





ЛЕКЦИЯ 9

СЪЗДАВАНЕ НА КОМПОНЕНТИ ОТ ВИСОКО РАВНИЩЕ

-  **Разделяне на модела CedarBog**
-  **Компоненти от високо равнище в модела CedarBog**
-  **Разделяне на модели с региони**
-  **Състояния и преходи между състояния в йерархични модели**

ВЪВЕДЕНИЕ

**SIMPLEX MDL поддържа йерархични модели.
Това означава, че компонента може да се
изгради от подкомпоненти, които са
свързани чрез връзки (Connections).**

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

Разгледахме модела CedarBog като единична компонента. Възможно е модела да се раздели. Едно от най-важните свойства на SIMPLEX MDL е, че се позволява моделите да бъдат разделяни по произволен начин без това да влияе върху симулационните резултати.

Ще представим примерно разделяне на модела CedarBog, което съдържа слънцето като източник, езерото, околната среда и мъртвата органична материя.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

BASIC COMPONENT Sun1

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

CONSTANTS

Pi(REAL) := 3.14

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

DEPENDENT VARIABLES

CONTINUOUS

suns (REAL[kJ/m²]) := 0 [kJ/m²] #

Слънчева енергия

DYNAMIC BEHAVIOUR

suns := 95.9[kJ/m²] * (1+0.635 * SIN (2[1/a] * Pi
*T));

END OF Sun1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

Всяка от тези четири компоненти е независима основна компонента. Те са свързани заедно в компонентата от по-високо равнище CedarBog_High1.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

BASIC COMPONENT Lake1

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

CONSTANTS

Bio_Fac (REAL[a]) := 1E-15 [a]

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

STATE VARIABLES

CONTINUOUS

p (REAL[t]) := 0 [t], # Растения
h (REAL[t]) := 0 [t], # Тревопасни
c (REAL[t]) := 0 [t] # Месоядни

DEPENDENT VARIABLE

CONTINUOUS

sun_bio (REAL[t/a]) := 0 [t/a]

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

SENSOR VARIABLE

CONTINUOUS

`sun (REAL[kJ/m^2]) := 95.9 [kJ/m^2] #`

Слънчева енергия

DYNAMIC BEHAVIOUR

`sun_bio := sun * Bio_Fac;`

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

DIFFERENTIAL EQUATIONS

$$p' := \text{sun_bio} - 4.03[1/a] * p;$$

$$h' := 0.48[1/a] * p - 17.87[1/a] * h;$$

$$c' := 4.85[1/a] * h - 4.65[1/a] * c;$$

END

END OF Lake1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

Променливи, които правят достъпна стойността на променлива от друга компонента се наричат сензорни променливи.

Ако от вън се внесе променлива, тя трябва да се декларира като сензорна променлива.

Такъв е случая за променливата `sun` в компонентата `Lake1`. Не е необходима съответна декларация за променлива, която ще се експортира. В компонентата `Sun1` няма индикация, че друга компонента ще има достъп до променливата `suns`. Това означава, че всяка величина на модела в компонентата е достъпна отвън без това да се знае вътре в компонента. Този подход се нарича `glass-box-concept`. Той има предимството, че други компоненти могат да декларират сензорни променливи без да е нужно да се изменя компонентата, чийто променливи са достъпни. В частност не е необходимо да се прекомпилира компонента.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

Връзката символизира причинена зависимост, а не поток от материал или енергия. Динамичното поведение на компонентата Lake1 се влияе от променливата sun, чието поведение се определя в компонента Sun1. Това означава, че има причинна връзка от компонента Sun1 към компонента Lake1.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

BASIC COMPONENT Organic1

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

STATE VARIABLES

CONTINUOUS

o (REAL[t]) := 0 [t] # Мъртва органична материя

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

SENSOR VARIABLES

CONTINUOUS

p (REAL[t]),	# Растения
h (REAL[t]),	# Тревопасни
c (REAL[t])	# Месоядни

DYNAMIC BEHAVIOUR

DIFFERENTIAL EQUATIONS

$$o' := 2.55[1/a] * p + 6.12[1/a] * h + 1.95[1/a] * c;$$

END

END OF Organic1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

Ще разгледаме основните компоненти Organic1 и Environ1. Тяхното поведение се влияе от променливите на състоянието p , h и c от компонентата Lake.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

BASIC COMPONENT Environ1

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

STATE VARIABLES

CONTINUOUS

$e \text{ (REAL[t])} := 0 \text{ [t]}$

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

SENSOR VARIABLES

CONTINUOUS

p (REAL[t]),	# Растения
h (REAL[t]),	# Тревопасни
c (REAL[t])	# Месоядни

DYNAMIC BEHAVIOUR

DIFFERENTIAL EQUATIONS

$e' := 1.00[1/a] * p + 6.90[1/a] * h + 2.70[1/a] * c;$

END

END OF Environ1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

Досега бяха описани само основни компоненти. Чрез деклариране на променливи като сензорни, техните стойности се внасят отвън. Връзката все още не е описана. Все още имаме да определим кои променливи от кои компоненти ще се свържат. Тази връзка се дефинира в компоненти от високо равнище.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

HIGH LEVEL COMPONENT CedarBog_High1

SUBCOMPONENTS

Environ1,

Organic1,

Lake1,

Sun1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG COMPONENT CONNECTIONS

Lake1.c --> Organic1.c;

Lake1.h --> Organic1.h;

Lake1.p --> Organic1.p;

Sun1.suns --> Lake1.sun;

Lake1.c --> Environ1.c;

Lake1.h --> Environ1.h;

Lake1.p --> Environ1.p;

END OF CedarBog_High1

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARVOG

Забележка:

Динамичното поведение на модела се описва в основната компонента. Компонентите от високо равнище описват само структурата на модела.

Различни връзки могат да имат начало в един и същ елемент.

$A.s \rightarrow (B.x, C.y) ;$

е синоним на:

$A.s \rightarrow B.x;$

$A.s \rightarrow C.y;$

Не са позволени множество връзки към един и същ елемент, за да се осигури уникалност.

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

За да се изпълни модела са необходими следните стъпки:

- 1) Редактират се основните компоненти Sun1, Lake1, Organic1 и Environ1 и компонентата от високо равнище CedarBog_High1.
- 2) Използва се командата Check Version, за да провери синтаксиса на основните компоненти, за да се преведат в С код. Това ги привежда в състояние Checked (проверена).

Командата Check Version за компонента от високо равнище CedarBog_High1 води до изпълнението на следните действия:

- 1) Синтактична проверка на компонентата CedarBog_High1 и всички нейни подчинени компоненти и превеждането им в С код.
- 2) Проверка на съгласуваността. Тук се проверява дали са валидни връзките, създадени в компонентата от високо равнище.

Командата Prepare version компилира всички компоненти в обектен код и ги привежда в състояние Prepared (подготвена).

РАЗДЕЛЯНЕ НА МОДЕЛА CEDARBOG

Тогава трябва да се създаде и инсталира модел.

На него трябва да му се даде име MCedarBog-Version1. Допълнително трябва да се създаде и стартира нов експеримент ECedarBog-Version1.

Създава се наблюдател. В йерархично структуриран модел трябва да се даде пълния път на наблюдателя, за да следи съответните променливи. Това означава, че трябва да се отбележат имената на моделните величини и имената на компонентите, които той съдържа.

Компоненти от високо равнище в модела CedarBog

Сега разделяме допълнително модела CedarBog.

Компонентата Lake се разбива на три подкомпоненти, за да представи индивидуалните пластове вода. Приема се, че наситеността на слънчевата радиация е различна във всеки пласт.

Компоненти от високо равнище в модела CedarBog

Трите нива са идентични като структура и динамики.

Това означава, че всичките три нива могат да се разглеждат като инстанции на класа Layer2.

Всички константи са запазени заедно, заради опростеност. Трябва да се отчита, че динамичното поведение на всяко равнище да може да се различава. Единственото изменение е добавянето на константата f , която представя различната сила на слънчевата радиация. В описанието на класа Layer2 константата f се инициализира с $f := 1$. Тогава всяка инстанция ще получи своя собствена стойност.

Компоненти от високо равнище в модела CedarBog

Забележка:

Компонентата Layer2 може да се изпълни също
като независима основна компонента.

Компоненти от високо равнище в модела CedarBog

BASIC COMPONENT Layer2

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

CONSTANT

f (REAL) := 1, # Слънчев фактор

Bio_Fac (REAL[a]) := 1E-15 [a]

Компоненти от високо равнище в модела CedarBog

STATE VARIABLES

CONTINUOUS

$p \text{ (REAL[t])} := 0 \text{ [t]},$

$h \text{ (REAL[t])} := 0 \text{ [t]},$

$c \text{ (REAL[t])} := 0 \text{ [t]},$

$o \text{ (REAL[t])} := 0 \text{ [t]},$

$e \text{ (REAL[t])} := 0 \text{ [t]}$

DEPENDENT VARIABLE

CONTINUOUS

$\text{sun_bio (REAL[t/a])} := 0 \text{ [t/a]}$

SENSOR VARIABLE

CONTINUOUS

$\text{sun (REAL[kJ/m}^2\text{])} := 0 \text{ [kJ/m}^2\text{]}$

Компоненти от високо равнище в модела CedarBog

DYNAMIC BEHAVIOUR

$\text{sun_bio} := \text{sun} * \text{Bio_Fac};$

DIFFERENTIAL EQUATIONS

$p' := f * \text{sun_bio} - 4.03[1/a] * p;$

$h' := 0.48[1/a] * p - 17.87[1/a] * h;$

$c' := 4.85[1/a] * h - 4.65[1/a] * c;$

$o' := 2.55[1/a] * p + 6.12[1/a] * h + 1.95[1/a] * c;$

$e' := 1.00[1/a] * p + 6.90[1/a] * h + 2.70[1/a] * c;$

END

END OF Layer2

Компоненти от високо равнище в модела CedarBog

В моделното описание за Lake2 трябва първо да регистрираме списъка на подкомпонентите, които той съдържа. Това са трите инстанции на класа Layer2, наречени Layer2_1, Layer2_2 и Layer2_3.

В началото трите пласта се инициализират идентично. Те са идентични инстанции на един компонентен клас.

Последователно всички променливи във всяка инстанция могат да получат индивидуални стойности при инициализация. Това се извършва от компонентата от високо равнище, в случаят компонентата Lake2.

Като допълнение трябва да декларираме променливите в Lake2. В компонентите от по-високо равнище, които са сами по себе си подкомпоненти и могат също да съдържат подкомпоненти, променливите не могат да бъдат променяни.

Компоненти от високо равнище в модела CedarBog

Те само могат да преминат от едно равнище в йерархията към друго.

Декларацията на моделните величини, които се разпространяват отгоре надолу, се извършва чрез входна връзка. Моделните величини, които са декларирани във входна връзка отговарят на сензорни променливи. В този случай връзка може да се създаде при следващото по-високо равнище.

Декларацията на входна връзка трябва да декларира към коя моделна величина в коя компонента ще премине съответната променлива.

Затова входната връзка има функции от две части. Тя декларира моделна величина в компонентата и определя с какво ще се свърже тази моделна величина.

Компоненти от високо равнище в модела CedarBog

Изходните еквиваленти се използват в компонентата от високо равнище, за да декларират моделна величина, която има същата функция в компонентата от по-високо равнище, която има в основната компонента. Тогава те могат да получат достъп отвън чрез връзки съответни на концепцията на стъклената кутия.

По същото време изходните еквиваленти (като входни връзки) описват коя моделна величина от подчинената компонента ще се използва.

Компоненти от високо равнище в модела CedarBog

HIGH LEVEL COMPONENT Lake2

SUBCOMPONENTS

Layer2_1 OF CLASS Layer2,

Layer2_2 OF CLASS Layer2,

Layer2_3 OF CLASS Layer2

INPUT CONNECTIONS

sun --> (Layer2_1.sun, Layer2_2.sun,
Layer2_3.sun);

Компоненти от високо равнище в модела CedarBog

OUTPUT EQUIVALENCES

o1 := Layer2_1.o;

e1 := Layer2_1.e;

o2 := Layer2_2.o;

e2 := Layer2_2.e;

o3 := Layer2_3.o;

e3 := Layer2_3.e;

INITIALIZE

Layer2_1.f := 1;

Layer2_2.f := 0.8;

Layer2_3.f := 0.6;

END OF Lake2

Компоненти от високо равнище в модела CedarBog

Компонентите Sun2, Organic2 и Environ2 са на същото равнище от йерархията като компонентата Lake2. Те са основни компоненти.

Четири компоненти Sun1, Lake2, Organic2 и Environ2 трябва да са свързани на най-високото равнище на абстракция в компонентата CedarBog_High2.

Компоненти от високо равнище в модела CedarBog

BASIC COMPONENT Sun2

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

CONSTANTS

Pi(REAL) := 3.14

Компоненти от високо равнище в модела CedarBog

DEPENDENT VARIABLES

CONTINUOUS

$\text{suns (REAL[kJ/m}^2\text{])} := 0 \text{ [kJ/m}^2\text{]}$

DYNAMIC BEHAVIOUR

$\text{suns} := 95.9 \text{ [kJ/m}^2\text{]} * (1 + 0.635 * \text{SIN} (2[1/a] * \text{Pi} * T));$

END OF Sun2

Компоненти от високо равнище в модела CedarBog

BASIC COMPONENT Organic2

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES

CONTINUOUS

o (REAL[t]) := 0 [t]

Компоненти от високо равнище в модела CedarBog

SENSOR VARIABLES

CONTINUOUS

o1 (REAL[t]),

o2 (REAL[t]),

o3 (REAL[t])

DYNAMIC BEHAVIOUR

$o := o1 + o2 + o3;$

END OF Organic2

Компоненти от високо равнище в модела CedarBog

BASIC COMPONENT Environ2

USE OF UNITS

TIMEUNIT = [a]

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES

CONTINUOUS

$e \text{ (REAL[t])} := 0 \text{ [t]}$

Компоненти от високо равнище в модела CedarBog

SENSOR VARIABLES

CONTINUOUS

$e1 \text{ (REAL[t])} := 0 \text{ [t]},$

$e2 \text{ (REAL[t])} := 0 \text{ [t]},$

$e3 \text{ (REAL[t])} := 0 \text{ [t]}$

DYNAMIC BEHAVIOUR

$e := e1 + e2 + e3;$

END OF Environ2

Компоненти от високо равнище в модела CedarBog

HIGH LEVEL COMPONENT CedarBog_High2

SUBCOMPONENTS

Environ2,

Organic2,

Lake2,

Sun2

Компоненти от високо равнище в модела CedarBog

COMPONENT CONNECTIONS

Lake2.e1 --> Environ2.e1;

Lake2.e2 --> Environ2.e2;

Lake2.e3 --> Environ2.e3;

Sun2.suns --> Lake2.sun;

Lake2.o1 --> Organic2.o1;

Lake2.o2 --> Organic2.o2;

Lake2.o3 --> Organic2.o3;

END OF CedarBog_High2

Компоненти от високо равнище в модела CedarBog

Пълният синтаксис за компоненти от високо равнище
има вида:

high_level_component ::= HIGH LEVEL COMPONENT
 identifier

[dimension_constant_definition_part]

subcomponent_declaration

[input_connection_part]

[output_equivalence_part]

[component_connection_part]

[initialization_part]

END OF identifier

Компоненти от високо равнище в модела CedarBog

Описанието на езика дава следното за

input_connection_part:

input_connection_part ::= INPUT CONNECTION [S]

input_connection { input_connection }

Всички input_connections са последвани от ключовите думи INPUT CONNECTION или алтернативно INPUT CONNECTIONS.

Input_connections имат следната форма:

input_connection ::= identifier '-->' input_ident ';

| identifier '-->'

'(' input_ident { ',' input_ident } ')' ';

Компоненти от високо равнище в модела CedarBog

Тя свързва името на моделната величина с `input_ident` чрез символа `-->`.

`input_ident` се дава чрез:

`input_ident ::= indexed_identifier '.' identifier`

Може да се види, че са позволени само пътища с две равнища.

Същото е вярно и за изходните еквиваленти и инициализацията.

В частност е възможно директно да се инициализира само от едно равнище на йерархията към следващо. Следният израз не е позволен:

`INITIALIZE`

`Lake2.Layer2_1.f := 1;`

Компоненти от високо равнище в модела CedarBog

Ако константата f в компонентата CedarBog_High2 се инициализира, тя първо трябва да се направи достъпна чрез изходен еквивалентен израз в компонента Lake2.

OUTPUT EQUIVALENCE

$f1 := \text{Layer2_1.f};$

$f1$ е нормална моделна величина в компонента Lake2, на която може да се присвои стойност в следващата повисока компонента CedarBog_High2 чрез следния израз:

INITIALIZE

$\text{Lake.f1} := 1;$

Компоненти от високо равнище в модела CedarBog

За да бъде възможен достъп до моделна величина от подчинени компоненти, в наблюдателя са позволени по-дълги пътища в прозореца за параметрите. Ако например променливата на състоянието `p` от компонента `Layer2_3` трябва да се запише от наблюдател `OLayer2_3` (наблюдател за променливата на състоянието `Layer2_3`), верният път е:

`CedarBog_High2/Lake2/Layer2_3/p`

Компоненти от високо равнище в модела CedarBog

Забележки:

- Компонентите се обработват от стартиращи контроли в реда, в който са декларирани в компонента от високо равнище. Наредбата няма влияние върху симулационния резултат.
- Когато се компилира йерархичен модел, потребителят трябва да се уверим, че сме избрали правилната текуща версия за всички подчинени компоненти.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Моделите Queue и Transport се състоят от индивидуални секции, които се комбинират от основна компонента. Тези секции могат да се изнесат в независими компоненти, които се свързват, за да формират цялостен модел.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Разделяне на модел Queue.

Моделът Queue се състои от източник, опашка, сървър и контейнер. Източникът, сървърът с опашка и контейнера ще бъдат конвертирани в основни компоненти.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

HIGH LEVEL COMPONENT Queue5

SUBCOMPONENTS

Source5,

Station5,

Sink5

COMPONENT CONNECTION

Source5.OutputP --> Station5.WaitP;

Station5.OutputP --> Sink5.WaitP;

END OF Queue5

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

В компонентата Source5 новосъздадените работи се събират в регион OutputP. В компонентата от високо равнище Queue5 този регион се свързва с региона WaitP в компонента Station5. Региона WaitP се декларира като сензорна променлива в компонента Station5. Работите се изтриват оттук както се изисква и се преместват в региона Station.

Няма разлика между GET и SEND командите в основна компонента.

Могат да се променят региони, но не и сензорни региони. Сензорните региони не могат да стоят отляво на ADD, GET, SEND, REMOVE или PLACE команда.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Source5

MOBILE SUBCOMPONENTS OF CLASS Customer5

DECLARATION OF ELEMENTS

STATE VARIABLES

DISCRETE

TArrive (REAL) := 0,

Protocol (LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

RANDOM VARIABLES

Arrive (REAL) : EXPO

(Mean:=15,LowLimit:=0.5,UpLimit:=75)

LOCATIONS

OutputP (Customer5) := 0 Customer5

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

Създаване на задача

WHENEVER $T \geq T_{Arrive}$

DO

OutputP[^] : ADD 1 NEW Customer5;

$T_{Arrive}^{\wedge} := T + Arrive;$

IF Protocol

DO DISPLAY ("T= %f New customer \n",T); END

END

END OF Source5

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Station е деклариран като регион. На него е присвоена подвижна компонента от клас Customer5. На следващия цикъл работата ще се намира в Station. От дясната страна, мястото, от което е преместена подвижната компонента е сензорен регион.

GET и SEND трябва да се изберат подходящо, за да осигурят, че лявата страна на присвояването съдържа региони.

За да се постигне това временния регион WaitP се въвежда в спад. Това се използва само за прехвърляне на подвижни компоненти в региона Sink. Тогава те могат да се унищожат.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Station5

MOBILE SUBCOMPONENTS OF CLASS Customer5

DECLARATION OF ELEMENTS

STATE VARIABLES

DISCRETE

TWork (REAL) := 0,

Protocol (LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

RANDOM VARIABLES

Work (REAL) : EXPO
(Mean:=10,LowLimit:=0.32,UpLimit:=50)

LOCATIONS

Station (Customer5) := 0 Customer5,
OutputP (Customer5) := 0 Customer5

SENSOR LOCATION

WaitP (Customer5)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

Начало на обработката

WHENEVER (NUMBER(Station)=0) AND
(NUMBER(WaitP)>0)

DO Station[^] : FROM WaitP GET Customer5[1];

TWork[^] := T + Work;

IF Protocol

DO DISPLAY ("T= %f Cust. enters Station \n",T);

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Край на обработката

WHENEVER (T >= TWork) AND
(NUMBER(Station)=1)

DO

Station^ : TO OutputP SEND Customer5[1];

IF Protocol

DO DISPLAY ("T= %f Cust. leaves Station \n",T);

END

END

END OF Station5

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Sink5

MOBILE SUBCOMPONENTS OF CLASS
Customer5

DECLARATION OF ELEMENTS

STATE VARIABLES

DISCRETE

Protocol (LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

LOCATIONS

Sink (Customer5) := 0 Customer5

SENSOR LOCATION

WaitP (Customer5)

DYNAMIC BEHAVIOUR

Преместване на задачи към контейнера

WHENEVER NUMBER(WaitP) >= 1

DO

Sink[^] : FROM WaitP GET Customer5[1];

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Унищожаване на задачи

WHENEVER NUMBER(Sink) >= 4

DO

Sink^ : REMOVE Customer5{ ALL};

IF Protocol

DO DISPLAY("T= %f 4 Customers destroyed \n",
T); END

END

END OF Sink5

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Всички преходи на състоянията са начално в една основна компонента. Добра идея е да се направят индивидуални сегменти като WaitP или Station1 в отделни компоненти, които са свързвани заедно в компонента от високо равнище Trans2.

Поради тяхната подобност, всяка от компонентите QTaxi1, QTaxi2 и Road1, Road2 се разглежда като членове на класове съответно QTaxi_2 и Road_2.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

HIGH LEVEL COMPONENT Trans2

SUBCOMPONENTS

WaitP2,

QTaxi1 OF CLASS QTaxi_2,

QTaxi2 OF CLASS QTaxi_2,

Road1 OF CLASS Road_2,

Road2 OF CLASS Road_2,

Station1_2,

Station2_2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

COMPONENT CONNECTIONS

Региони

QTaxi1.LocQTaxi --> Station1_2.LocQTaxi;

WaitP2.LocWaitP --> Station1_2.LocWaitP;

Station1_2.LocSta1 --> Road1.LocSta;

Road1.LocRoad --> QTaxi2.LocRoad;

QTaxi2.LocQTaxi --> Station2_2.LocQTaxi;

Station2_2.LocSta2 --> Road2.LocSta;

Road2.LocRoad --> QTaxi1.LocRoad;

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Променливи на състоянието

Station1_2.TLoad --> Road1.TLoad;

Station2_2.TUnload --> Road2.TLoad;

INITIALIZE

Road1.RoadN := 1;

Road2.RoadN := 2;

QTaxi1.QTaxiN := 1;

QTaxi2.QTaxiN := 2;

END OF Trans2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Аналогично се извършва инициализацията с 10 таксите на Trans1 в компонента QTaxi1. Инициализацията се извършва след създаването на експеримента и първото изпълнение чрез избирането на Break0. Тогава може да се използва командата Model Parameters в контекстното меню.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

В прозореца на измененията QTaxi1 може да се разшири и тогава да се избере региона LocQTaxi. След натискане на бутона Change се появява входно поле, в което на броя на подвижните компоненти може да се даде стойност 10. В дясната част на прозореца на измененията се показва командата

AddMobile/Trans2/QTaxi1/LocQTaxi 10

Промяната може да се провери чрез проверка на стойностите за следващия сегмент на следващото изпълнение. Това се постига чрез избиране на Break0 и изпълняване на командата Show Initial State.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT QTaxi_2

MOBILE SUBCOMPONENTS OF CLASS Taxi2

DECLARATION OF ELEMENTS

STATE VARIABLES

QTaxiN(INTEGER) := 1, # Номер на QTaxi

Protocol(LOGICAL) := FALSE, # Контрол на
протокола

TaxiOu(INTEGER) := 0 # Такси излиза от
маршрут

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

TRANSITION INDICATORS

Print

LOCATIONS

LocQTaxi (Taxi2) := 0 Taxi2

SENSOR LOCATIONS

LocRoad(Taxi2)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

ON START

DO

IF $Q_{TaxiN} = 1$

DO

$LocQ_{Taxi}^{\wedge}$: FROM $LocQ_{Taxi}$ GET $Taxi2\{ALL\ i\}$

CHANGING

$TaxiN^{\wedge} := i;$

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

IF Protocol

DO

DISPLAY(" T= %f Component QTaxi%ld \n", T,
QTaxiN);

DISPLAY(" Numbering of Taxis \n\n");

END

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

```
IF NUMBER(LocRoad) > 0
```

```
DO
```

```
    WHENEVER T >= LocRoad:Taxi2[1].TTravel
```

```
    DO
```

```
        LocQTaxi^ : FROM LocRoad GET Taxi2[1];
```

```
        TaxiOu^ := LocRoad:Taxi2[1].TaxiN;
```

```
        SIGNAL Print;
```

```
    END
```

```
END
```

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Изходна функция

ON Print

DO

IF Protocol

DO

DISPLAY(" T= %f Component QTaxi%ld \n", T,
QTaxiN);

DISPLAY(" Taxi Number %ld enters QTaxi%ld \n\n",
TaxiOu, QTaxiN);

END

END

END OF QTaxi_2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT WaitP2

MOBILE SUBCOMPONENTS OF CLASS

Customer2

DECLARATION OF ELEMENTS

STATE VARIABLES

TNext(REAL) := 0

Protocol(LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

RANDOM VARIABLES

ZArrive(REAL) :

EXPO(Mean:=10,LowLimit:=0.32,UpLimit:=50)

TRANSITION INDICATORS

Print

LOCATIONS

LocWaitP(Customer2) := 0 Customer2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

WHENEVER $T \geq T_{Next}$

DO

LocWaitP[^] : ADD 1 NEW Customer2;

$T_{Next}^{\wedge} := T + Z_{Arrive};$

SIGNAL Print;

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print

DO IF Protocol

DO

DISPLAY(" T= %f Component WaitP \n",T);

DISPLAY(" Cust. enters WaitP \n");

DISPLAY(" Next generation TNext = %f \n",TNext);

DISPLAY(" Number of cust. in WaitP2 = %ld
\n\n",NUMBER(LocWaitP));

END

END END OF WaitP2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Station1_2

MOBILE SUBCOMPONENTS OF CLASS Taxi2, Customer2

DECLARATION OF ELEMENTS

CONSTANTS

Load(REAL) := 3

STATE VARIABLES

TLoad (REAL) := 0,

Protocol(LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

TRANSITION INDICATORS

Print1,

Print2

LOCATIONS

LocSta1(Taxi2) := 0 Taxi2

SENSOR LOCATIONS

LocWaitP(Customer2),

LocQTaxi(Taxi2)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

WHENEVER (NUMBER(LocWaitP) > 0) AND
(NUMBER(LocQTaxi) > 0) AND
(NUMBER(LocSta1) = 0)

DO

LocSta1^ : FROM LocQTaxi GET Taxi2[1];
SIGNAL Print1;

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

IF NUMBER(LocSta1) = 1

DO

WHENEVER NUMBER(LocSta1:Taxi2[1].Seat) = 0

DO

LocSta1:Taxi2[1].Seat[^] : FROM LocWaitP GET

Customer2{ ALL i | i ≤ 4 };

TLoad[^] := T + Load;

SIGNAL Print2;

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print1

DO

IF Protocol

DO

DISPLAY(" T= %f Component Station1 \n",T);

DISPLAY(" Taxi Number %ld enters Station1 \n\n",
LocSta1:Taxi2[1].TaxiN);

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print2

DO

IF Protocol

DO

DISPLAY(" T= %f Component Station1 \n",T);

DISPLAY(" Taxi Number %ld, Num. cust. %ld \n",

LocSta1:Taxi2[1].TaxiN,

NUMBER(LocSta1:Taxi2[1].Seat));

DISPLAY(" End loading in T= %f \n\n",TLoad);

END

END

END OF Station1_2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Station2_2

MOBILE SUBCOMPONENTS OF CLASS Taxi2, Customer2

DECLARATION OF ELEMENTS

CONSTANT

Unload(REAL) := 2

STATE VARIABLES

TUnload (REAL) := 0,

Protocol(LOGICAL) := FALSE,

CustomN (INTEGER) := 0

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

TRANSITION INDICATORS

Print1,

Print2

LOCATIONS

LocSta2(Taxi2) := 0 Taxi2

SENSOR LOCATIONS

LocQTaxi(Taxi2)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

WHENEVER (NUMBER(LocSta2) = 0) AND
(NUMBER(LocQTaxi) > 0)

DO

LocSta2^ : FROM LocQTaxi GET Taxi2[1];

SIGNAL Print1;

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

IF NUMBER(LocSta2) =1

DO

WHENEVER NUMBER(LocSta2:Taxi2[1].Seat) >0

DO

LocSta2:Taxi2[1].Seat^ : REMOVE Customer2{ALL};

TUnload^ := T + Unload;

CustomN^ := NUMBER(LocSta2:Taxi2[1].Seat);

SIGNAL Print2;

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print1

DO

IF Protocol

DO

DISPLAY(" T= %f Component Station2 \n",T);

DISPLAY(" Taxi number %ld enters Station2 \n\n",
LocSta2:Taxi2[1].TaxiN);

END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print2

DO

IF Protocol

DO

DISPLAY(" T= %f Component Station2 \n",T);

DISPLAY(" Taxi number %ld num. cust. %ld \n",
LocSta2:Taxi2[1].TaxiN,CustomN);

DISPLAY(" Unloading ends in %f \n\n",TUnload);

END

END END OF Station2_2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Road_2

MOBILE SUBCOMPONENTS OF CLASS Taxi2

DECLARATION OF ELEMENTS

STATE VARIABLES

DISCRETE

RoadN (INTEGER) := 1,

Protocol(LOGICAL) := FALSE,

TaxiIn (INTEGER) := 0

CONTINUOUS

TaxiTT(REAL) := 0

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

SENSOR VARIABLES

TLoad(REAL) := 0

RANDOM VARIABLES

ZTravel(REAL)

:GAUSS(Mean:=36,Sigma:=10,LowLimit:=0,UpLimit:=72)

TRANSITION INDICATORS

Print

LOCATIONS

LocRoad(Taxi2 ORDERED BY INC TTravel) := 0 Taxi2

SENSOR LOCATIONS

LocSta(Taxi2)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

ON $T \geq TLoad$

DO $LocRoad^{\wedge} : FROM LocSta GET Taxi2[1]$

CHANGING

$TTravel^{\wedge} := T + ZTravel;$

END

$TaxiIn^{\wedge} := LocSta:Taxi2[1].TaxiN;$

$TaxiTT^{\wedge} := T + ZTravel;$

SIGNAL Print;

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

ON Print

DO

IF Protocol

DO

DISPLAY(" T= %f Component Road%ld \n", T, RoadN);

DISPLAY(" Taxi number %ld enters Road%ld \n",
TaxiIn, RoadN);

DISPLAY(" End travelling in T= %f \n\n", TaxiTT);

END

END END OF Road_2

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Масиви от компоненти.

Разглеждаме модела Queue5.

Вместо само един сървър сега моделът ще използва последователност от три сървъра.

Начално можем да използваме концепцията на класа и да декларираме различни инстанции на клас Station. Компонентата Station представя описанието на класа.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

Моделът може да се подобри от един от тези три сървъра чрез заместване на компонента от високо равнище Queue5 с компонента от високо равнище Queue6.

Тъй като компонентата Station се среща няколко пъти в модела, извежданията не могат да се асоциират уникално с всеки един сървър, когато протоколът е включен (Protocol = TRUE). Информацията може да се идентифицира чрез даване на всеки сървър на уникален номер StationN. Инициализацията се осъществява в компонента от високо равнище Queue6.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

HIGH LEVEL COMPONENT Queue6

SUBCOMPONENTS

Source6,

Station_1 OF CLASS Station6,

Station_2 OF CLASS Station6,

Station_3 OF CLASS Station6,

Sink6

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

COMPONENT CONNECTION

Source6.OutputP --> Station_1.WaitP;

Station_1.OutputP --> Station_2.WaitP;

Station_2.OutputP --> Station_3.WaitP;

Station_3.OutputP --> Sink6.WaitP;

INITIALIZE

Station_1.StationN := 1;

Station_2.StationN := 2;

Station_3.StationN := 3;

END OF Queue6

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

BASIC COMPONENT Station6

MOBILE SUBCOMPONENTS OF CLASS

Customer6

DECLARATION OF ELEMENTS

STATE VARIABLES

DISCRETE

TWork (REAL) := 0,

StationN (INTEGER) := 0,

Protocol (LOGICAL) := FALSE

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

RANDOM VARIABLES

Work (REAL) : EXPO (Mean :=
10, LowLimit:=0.32, UpLimit:=50)

LOCATIONS

Station (Customer6) := 0 Customer6,
OutputP (Customer6) := 0 Customer6

SENSOR LOCATION

WaitP (Customer6)

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

DYNAMIC BEHAVIOUR

WHENEVER (NUMBER(Station)=0) AND
(NUMBER(WaitP)>0)

DO

Station[^] : FROM WaitP GET Customer6[1];

TWork[^] := T + Work;

IF Protocol

DO DISPLAY ("T= %f Cust. enters Station%ld \n",T,
StationN); END

END

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

WHENEVER (T >= TWork) AND
(NUMBER(Station)=1)

DO

Station^ : TO OutputP SEND Customer6[1];

IF Protocol

DO DISPLAY ("T= %f Cust. leaves Station%ld
\n",T, StationN); END

END

END OF Station6

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

По-нататъшно опростяване може да се направи чрез използване на възможностите SIMPLEX MDL, за да се декларират масиви от компоненти.

Позволени са всички операции. Това дава възможност за много ясна и сбита бележка за големи модели с много връзки между компонентите на един и същ клас.

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

HIGH LEVEL COMPONENT Queue7

SUBCOMPONENTS

Source7,

ARRAY [3] Station7,

Sink7

РАЗДЕЛЯНЕ НА МОДЕЛИ С РЕГИОНИ

COMPONENT CONNECTION

Source7.OutputP --> Station7[1].WaitP;

Station7{i OF 1..2}.OutputP -->
Station7[i+1].WaitP;

Station7[3].OutputP --> Sink7.WaitP;

INITIALIZE

Station7{ ALL i }.StationN := i;

END OF Queue7

Състояния и преходи между състоянията в йерархични модели

С добавянето на подкомпоненти трябва да се прибавят по-нататъшните състояния.

Разглеждаме йерархичен модел с подкомпоненти и компоненти от високо равнище, които ще бъдат използвани, за да илюстрират състояния и преходите между състоянията. На най-ниското равнище са основните компоненти A1, A2, B1, B2 и C. Компонентите B1, B2 и C са част от компонента D на следващото по-високо равнище. Накрая компонентите A1, A2 и D се свързват, за да формират компонента E.

Компонента D се състои от подкомпоненти B1, B2 и C. Сама по себе си D е подкомпонента на E. От гледна точка на D, E е подкомпонента.

Състояния и преходи между състоянията в йерархични модели

Състояния, използвани в управлението на моделната библиотека.

- Непроверена.

Компонента има такова състояние веднага след като се създаде. Това може да се случи или чрез редактиране (текстово или графично) или чрез операция за копиране. Също в това състояние влизат версиите, чийто MDL код се променя и чийто синтаксис и интерфейси все още не са проверени за вярност.

- Синтаксис Наред.

В това средно състояние синтаксиса на компонента е бил променен, но не и съвместимостта на интерфейсите към подкомпонентите.

Състояния и преходи между състоянията в йерархични модели

- Проверена.

MDL компилаторът е превел компонента в C код успешно.

- Подготвена.

Тук се компилира C кода в обектен код.

Състояния и преходи между състоянията в йерархични модели

Компонента влиза в състояние Синтаксис Наред, когато е сигурно, че описанието на модела в **SIMPLEX MDL** е вярно. Проверката дали подчинени компоненти са съвместими с компоненти от високо равнище вече не се изпълнява.

Състояния и преходи между състоянията в йерархични модели

Пример:

Компонентите от високо равнище могат например да опишат две променливи в различни компоненти, които ще се свързват. Ако описанието е синтактично вярно, компонентата може да влезе в състояние Синтаксис наред. В това състояние не е сигурно, че и двете променливи са декларирани в компоненти от ниско равнище, тъй като няма извършена проверка за съвместимост.

Само компоненти от високо равнище могат да бъдат в състояние Синтаксис наред. Основните компоненти на най-ниското равнище на йерархията не могат да бъдат в това състояние.

Състояния и преходи между състоянията в йерархични модели

1. Компонента в състояние Непроверена се премества в състояние Синтаксис Наред чрез MDL компилатор, когато се определи, че синтаксиса му е правилен.
2. Преходът от състояние Синтаксис Наред към Проверена се извършва, когато MDL компилатора успешно провери съвместимостта на компонентата.
3. Компонента е в състояние подготвена, когато C кода е компилиран успешно.
4. Когато MDL кода на всяка компонента се промени, тя се връща в състояние Непроверена.

Състояния и преходи между състоянията в йерархични модели

5. Ако се направят по-нататъшни промени надолу в йерархията на симулационния модел вместо на MDL кода на самата компонента, тогава всички компоненти от по-високо равнище, които са били поне в състояние Проверена, се връщат към състояние Синтаксис наред. Това е следствие от факта, че въпреки че, MDL кода на компонента от високо равнище е все още синтактично коректен, измененията на подчинените компоненти могат да причинят интерфейсите вече да не са верни. Например променлива, която се нуждае от компоненти от високо равнище може да се преименува или изтрива в подчинена компонента.

Състояния и преходи между състоянията в йерархични модели

Забележки:

- Системата на моделна библиотека съхранява и управлява файлове, отговарящи на всички предишни състояния на компонента в по-напреднало състояние.
- MDL версията на основна компонента, която е например в състояние Проверена, все още съществува. Командата Edit version... получава достъп до тази MDL версия. Тази команда става налична, когато се избере компонентна версия.
- Когато се измени компонента чрез командата Edit version..., компонентата влиза в състояние Неопределена. Файлът, съдържащ C версията на компонента (в състояние Проверена) се изтрива.

Състояния и преходи между състоянията в йерархични модели

Процесът на превод, и на преходи между състоянията са засегват от потребителските команди New component, Copy, Edit version..., Check version и Current version.

- New component – нова компонента

Създава нова компонента. Компонентата е в състояние Непроверена.

- Copy – копиране

Компонентата е в състояние Непроверена.

- Edit version... - редактиране на версия...

Сменя компонента.

Изменената компонента влиза в състояние Непроверена. Ако модела е йерархичен, управлението на моделната библиотека осигурява, че състоянието на всички компоненти от високо, равнище се променя на Синтаксис Наред. Всички компоненти, които са подкомпоненти на променената компонента запазват състоянието си.

Състояния и преходи между състоянията в йерархични модели

- Check version

Синтактична проверка и проверка за съвместимост .
Превеждане на С.

Командата Check version премества компонента от
състояние Неопределена в състояние Проверена.

Тогава основните компоненти имат проверен
синтаксис. Ако има синтактични грешки, се извежда
съобщение за грешка. Грешката може да се поправи с
командата Edit version... Когато синтаксисът е
правилен се извършва превод в С. Тогава
компонентата е в състояние Проверена.

Състояния и преходи между състоянията в йерархични модели

Ако командата Prepare version е приложена на компоненти от високо равнище, се извършват следните действия:

1. Синтактична проверка на компоненти от по-високо равнище и преход в С.

Състояние на компонентата Синтаксис Наред.

2. Превод на всички подчинени компоненти.

Тези компоненти преминават в състояние Проверени.

3. Проверка за съвместимост.

Премества компонентите от по-високо равнище в състояние Проверени. Ако проверката за съвместимост открива грешка, компонентата от по-високо равнище остава в състояние Синтаксис Наред.

Състояния и преходи между състоянията в йерархични модели

- Prepare version – подготви версия

Компилира до обектен код.

Командата Prepare version привежда компонентата от състояние Checked в състояние Prepared. Компонента и всичките му подкомпоненти се компилират в обектен код и преминават в състояние Подготвена.

Командата Prepare version съдържа извикване на командата Check version за всички компоненти, които все още не са в състояние Проверени. Така командата Prepare version прави възможен прехода от състояние Непроверена в състояние Подготвена. Извикването (ръчно) на Check version в такъв случай не е необходимо.

Състояния и преходи между състоянията в йерархични модели

– Current version – текуща версия

- Една версия се избира като текуща версия, която се използва в йерархичния модел.
- При йерархичните модели е важно да се осигури, че когато се представят няколко версии на подчинени компоненти, върнатата версия се избира за текуща.
- Истинското преимущество на концепцията на версията се изяснява, когато се използват йерархично структурирани модели. Управлението на моделната библиотека автоматично осигурява, че от всички възможни версии винаги се използва текущата. Не са необходими промени за описанието на модела, когато се промени версия.

Състояния и преходи между състоянията в йерархични модели

Следващата фигура показва компонента C в 2 различни версии. Командата Current version може да се използва, за да се специфицира текущата версия. Текущата версия е тази, която се интегрира в модела.

Фигурата показва също, че версиите могат да имат различни нива на сложност. Всички версии трябва да имат идентичен интерфейс към външния свят. Може да се види, че входната и изходната точка е една и съща, въпреки различната вътрешна структура.

Състояния и преходи между състоянията в йерархични модели

Концепцията на версията има две приложения:

- Могат да се изучат изменения на модела.
- Първо се изгражда проста версия (бърз прототип), която може да се замени по-късно от по-подробна версия.