستند. وقتی دیود خاموش است هیچ جریان عبوری وجود ندارد و ولتاژ منفی است، درحالی که وقتی روشن است هیچ افت ولتاژی در دیود وجود ندارد و جریان عبوری داریم.

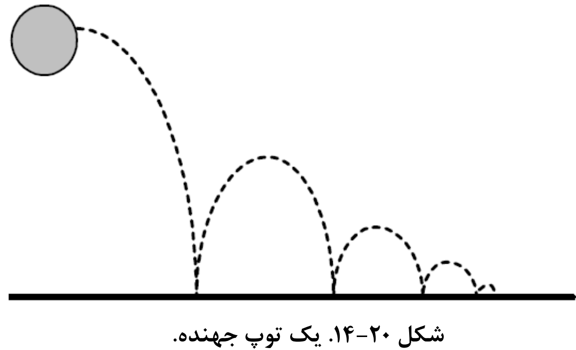
```
model Diode "Ideal diode"
	extends TwoPin;
	Real s;
	Boolean off;
equation
	off = s < 0;
```

```
    if off then
        v=s;
    else
        v=0;
    end if; // conditional equations
    i = if off then 0 else s; // conditional expression
end Diode;
```

عبارت when در Modelica برای بیان کردن معادلات آنی تعریف شده است، یعنی معادلاتی که فقط در نقاط معین نافذ هستند. برای مثال در ناپیوستگی‌ها، وقتی شرایط خاص اتفاق می‌افتد. ساختار عبارتwhen در معادله زیر نشان داده شده است.

```
when {condition1, condition2, …} then
    equations
end when;
```

یک توپ جهنده مثال خوبی از سیستم ترکیبی است.حرکت توپ به وسیله متغیر ارتفاعheight بالاتر از سطح زمین و سرعت عمودی v تعریف می‌گردد. توپ در بین جهش‌ها پیوسته حرکت می‌کند در حالی که در زمان جهش تغییرات گسسته اتفاق می‌افتد؛ همانطور که در شکل ۲۰-۱۴ نشان داده شده است. وقتی توپ با زمین برخورد می‌کند جهت سرعتش برعکس می‌شود. یک توپ ایده‌آل دارای ضریب الاستیسیته ۱ می‌باشد و در هنگام برخورد هیچ انرژی را از دست نمی‌دهد. یک توپ واقعی‌تر، همانطوری که در مثال زیر مدل شده است، دارای ضریب الاستیسیته ۰.۹ می‌باشد که باعث می‌شود ۹۰٪ سرعت خود را بعد از هر برخورد نگه دارد.



شکل ۲۰-۱۴. یک توپ جهنده.

مدل توپ جهنده شامل دو معادله اساسی حرکت است که سرعت، ارتفاع و همین طور شتاب ناشی از نیروی جاذبه را به هم ربط می‌دهد. در لحظه برخورد مقدار سرعت اندکی کاهش یافته و

جهت سرعت به طور ناگهانی معکوس می‌شود. یعنی v(after bounce) = -c*v(before bounce)که توسط شکل خاصی از معادله لحظه‌ای reinit(v,-c*v) انجام شده است.

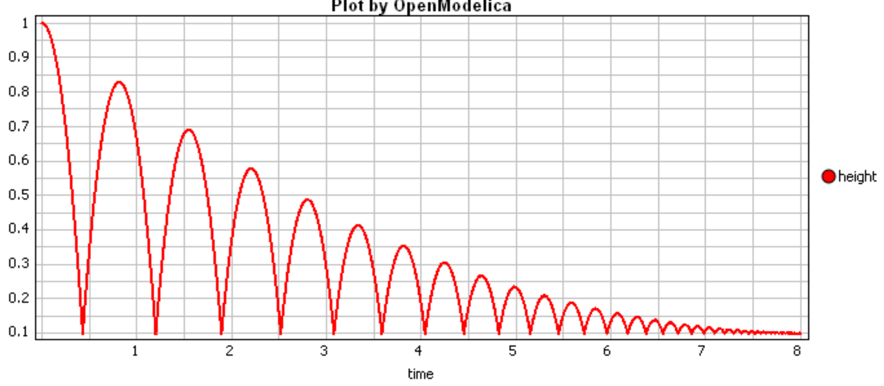
```
model BouncingBall "the bouncing ball model"
        parameter Real g = 9.81; // gravitational acceleration
        parameter Real c = 0.90; // elasticity constant of ball
        Real height (start=0); // height above ground
        Real v (start=5); // velocity
Equation
        der(height) = v ;
        der(v) = -g ;
        when height < 0 then
                reinit(v, -c*v) ;
        end when;
end BouncingBall ;
```

در ادامه برای هشت ثانیه اول مثال توپ جهنده شبیه‌سازی را انجام می‌دهیم:

```
simulate( BouncingBall, stopTime=8 );
```

ارتفاع توپ را رسم می‌کنیم:

```
plot( height);
```



شکل ۲۱-۱۴. نمودار ارتفاع توپ جهنده.

۱۴-۱۸) بسته‌ها[1]

وقتی می‌خواهیم کدهایی ایجاد کنیم که قابلیت استفاده مجدد داشته باشند، مانند کتابخانه کلاس‌ها و توابع Modelica، ایجاد ناسازگار در اسامی استفاده شده به علت استفاده از اسامی تکراری یک مشکل عمده است. اهمیتی ندارد که شما چقدر اسامی کلاس‌ها و متغیرها را با دقت انتخاب کرده باشید، ممکن است شخص دیگری همان اسم را برای هدف دیگری استفاده کرده باشد. اگر شما از اسم‌های کوتاه استفاده کنید، این اسامی‌ها بعلت کوچکتر بودن برای استفاده آسان‌تر هستند و نسبتاً محبوب‌تر، احتمال استفاده از اسامی یکسان بیشتر شده و مشکل ناسازگاری اسامی بدتر می-شود. راه حل چیست؟

یک راه حل معمول برای جلوگیری از تداخل اسم‌ها استفاده از یک پیشوند کوتاه قبل از اسامی هر گروه از اسم‌های مربوط به هم و قرار دادن آنها در بسته‌های مجزا است. برای مثال، تمامی اسم‌ها در X-windows دارای پیشوند Xt هستند، و WIN32 پیشوندی برای API های ویندوز ۳۲ بیتی است. این کار برای بسته‌های کوچک به طور معقول کار می‌کند ولی با بزرگتر شدن بسته‌ها احتمال تداخل اسم‌ها افزایش می‌یابد.

خیلی از زبان‌های برنامه‌نویسی مانند Java و Ada و همین طور Modelica با تعریف مفهوم بسته، راه اصولی‌تر و ایمن‌تری برای اجتناب از تداخل اسامی ارائه نموده‌اند. یک بسته یک ظرف[2] یا یک فضای اسمی[3] برای اسامی کلاس‌ها، توابع، ثابت‌ها، و تمامی تعاریف مجاز می‌باشد. نام بسته با استفاده از اعلان استاندارد نقطه‌ای به صورت پیشوند به تمام اسامی درون بسته اضافه می‌شود. تعاریف (هر چیز قابل تعریف) را می‌توان در فضای اسمی یک بسته وارد کرد. در بخشهای گذشته از کلمه کتابخانه به جای بسته استفاده نمودیم که با توجه به مفهوم بسته واژه مناسبی است؛ در این بخش از واژه بسته استفاده خواهیم نمود.

در حال حاضر Modelica مفهوم بسته را به صورت کلاس بامحدودیت تعریف کرده است. بنابراین، می‌توان از توارث برای وارد کردن تعاریف از بسته‌های دیگر به فضای اسمی بسته استفاده نمود. این کار نسبتاً خوب عمل می‌کند ولی از آنجایی که توارث اختصاصاً برای وارد کردن کلاس به یک بسته نیست باعث ایجاد مشکلات مفهومی در مدل‌سازی می‌گردد. در مثال زیرگونه voltage به وسیله توارث وارد بسته شده است، این کار استفاده از آن را بدون پیشوند برای اعلان متغیرv ممکن می‌سازد. برعکس تعریف متغیر I که با استفاده از دستور کامل و با استفاده از نامگذاری نقطه‌ای از نوع Ampere نوشته شده است، که ممکن بود مانند متغیر v به اختصار تعریف شود.

```
package mypack
```

---

[1] Package
[2] Container
[3] Name space

```
        extends Modelica.SIunits;
class foo;
        Voltage V;
        Modelica.SIunits.Ampere I;
        ...
end foo;
...
end mypack;
```

وارد کردن تعاریف از یک بسته به یک بسته دیگر درست مانند مثال بالا دارای این اشکال است که معرفی تعاریف جدید در یک بسته ممکن است باعث تداخل تعاریف در بسته‌هایی شود که از آن بسته استفاده می‌کنند. برای مثال اگر یک تعریف جدید که V نامیده می‌شود به بسته Modelica.SIunits اضافه شود، در بسته mypack خطای کامپایل به وجود می‌آید.

یک راه حل جایگزین برای حل مشکل نام‌های کوتاه، که در صورت اضافه شدن تعاریف جدید به کتابخانه اصلی باعث خطای ترجمه نگردد، این است که به جای پیشوندهای طولانی بسته‌ها، اسامی مستعار مناسب (با توجه به نام بسته) به عنوان پیشوند معرفی شود. این کار با استفاده از تعریف کوتاهی برای بسته ممکن است؛ نام SI به جای نام طولانی‌تر Modelica.SIUnits برای بسته تعریف می‌گردد. هنوز هم اسم کامل Amper قابل استفاده است.

```
package mypack
        package SI= Modelica.SIunits;
class foo;
        SI.Voltage V;
        Modelica.SIunits.Ampere I;
        ...
end foo;
        ...
end mypack;
```

۱۴-۱۹) پیاد سازی واجرای Modelica

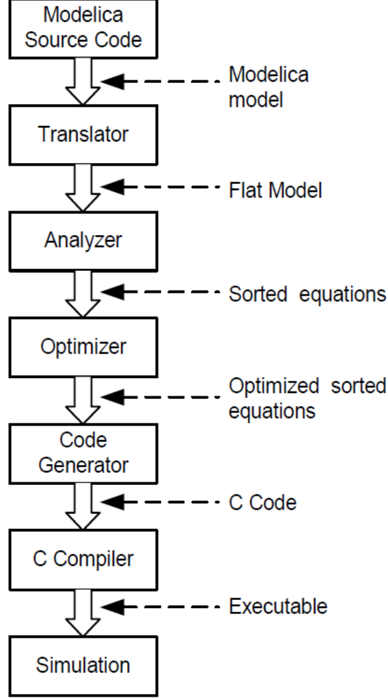
به منظور درک بهتر از نحوه کارکرد Modelica، مفید است که به فرآیند ترجمه و اجرای مدل توسط مترجم Modelica نگاهی بیاندازیم (شکل ۲۲-۱۴). در ابتدا کد منبع Modelica تجزیه شده و به شکل ساختاری قابل فهم برای مترجم تبدیل می‌شوند، معمولاً به شکل درخت خلاصه ساختار در می‌آید. این ساختار ارزیابی می‌گردد، بررسی گونه‌ها انجام می‌شود، کلاس‌ها جانشین شده و گسترش می‌یابند، اصلاحات و مقدار دهی اولیه انجام می‌شود، عبارات اتصال به معادلات تبدیل می‌شوند و .... نتیجه این تحلیل و ترجمه مجموعه‌ای یک دست از معادلات، ثابت‌ها، متغیرها و تعریف

توابع است. بجز علامت‌گذاری نقطه‌ای اسامی، هیچ نشانه‌ای از ساختار شیءگرا در ترجمه نهایی باقی نمی‌ماند. این کار را مسطح کردن می‌نامیم.

بعد از مسطح‌سازی، همه معادلات بر اساس جریان داده‌های بین معادلات به صورت مکانی دسته‌بندی می‌گردند. در حالت کلی معادلات دیفرانسیل جبری (DAEs) این کار فقط دسته بندی معادلات نخواهد بود، بلکه دستکاری معادلات برای تبدیل ماتریس‌ضرایب به ماتریس پایین مثلثی که تبدیل BLT نامیده می شود، نیز صورت می‌گیرد. سپس یک ماژول بهینه‌ساز شامل الگوریتم ساده‌سازی جبری، روشهای نمادین Index reduction و ...، خیلی از معادلات را حذف می‌کنند و در نهایت فقط مجموعه کوچکی باقی می‌ماند تا به صورت عددی حل گردد. به عنوان یک مثال ساده، اگر دو معادله مشابه ظاهر شوند، فقط یک نسخه از آنها باقی می‌ماند. سپس معادلات مستقل به عبارات جایگزینی تبدیل می‌شوند. این کار از آنجایی امکان‌پذیراست که معادلات دسته‌بندی شده‌اند و ترتیب اجرا برای ارزیابی معادلات با توجه به گام‌های حل‌کننده عددی مشخص شده است. اگر مجموعه‌ای از معادلات درهم تنیده ظاهر شود، این مجموعه توسط یک حل‌کننده نمادین با اعمال تبدیل‌های جبری چنان تغییر می‌یابند که روابط و وابستگی بین متغیرها ساده‌تر گردد. اگر معادلات دیفرانسیل پاسخ تحلیلی داشته باشند، بعضی وقتها ممکن است با استفاده از حل‌کننده نمادین این پاسخ‌ها را یافت. در نهایت کد C تولید می‌شود و با حل‌کننده عددی معادلات ادغام می‌گردد تا معادلات باقی مانده که بسیار ساده شده‌اند را حل کند.

تقریب مقادیر اولیه از تعریفات موجود در مدل استخراج می‌گردد یا متقابلاً توسط کاربر مشخص می‌گردد. همچنین اگر لازم باشد کاربر مقادیر پارامترها را مشخص می‌کند. یک حل‌کننده عددی معادلات جبری-دیفرانسیلی، مقادیر متغیرها را برای مدت زمان مشخص شده برای شبیه‌سازی $[t_0,t_1]$ محاسبه می‌کند. نتیجه شبیه‌سازی سیستم دینامیک مجموعه‌ای از توابع زمانی است، مانند R2.v(t) در مدل مدار ساده. این توابع را می‌توان به صورت یک نمودار نمایش داد و در یک فایل ذخیره کرد.

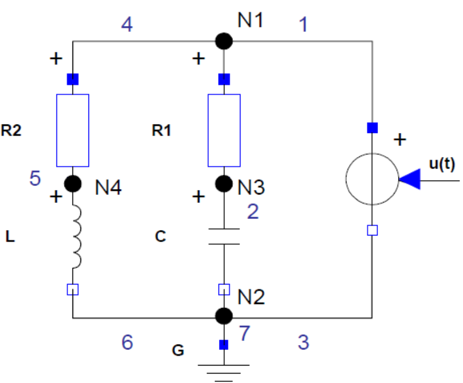
در بیشتر حالات (ولی نه همیشه) عملکرد کدهای شبیه‌سازی تولید شده (شامل حل‌کننده)، شبیه به کدهایی است که برنامه‌نویسان به زبان C می‌نویسند. اغلب کدهای تولید شده توسط Modelica در مقایسه با کدنویسی مستقیم در C و آن چیزی که یک برنامه‌نویس می‌تواند از پس آن برآید، بسیار موثرتر است؛ زیرا اگر ممکن باشد بهینه‌سازیهای نمادین معادلات به وسیله سیستم انجام می‌شود.



شکل ۲۲-۱۴. مراحل ترجمه و اجرای یک مدل Modelica.

۱۴-۱۹-۱) ترجمه دستی مدل مدار ساده

اجازه دهید بار دیگر به مدار ساده گذشته بازگردیم، که قبلاً در شکل ۳-۱۴ نشان داده شده است، برای راحتی خوانندگان دوباره مدار رادر شکل ۲۳-۱۴ آورده‌ایم. برای درک فرایند ترجمه، این مدل را به صورتی دستی ترجمه می‌کنیم.



شکل ۲۳-۱۴. نمایش دوباره مدار ساده الکتریکی.

کلاس‌ها، نمونه‌ها و معادلات بر اساس قوانین زیر به مجموعه‌ای یک‌دست از معادلات، ثابتها و متغییرها (معادلات جدول ۳-۲را ببینید)، ترجمه می‌شوند:

- برای هر نمونه کلاس (هر قطعه)، یک نسخه از همه معادلات این نمونه را به کل معادلات جبری-دیفرانسیلی یا معادلات دیفرانسیل معمولی (ODE) سیستم اضافه نمایید. نوشتن معادلات به هر دو حالت ODE یا DAE ممکن است، چون DAE به طور معمول می‌تواند به ODE تبدیل شود.
- برای هر اتصال بین نمونه قطعات در مدل، معادلات اتصال را چنان به سیستم DAE اضافه نمایید که متغیرهای غیرجریانی با هم برابر و جمع متغیرهای جریانی برابر صفر گردد.

کلاس Resistor از کلاس TwoPin ارث می‌برد و معادله v=p.v- n.v به وسیله کلاس TwoPin تعریف شده است؛ پس کلاس Resistor شامل این معادله هم می‌شود. کلاس SimpleCircuit شامل متغیر R1 از نوع Resistor است. بنابراین، ما این معادله را برای R1 به شکل R1.v = R1.p.v - R1.n.v در سیستم معادلات در نظر می‌گیریم.

سیمی که در مدل با شماره ۱ برچسب گذاری شده، به شکل connect(AC.p,R1.p) بیان می‌گردد. متغیرهای AC.p و R1.p از نوع Pin هستند. متغیر v از نوع متغیر غیرجریانی است و ولتاژ را نشان می‌دهد. بنابراین، معادله تساوی R1.p.v=AC.p.v تولید می‌شود. وقتی متغیرهای غیرجریانی داریم همیشه معادلات تساوی تولید می‌شوند.

سیم دیگری (با برچسب ۴) به همان پین R1.p متصل شده است. این سیم توسط دستور اتصال دیگری بیان می‌شود: connect(R1.p.R2.p). برای این دستور نیز معادله مشابهی می‌نویسیم.

در اتصالات بالا متغییر i به عنوان متغیر جریان تعریف می‌شود، بنابراین معادله AC.p.i+R1.p.i+R2.p.i=0 تولید می‌شود. وقتی متغیرهای جریانی داریم همیشه مطابق قانون جریان کیرشهف، معادلات با مجموع صفر تولید می‌شوند.

مجموعه کامل معادلات تولید شده از کلاس SimpleCircuit شامل ۳۳ معادله جبری-دیفرانسیلی است (به جدول ۲-۱۴ نگاه کنید). این معادلات شامل ۳۳ متغیر، همچنین زمان و تعدادی پارامتر و ثابت‌ها می‌باشد.

جدول ۲-۱۴.معادلات استخراج شده از مدار ساده الکتریکی.

AC	0 = AC.p.i + AC.n.i AC.v = AC.p.v − AC.n.v AC.i = AC.p.i AC.v = AC.u AC.u = AC.VA ∗ sin (2 ∗AC.PI ∗ AC.f ∗ time);	L	0 = L.p.i + L.n.i L.v = L.p.v − L.n.v L.i = L.p.i L.v = L.L ∗ der(L.i)

R1	0 = R1.p.i + R1.n.i R1.v = R1.p.v − R1.n.v R1.i = R1.p.i R1.v = R1.R ∗ R1.i	G	G.p.v = 0
R2	0 = R2.p.i + R2.n.i R2.v = R2.p.v − R2.n.v R2.i = R2.p.i R2.v = R2.R ∗ R2.i	wires	R1.p.v = AC.p.v C.p.v = R1.n.v AC.n.v = C.n.v R2.p.v = R1.p.v L.p.v = R2.n.v L.n.v = C.n.v G.p.v = AC.n.v
C	0 = C.p.i + C.n.i C.v = C.p.v − C.n.v C.i = C.p.i C.i = C.C ∗ der(C.v)	flow at node	0 = AC.p.i + R1.p.i + R2.p.i 0 = C.n.i + G.p.i + AC.n.i + L.n.i 0 = R1.n.i + C.p.i 0 = R2.n.i + L.p.i

در پایین ۳۳ متغیر در سیستم معادلات مشاهده می‌کنید، که از آنها ۳۱متغیر جبری هستند. دو متغیر C.v و L.i با توجه به این که مشتقات آنها در معادلات ظاهر شده است، متغیرهای حالت هستند.

جدول ۳-۱۴.متغیرهای مدل مدار ساده الکتریکی.

R1.p.i	R1.n.i	R1.p.v	R1.n.v	R1.v
R1.i	R2.p.i	R2.n.i	R2.p.v	R2.n.v
R2.v	R2.i	C.p.i	C.n.i	C.p.v
C.n.v	C.v	C.i	L.p.i	L.n.i
L.p.v	L.n.v	L.v	L.i	AC.p.i
AC.n.i	AC.p.v	AC.n.v	AC.v	AC.i
AC.u	G.p.i	G.p.v		

۱۴-۱۹-۲) تبدیل به فضای حالت

سیستم معادلات جبری-دیفرانسیلی ضمنی (سیستم DAE) در جدول ۳-۱۴ باید قبل از به کار بردن حل‌کننده عددی ساده‌تر شوند. قدم بعدی تعیین نوع متغیرهای سیستم DAE می‌باشد. ما چهار گروه زیر را داریم :

- همه متغیرهای ثابت که پارامتر شبیه‌سازی هستند، بنابراین به آسانی در بین شبیه‌سازی‌ها قابل اصلاح بوده و توسط کلمه کلیدی parameter تعریف می‌شوند، این متغیرها در بردار

پارامترها p گردآوری می‌شوند. بقیه ثابت‌ها می‌توانند توسط مقدارشان جایگزین شوند؛ بنابراین به جای آنها مقدار عددیشان قرار می‌گیرد و از معادلات حذف می‌شوند.

- متغیرهایی که با خصوصیت ورودی تعریف شده‌اند، دارای کلمه کلیدی input به عنوان پیشوند هستند. این متغیرها در سلسله مراتب نمونه‌ها در بالاترین سطح قرار دارند. این متغیرها در بردار ورودی u گردآوری می‌شوند.
- متغیرهایی که عملگر ()der بر آنها اعمال شده است یعنی مشتقات آنها در مدل وجود دارد را در بردار حالت x جمع‌آوری می‌کنیم.
- همه متغیرهای باقیمانده که مشتقات آنها در مدل ظاهر نمی‌شوند، در بردار جبری y جمع‌آوری می‌شوند.

برای مدل مدار ساده این چهار گروه متغیر را می‌بینید:

```
p = {R1.R, R2.R, C.C, L.L, AC.VA, AC.f}
u = {AC.u}
x = {C.v, L.i}
y = {R1.p.i, R1.n.i, R1.p.v, R1.n.v, R1.v, R1.i, R2.p.i, R2.n.i,
R2.p.v, R2.n.v, R2.v, R2.i, C.p.i, C.n.i, C.p.v, C.n.v, C.i,
L.n.i, L.p.v, L.n.v, L.v, AC.p.i, AC.n.i, AC.p.v, AC.n.v, AC.i,
AC.u, AC.v, G.p.i, G.p.v}
```

ما دوست داریم مسأله را به ساده‌ترین شکل و با کوچکترین دستگاه معادلات دیفرانسیلی معمولی (ODE) ممکن بیان کنیم (در حالت کلی سیستم DAE) و مقادیر بقیه متغیرها را از حل این مسأله کوچک محاسبه کنیم. سیستم معادلات باید ترجیحاً به شکل فضای حالت باشد، یعنی به شکل زیر:

```
x'= f (x,t)
```

که 'x، مشتق بردار حالت x، برابر با تابع برداری از بردار حالت x و زمان است. با استفاده از روش حل عددی تکراری برای دستگاه معادلات دیفرانسیلی معمولی ، در هر گام تکرار، مشتق بردار حالت از بردار حالت در همان زمان محاسبه می‌شود.

برای مدل مدار ساده داریم:

```
x = {C.v,L.i}, u = {AC.u}
(with constants: R1.R,R2.R,C.C,L.L,AC.VA,AC.f,AC.PI)
x' = {der(C.v),der(L.i)}
```

۱۴-۱۹-۳) روش حل

ما از روش حل عددی تکراری استفاده خواهیم کرد. اول فرض کنید که مقادیر تخمینی بردار حالت `x ={C.v,L.i}` در زمان t=0 (وقتی شبیه‌سازی شروع می‌شود) در دسترس است. از یک تقریب عددی برای مشتق 'x (یعنی der(x)) در زمان t استفاده می‌کنیم:

der(x) = (x(t+h)-x(t))/h

در نتیجه تقریب x در زمان t+h برابر است با:

x(t+h) = x(t)+der(x)*h

اگر بتوانیم مشتق x را در زمان مورد نظر محاسبه نماییم با استفاده از رابطه بالا می‌توان مقدار x را در لحظه بعدی (گام بعدی) یعنی t+h محاسبه نمود. مشتق بردارحالت der(x)، را می‌توان از رابطه x'=f(x,t) محاسبه کرد. با انتخاب کردن معادله‌های شامل der(x) و استخراج متغیرهای بردار x برحسب متغیرهای دیگر، به صورت زیر مشتق بردار x قابل محاسبه است:

der(C.v) = C.i/C.C
der(L.i) = L.v/L.L

حل بقیه معادلات در سیستم DAE نیازمند محاسبه مجهولات C.i و L.v در معادلات بالا است. ازC.i شروع می‌کنیم، با استفاده از شماری از معادلات مختلف با جایگزینی ساده و دستکاری جبری، ما به سه معادله زیر می‌رسیم:

C.i = R1.v/R1.R
using:
C.i = C.p.i = -R1.n.i = R1.p.i = R1.i = R1.v/R1.R

R1.v = R1.p.v - R1.n.v = R1.p.v - C.v
using:
R1.n.v = C.p.v = C.v+C.n.v = C.v+AC.n.v = C.v+G.p.v=C.v+0
= C.v

R1.p.v = AC.p.v = AC.VA*sin(2*AC.f*AC.PI*t)
using:
AC.p.v = AC.v+AC.n.v =
AC.u+G.p.v=AC.VA*sin(2*AC.f*AC.PI*t)+0

با روشی مشابه معادلات باقیمانده را به صورت زیر به دست می‌آوریم:

```
L.v = L.p.v-L.n.v = R1.p.v - R2.v
using:
L.p.v = R2.n.v = R1.p.v-R2.v
and: L.n.v = C.n.v = AC.n.v = G.p.v = 0

R2.v = R2.R*L.p.i
using:
R2.v = R2.R*R2.i = R2.R*R2.p.i = R2.R*(-R2.n.i)= R2.R*L.p.i = R2.R*L.i
```

همه پنج معادله را یک جا می‌نویسیم:

```
C.i         = R1.v/R1.R
R1.v        = R1.p.v - C.v
R1.p.v      = AC.VA*sin(2*AC.f*AC.PI*t)
L.v         = R1.p.v - R2.v
R2.v        = R2.R*L.i
```

با مرتب کردن معادلات براساس وابستگی داده‌ها می‌توان معادلات بالا را به دستورات جایگزینی تبدیل کرد. ما به مجموعه دستورات جایگزینی زیر می‌رسیم که باید در هر تکرار با داشتن مقادیر C.v، L.i و t محاسبه شوند.

```
R2.v := R2.R*L.i
R1.p.v := AC.VA*sin(2*AC.f*AC.PI*t)
L.v := R1.p.v - R2.v
R1.v := R1.p.v - C.v
C.i := R1.v/R1.R
der(L.i) := L.v/L.L
der(C.u) := C.i/C.C
```

بهتر از روش بیان شده، این معادلات جایگزینی را می‌توان به کدهایی در یک زبان برنامه‌نویسی مانند C تبدیل کرده و همراه با یک حل‌کننده مناسب ODE، با روش تقریب حل نمود. روشی که در مثال بالا شرح داده روش تکراری رو به جلو به نام روش انتگرالی Euler خوانده می‌شود. همچنین می‌توان روش تبدیل جبری که ما به صورت دستی و به سختی برای ساده‌سازی معادلات استفاده کردیم را به صورت کاملاً خودکار انجام داد، که به نام تبدیل BLT شناخته می‌شود که ماتریس ضرایب سیستم معادلات را به شکل ماتریس پایین مثلثی در می‌آورد.

۲۶ متغیر جبری باقیمانده در سیستم معادلات مدل مدار ساده که جزئی از ۷ معادله اصلی سیستم ODE که در بالا حل شد، نیستند، را می‌توان پس از انجام محاسبات اصلی به دست آورد. این محاسبات برای حل سیستم اصلی ODE لازم نیست.

لازم است تأکید شود که مثال مدل مدار ساده همان طور که از نامش و تعداد قطعاتش پیداست، بسیار ساده بود. شبیه‌سازی مدلهای واقعی شامل ده‌ها هزار از معادلات، معادلات غیرخطی، مدلهای ترکیبی و ... است. تبدیلات نمادین و کاهش سیستم معادلاتی که واقعاً توسط مترجم Modelica انجام می‌شود، بسیار پیچیده‌تر از کاهش تعداد معادلات و شکستن معادلاتی است که در این مثال نشان داده شد.

فهرست مراجع

1. **Consortium, Open Source Modelica.** *OpenModelica Users Guide.* 2012.

2. **Consortium, Open Source Modelica.** *OMEdit OpenModelica Connection Editor User Manual.* 2012.

3. **Research, Wolfram.** *Wolfram SystemModeler, Getting Started.* 2012.

4. **Fritzson, Peter.** *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* s.l. : Wiley-IEEE Press, 2003. ISBN:0-471-471631.