



# MODELLALAPÚ SZOFTVERFEJLESZTÉS

## II. GYAKORLAT COMPILER AS A SERVICE PROJEKCIÓS EDITOROK

DR MEZEI GERGELY  
BALOGH ÁKOS

# A MAI GYAKORLAT

**I. Fejezet** Fordító, mint szervíz

**II. fejezet** Projekciós editorok



## BLACK BOX – WHITE BOX

- Hogyan működik egy szoftverfejlesztő eszköz? (pl. IntelliJ)
  - Szerkesztjük a kódot
    - Syntax highlight
    - Szintaktikai hibák
    - Kódkiegészítés
  - Futtatjuk a kódot

A fordítás folyamata egy fekete doboz

# BLACK BOX – WHITE BOX

- Compiler, mint fekete doboz
  - Mi történik ha hibás? (syntax highlight)
  - Hogyan terjeszthetjük ki?
  - Sok esetben elég, de...
    - A fejlesztőeszköz nem lát bele a fordítóba
    - AI-vezérelt kódkiegészítés? (pl. Github Copilot)

# BLACK BOX – WHITE BOX

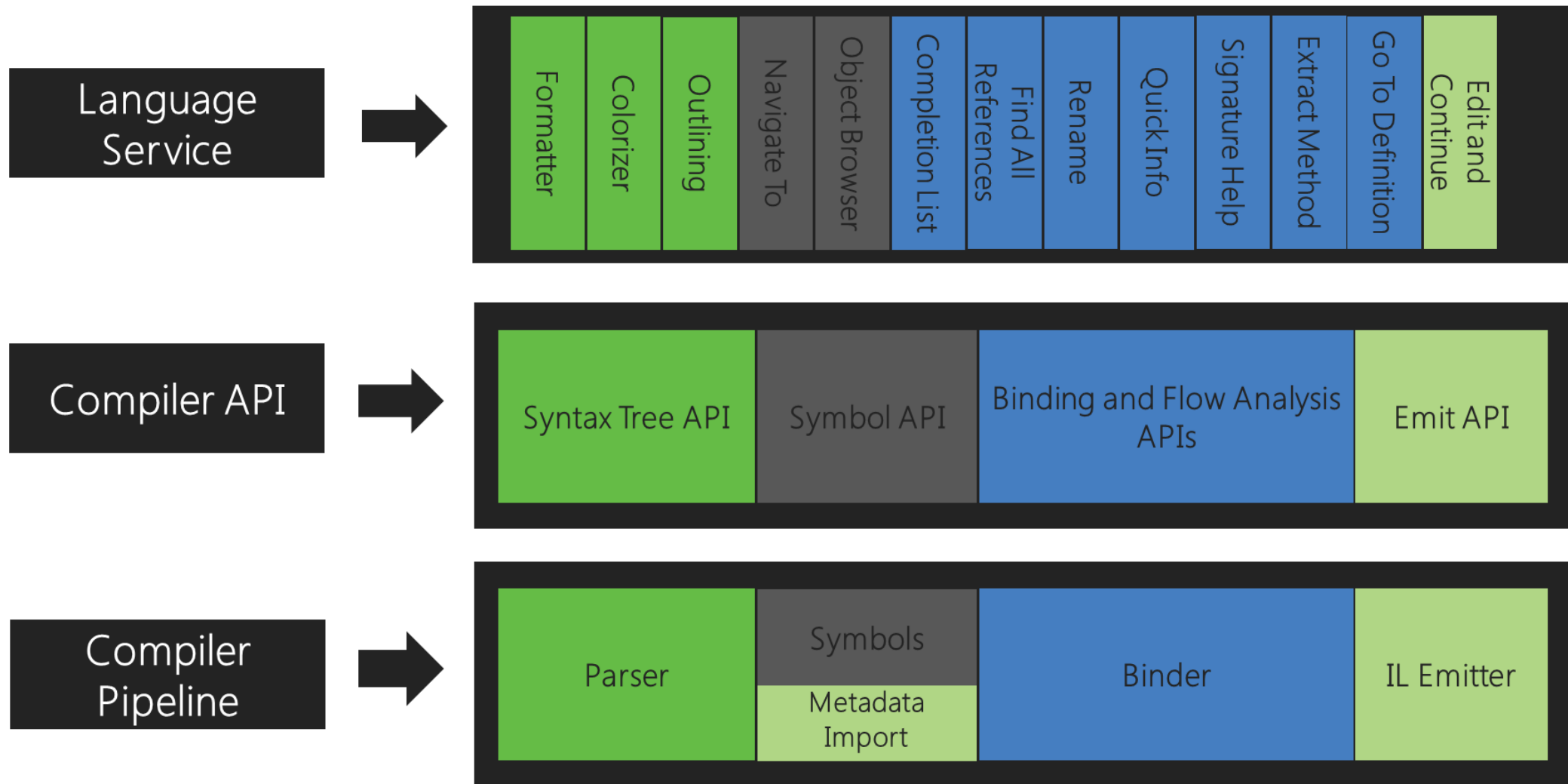
- Compiler – fehér dobozként?
- Roslyn – Compiler as a Service
  - Kódanalízálás
  - Kódfixálás
  - Refaktorálás
  - Kódgenerálás
  - Visual Studioba beépülve

```
static void Main(string[] args)
{
    int i = 1;
    int j = 2;
    int k = i + j;
}
```

# COMPILER AS A SERVICE

- CaaS – miért jó?
  - Saját (csapatszintű) kódkonvenciók követése
  - Könyvtárak javasolt használata
  - Segítség az általános konvenciókhoz, mintákhoz

# ROSLYN - FELÉPÍTÉS



# ROSLYN - FELÉPÍTÉS

- Compiler APIs
  - A fordítás lépései során előálló információk
- Diagnostic APIs
  - Kód diagnosztika, MSBuild és VS integráció
- Scripting APIs
  - Kódvégrehajtás
- Workspace APIs
  - Solution-szintű analízis és refaktorálás



# DEMO: SZINTAKTIKAI ELEMZÉS (SYNTAX ANALYZER)

The screenshot displays the Syntax Visualizer application, which is used for analyzing C# code. The interface is divided into several sections:

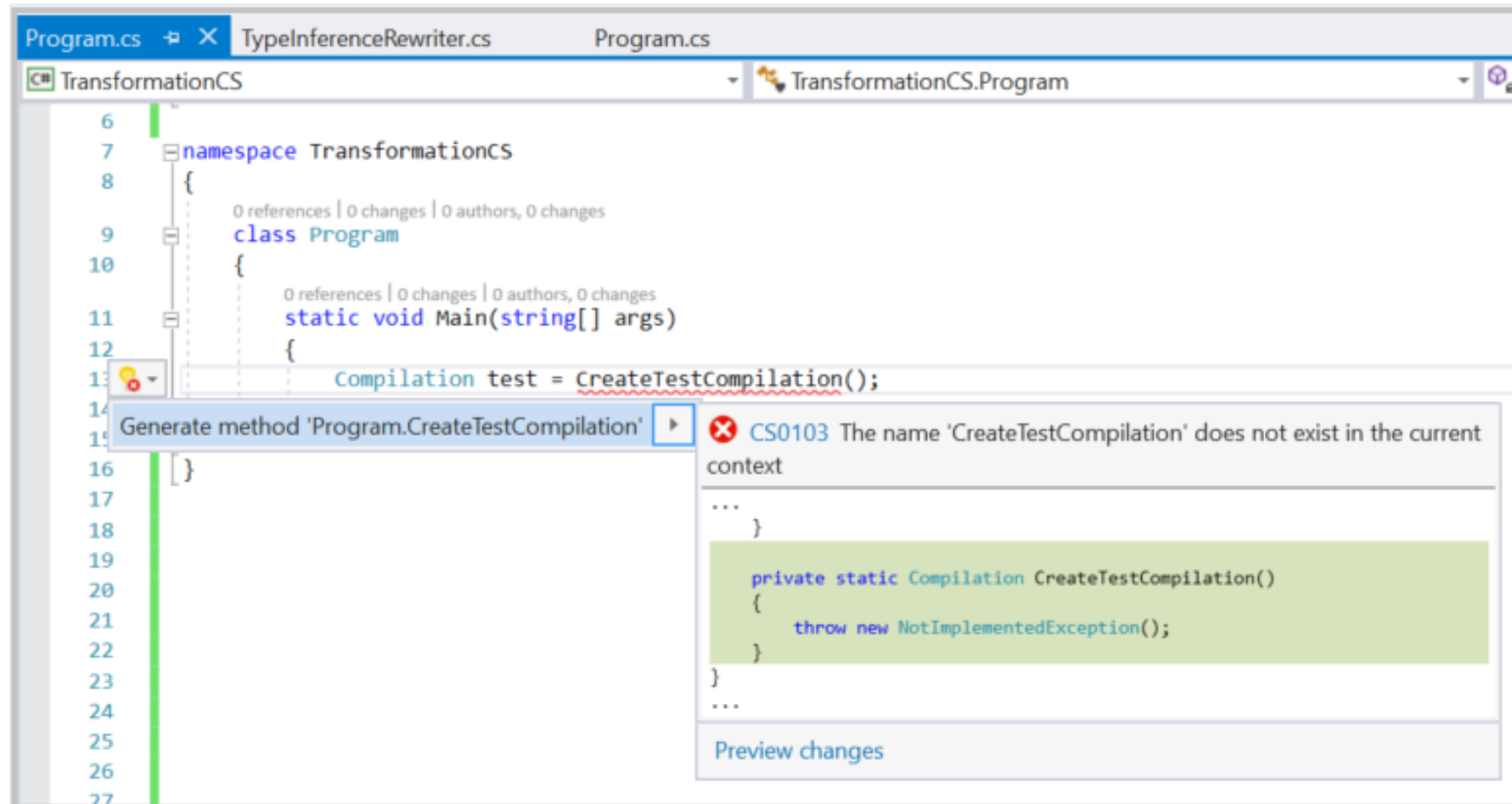
- Syntax Tree:** A tree view on the left showing the hierarchical structure of the code. The selected node is **MethodDeclaration** [78..179], which includes **ClassDeclaration** [48..186], **IdentifierToken** [54..61], **OpenBraceToken** [67..68], and **MethodDeclaration** [78..179].
- Properties:** A table on the left showing the properties of the selected node. The **Type** is **MethodDeclarationSyntax** and the **Kind** is **MethodDeclaration**. The **Body** property is expanded, showing the code snippet: `{ Console.WriteLine("Hello World!"); }`.
- Program.cs:** A code editor showing the source code of the **ConsoleApp12** program. The code is as follows:

```
6 {  
7     0 references  
8     static void Main(string[] args)  
9     {  
10        Console.WriteLine("Hello World!");  
11    }  
12 }  
13
```
- Syntax.dgml:** A diagram view at the bottom showing the syntax tree structure. The root node is **MethodDeclaration Node**, which branches into **Static Token**, **Void Token**, **Main Token**, **ParameterList Node**, and **Block Node**. The **Block Node** further branches into **ExpressionStatement Node** and **EndOfLineToken**. The **ExpressionStatement Node** branches into **InvocationExpression Node** and **EndOfLineToken**. The **InvocationExpression Node** branches into **SimpleMemberAccessExpression Node** and **ArgumentList Node**. The **SimpleMemberAccessExpression Node** branches into **IdentifierName Node** and **EndOfLineToken**. The **ArgumentList Node** branches into **Argument Node** and **EndOfLineToken**. The **Argument Node** branches into **StringLiteralToken** and **EndOfLineToken**.

## DEMO: SZEMANTIKAI ELEMZÉS (DATA FLOW, CONTROL FLOW)

- Felhasználhatóak a szemantikai elemzés eredményei
  - Adatfolyam
    - Milyen szimbólumok vannak definiálva?
    - Melyik szimbólum hol érhető el? Mit fed az adott név?
    - Használja-e bárki az adott szimbólumot (változót)?
  - Vezérlésfolyam
    - Milyen értékekkel térhet vissza egy adott metódus?

# DEMO: EDITOR BŐVÍTÉS (DIAGNOSTIC ANALYZER, CODE FIX PROVIDER)



# DEMO: FUTÁSIDEJŰ KÓDFUTTATÁS (SCRIPTING API)

```
PS C:\Users\Ali Bahraminezhad\source\repos\Scripter> |
```

I

<https://github.com/dotnet/roslyn/blob/main/docs/wiki/Scripting-API-Samples.md>

# A MAI GYAKORLAT

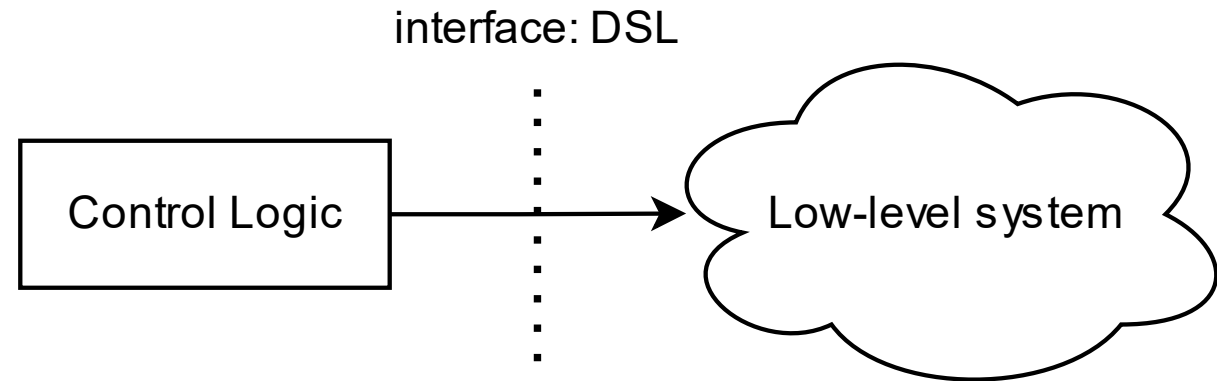
**I. Fejezet** Fordító, mint szervíz

**II. fejezet** Projekciós editorok



# SZAKTERÜLETI NYELV TERVEZÉSE

- „Ha lenne egy olyan nyelvem, amivel [...], akkor könnyebb lenne az életem.”
- Vezérlési logika:
  - Magas szintű
  - Jól elhatárolható
- Végrehajtás:
  - Alacsony szintű
  - Algoritmizálható



# SZAKTERÜLETI NYELV TERVEZÉSE

- Mi kell egy nyelvhez?

- Szakterületi fogalmak

- szimbólumok

*„ez a szótáram...”*

- szemantikai tartalom

*„de mit jelentenek a szavak?”*

- Szakterületi szabályok

- kapcsolódási szabályok

*„hogyan alkotok mondatot?”*

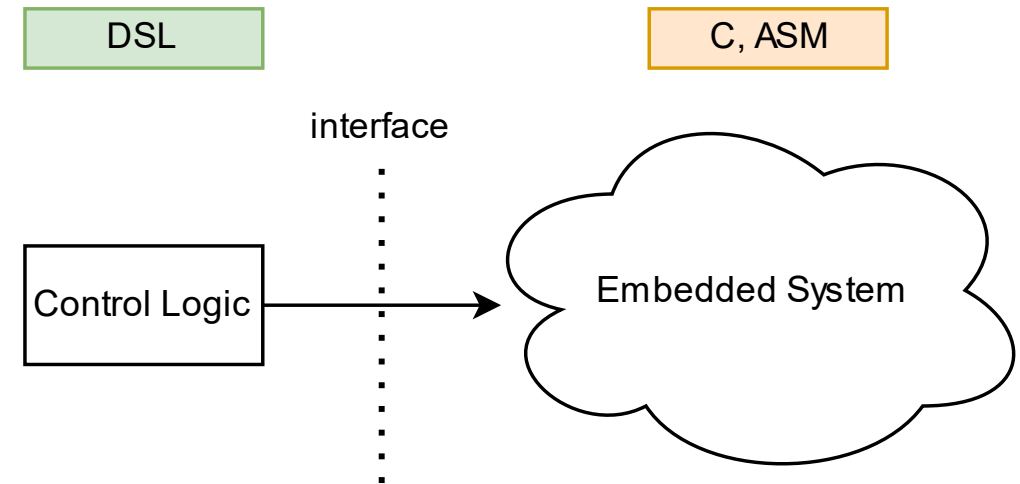
- Feldolgozási eljárás

*„hogyan értik majd meg?”*

- Gondolatkísérlet: rendszerleírás szóhísztoqram » gyakoriságból szótár 😊

# NYELVTERVEZÉS PÉLDA: BEÁGYAZOTT RENDSZER VEZÉRLÉSE

- Összetett folyamat végrehajtása
  - Milyen lépések vannak?
    - a magas szintű leírás hatékony
- Összetett beágyazott rendszer
  - Hogyan hajtom végre a lépéseket?
    - regiszter szintű vezérlés szükséges
- Megoldás: Adok egy DSL-t, amivel leírom a folyamatot, majd a leírásból generálok C kódot



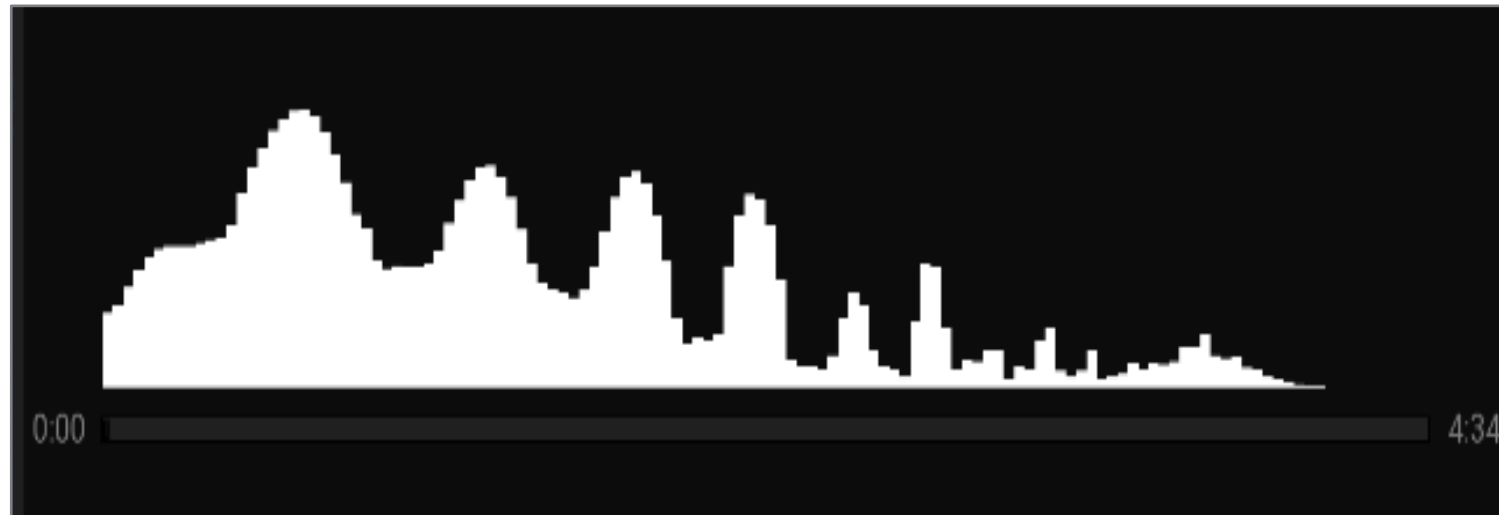


# NYELVTERVEZÉS – MUSIC VISUALISER (VSR)

- Ötlet: „Zenére ugráljanak a fények, és legyen szép”
- Leírás:
  - „Ha szól a hegedű, legyen egy fény kék.
  - Ha a gitár szól, akkor legyen minden piros.
  - Ha mindkettő szól, akkor legyen minden lila.”
- Célhardver: RGB LED szalag
  - WS2812B NeoPixel strip, 8MHz-es SPI buszon PWM négyszögjel kódol RGB biteket... ha a gitár szól, akkor legyen a Duty cycle 32%...? 😊

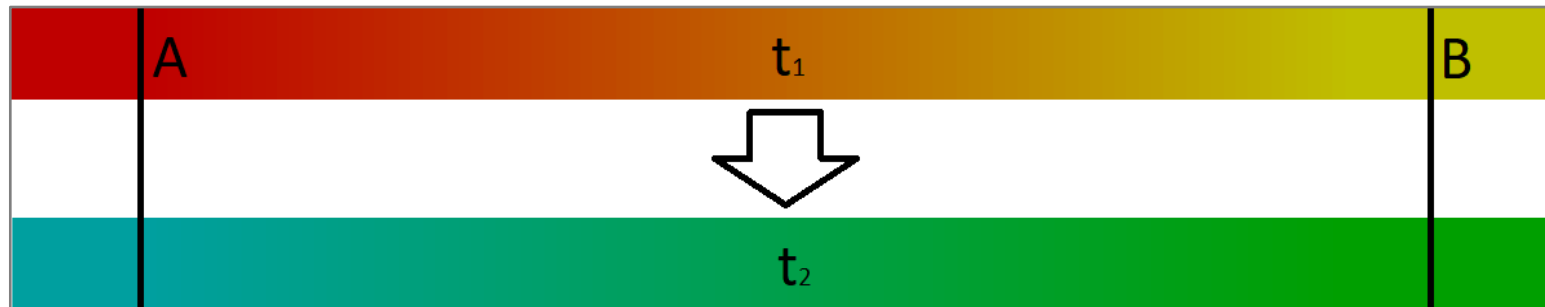
## VSR – MODELLEZŐ TÉR DEFINIÁLÁSA

- Ötlet: „*A zene spektruma modulálja a LED-ek fényerejét.*”
  - részleges automatizálás
  - szűkíti a nyelv modellező terét



## VSR – MODELLEZŐ TÉR DEFINIÁLÁSA

- Probléma: Minden időpillanatban definiálni kell az összes LED színét.
- Ötlet: „*A nyelv pillanatnyi állapotokat rögzítsen, köztük interpolálhatunk.*”
- » rövidíti a leírást
- » cserében komplexebb lesz a feldolgozás



*A, B rögzített LED-ek;  $t_1, t_2$  rögzített időpillanatok*

# VSR – ELŐZETES SPECIFIKÁCIÓ

- Ha lenne egy olyan nyelvem, ami alkalmas:
  - Egyes időpillanatokban...
  - egyes LED-ekhez...
  - egyes színt hozzárendelni.
- És lenne egy olyan interpreterem, ami:
  - Kiszámolja a zene a spektrumát...
  - Interpolálja a rögzített időpillanatokat...
- Akkor könnyen le tudnám írni egy zene vizualizációs logikáját. 😊

# VSR – MINIMÁLIS SZINTAXIS

- Szótár:

- Időpillanat, LED, szín

at 02:16

LED[12]      Magenta

- Feldolgozás:

LED[120]      Green

- Interpolálás, spektrum

at 03:00

- Szótár-feldolgozás interfész:

LED[0]      Black

- » Szavak kapcsolódási szabályai

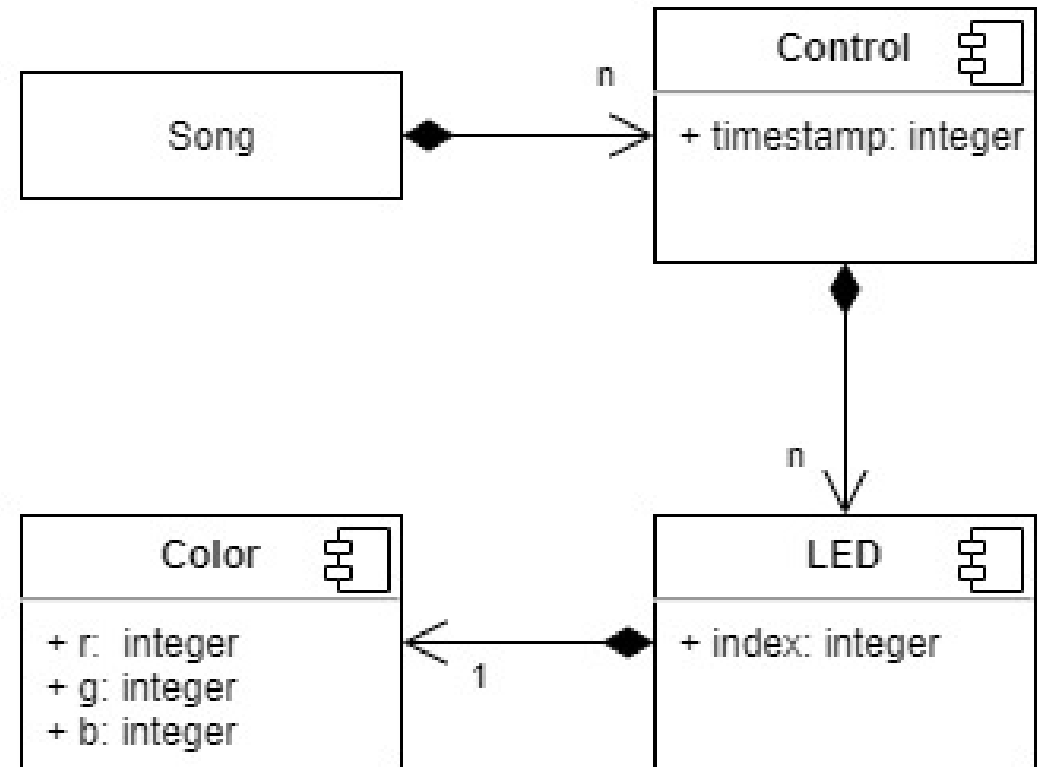
LED[67]      White

- Segítség:

- „*Hogyan lenne kényelmes...?*”

# VSR - SZÓTÁR

- DSL-interpreter interfész
  - szemantika
- OOP megközelítés
  - Szavak
    - objektumok
  - Nyelvtan
    - kompozíció



# VSR - MEGVALÓSÍTÁS

- Első gondolat:
  - Saját szövegszerkesztő, syntax highlight, compiler ☹️
- Lehet máshogy is:
  - JetBrains MetaProgramming System 😊
  - » minden nyelvhez integrált szerkesztőfelület
  - » speciális szerkesztő: nincs szintaktikai hibalehetőség
  - » minden nyelvhez integrált fordító
  - » MPS: nyelvtervező szakterületi nyelvek sokasága 😊

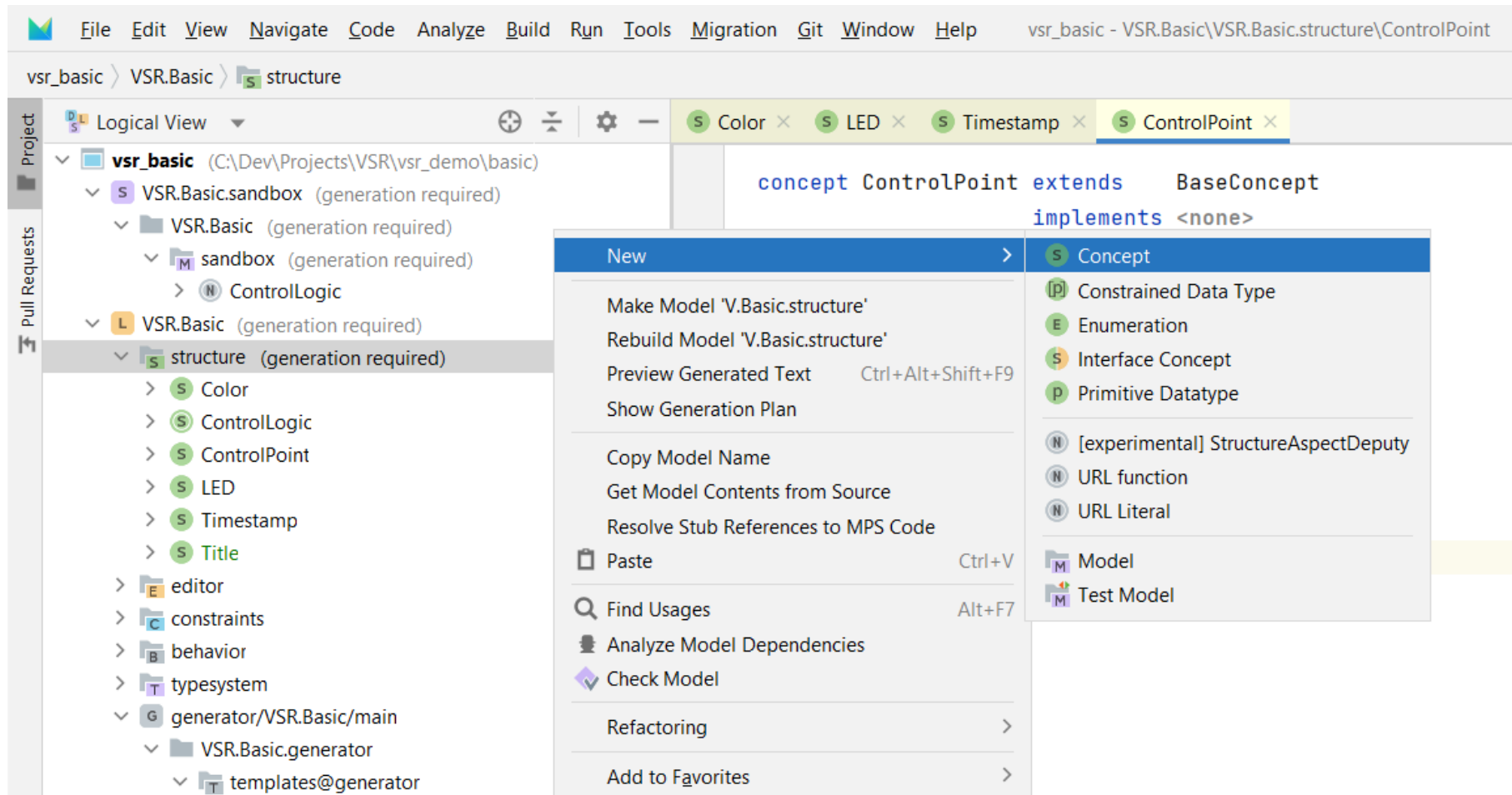
# MPS BEVEZETÉS: OOP

## ■ MPS terminológia:

- Objektumok » Concepts
- Attribútumok » Properties
- Kompozíció » Children
- Interface » Projectional editor
  
- Fordító » Generator
- Munkaterület » Sandbox

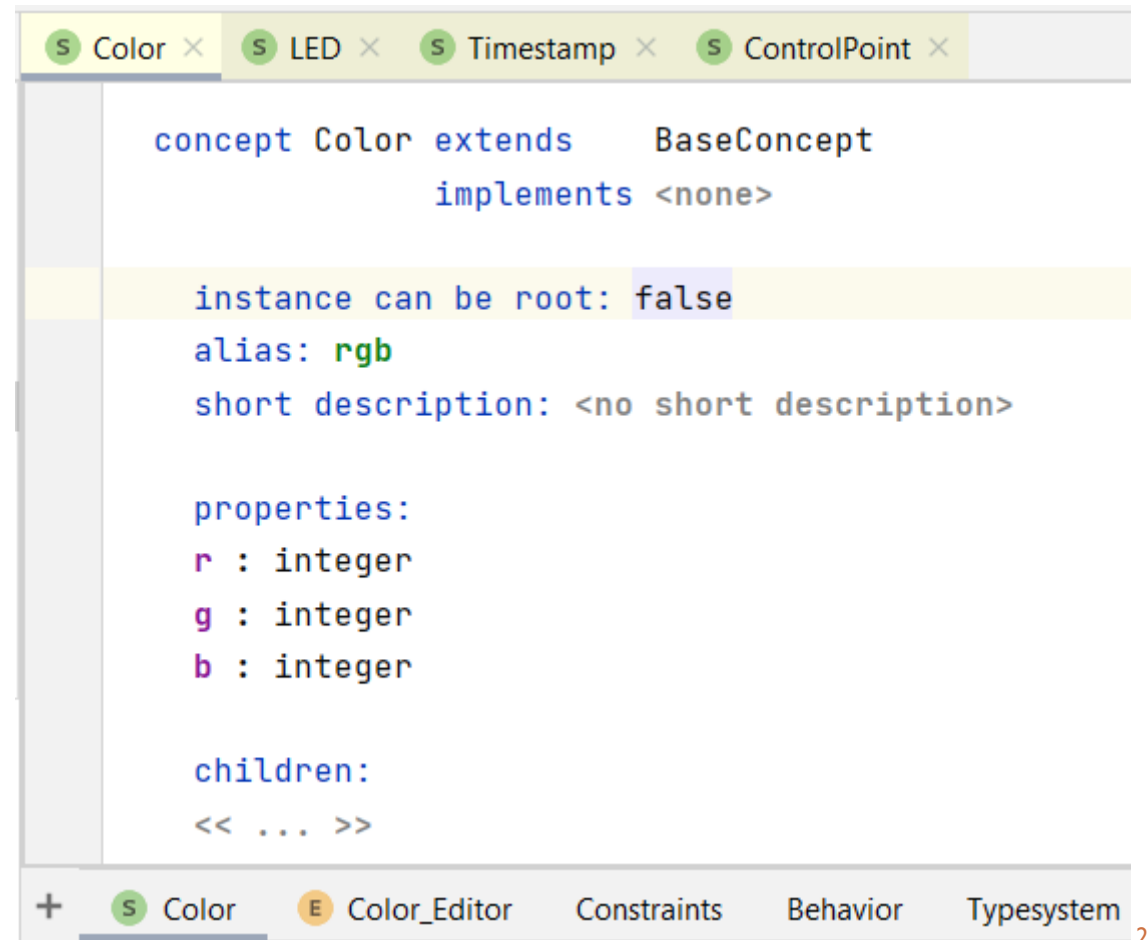


# MPS – CONCEPT LÉTREHOZÁSA



# MPS-VSR – COLOR CONCEPT KONFIGURÁLÁSA

- Properties:
  - Primitív adatrekordok (string, int, bool)
  - Színcsatornák » integer
- Alias:
  - Rövid példányosító azonosító
- Advanced: Constraints



The screenshot shows the MPS IDE interface with four tabs: 'Color', 'LED', 'Timestamp', and 'ControlPoint'. The 'Color' tab is active, displaying the following configuration:

```
concept Color extends BaseConcept
    implements <none>

instance can be root: false
alias: rgb
short description: <no short description>

properties:
  r : integer
  g : integer
  b : integer

children:
  << ... >>
```

At the bottom, a navigation bar shows the following tabs: '+', 'Color', 'Color\_Editor', 'Constraints', 'Behavior', and 'Typesystem'. The 'Color' tab is currently selected.

# MPS-VSR – LED CONCEPT

- Children:
  - Concept tartalmazása
    - Kardinalitás?
    - Behelyettesíthetőség?
  - Egymásba ágyazott konstruktorok
    - Behavior fül
- Advanced: Extends, Implements

The screenshot shows the MPS-VSR IDE interface. At the top, there are four tabs: 'Color', 'LED', 'Timestamp', and 'ControlPoint'. The 'LED' tab is currently selected. The main editor area displays the following code:

```
concept LED extends BaseConcept
    implements <none>

instance can be root: false
alias: led
short description: <no short description>

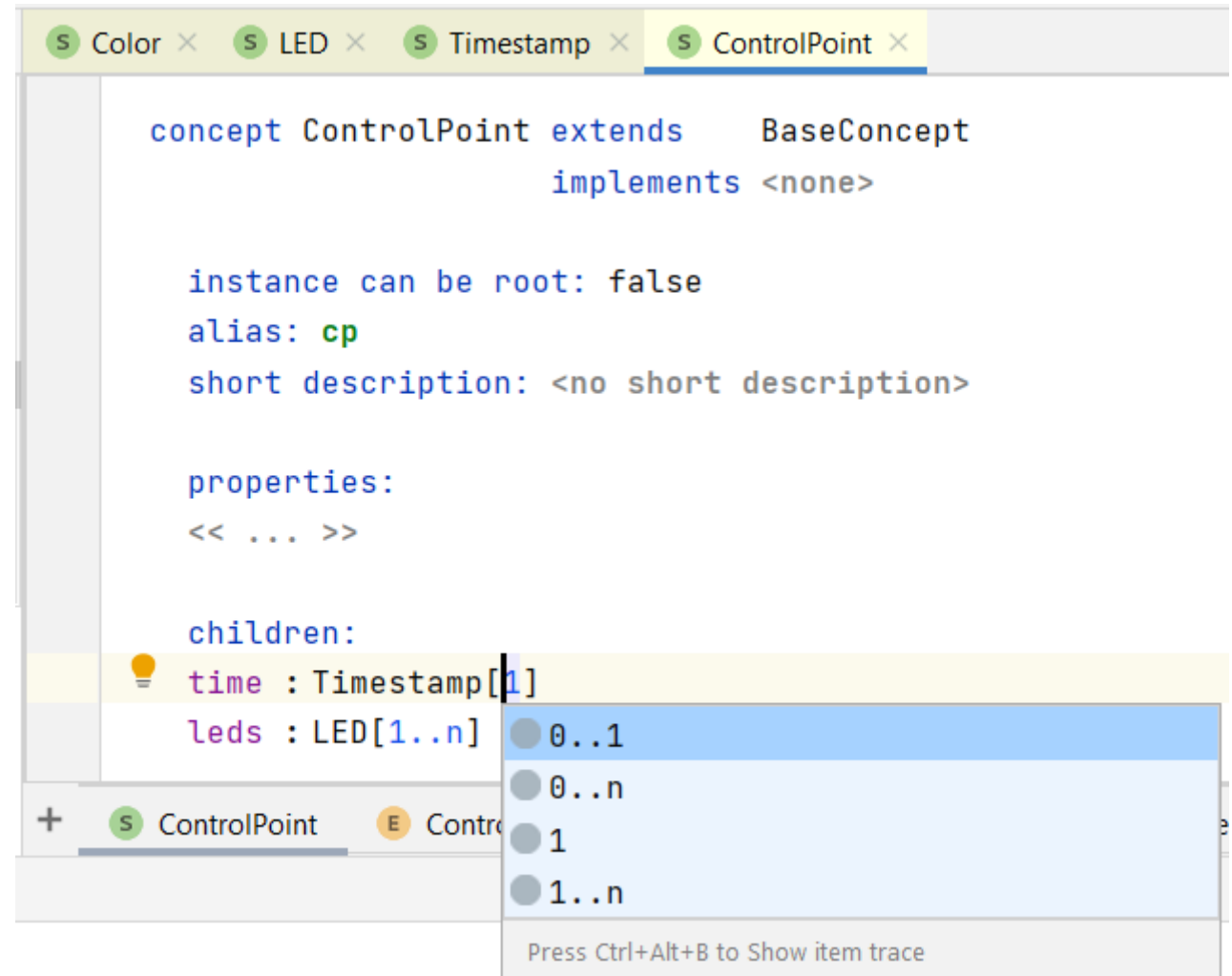
properties:
    index : integer

children:
    color : Color[1]
```

At the bottom of the IDE, there is a toolbar with a '+' icon and several buttons: 'LED' (selected), 'LED\_Editor', 'Constraints', 'Behavior', and 'Typesystem'.

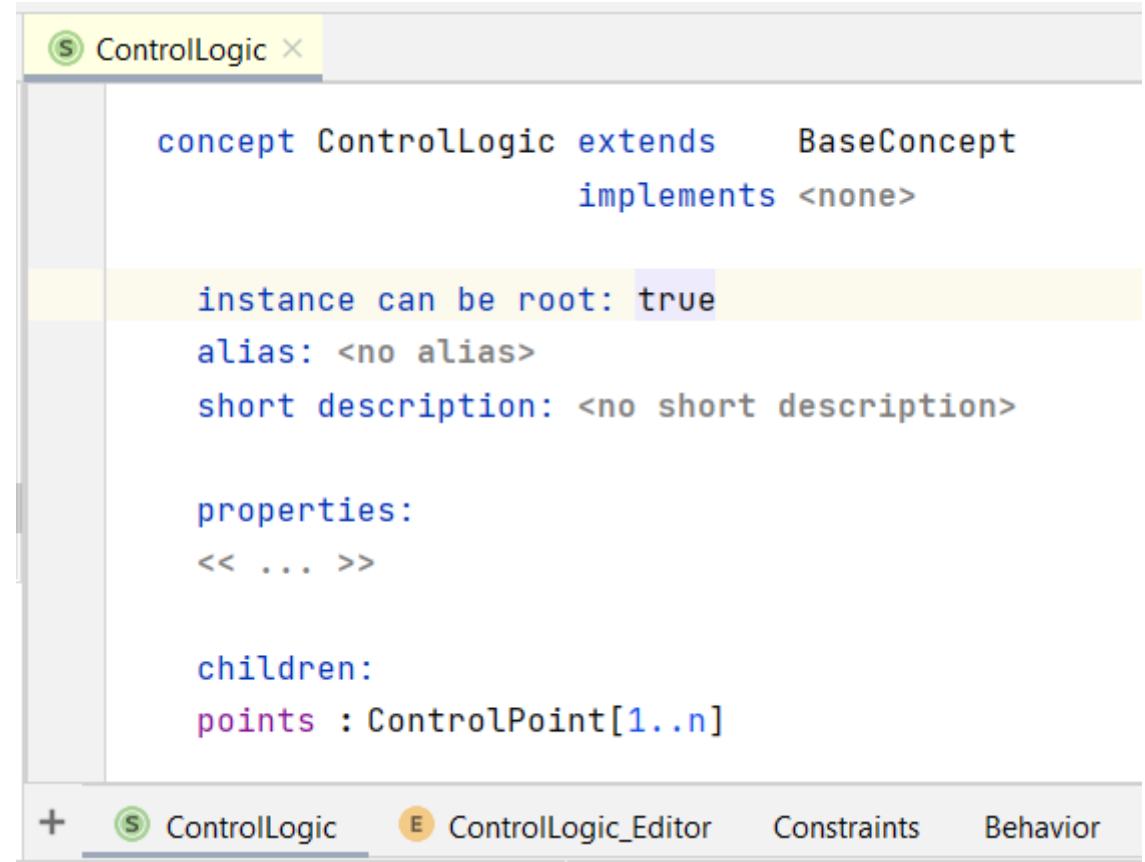
# MPS-VSR – CONTROL POINT

- Timestamp létrehozása nem új
- Kardinalitás:
  - Projekcionális 😊
  - | » automatikus példányosítás
  - 0..1 » szerkesztői döntési lehetőség
  - 0..n » tetszőleges kollekció
  - 1..n » nem üres kollekció



# MPS-VSR – CONTROL LOGIC

- Root:
  - Sandboxban példányosítható
    - szintaxisfa gyökere
    - *szintaxisfa?*
- Szótár, nyelvtan kész. 😊



The screenshot shows the MPS IDE interface with a tab titled 'ControlLogic'. The editor displays the following code:

```
concept ControlLogic extends BaseConcept
    implements <none>

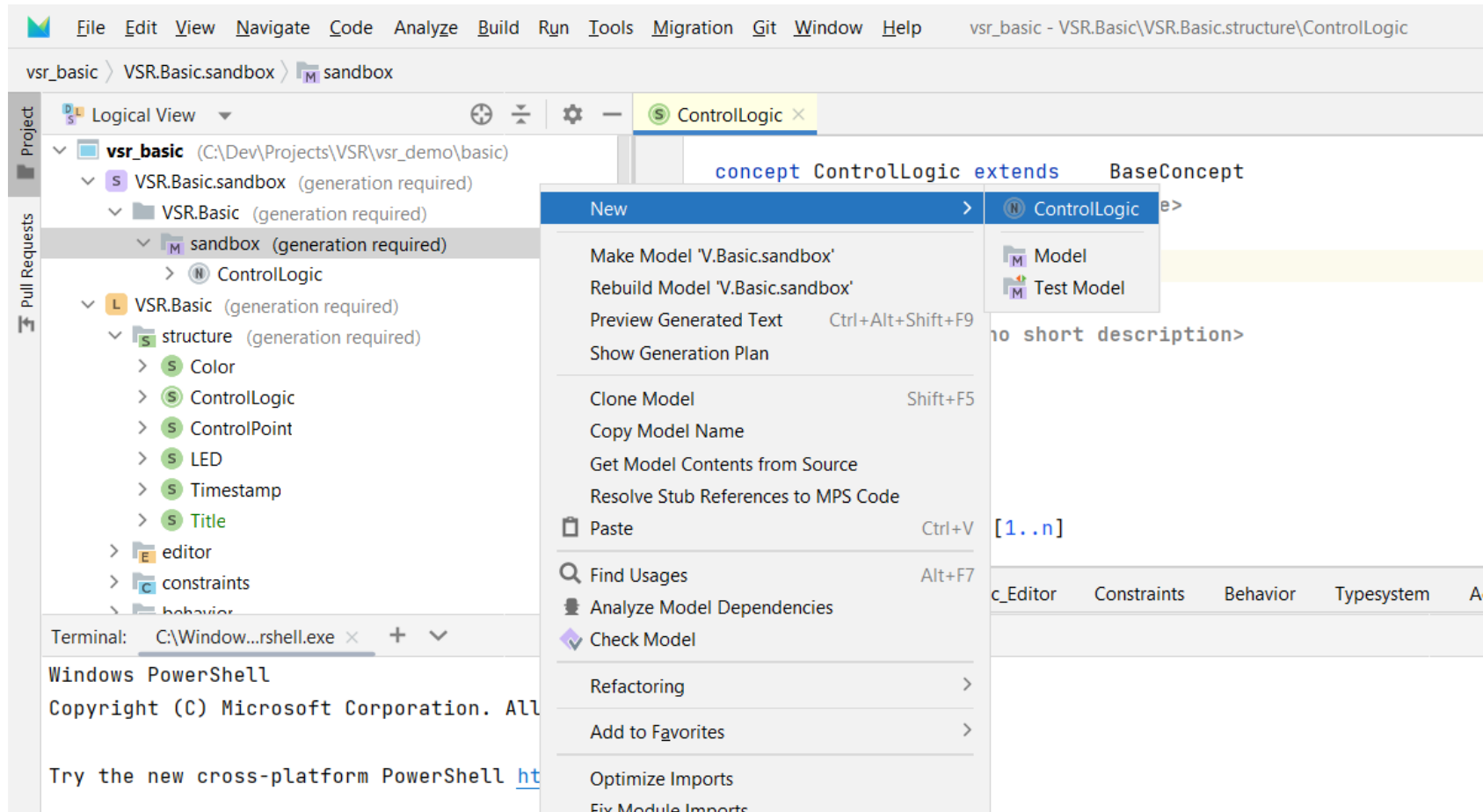
instance can be root: true
alias: <no alias>
short description: <no short description>

properties:
<< ... >>

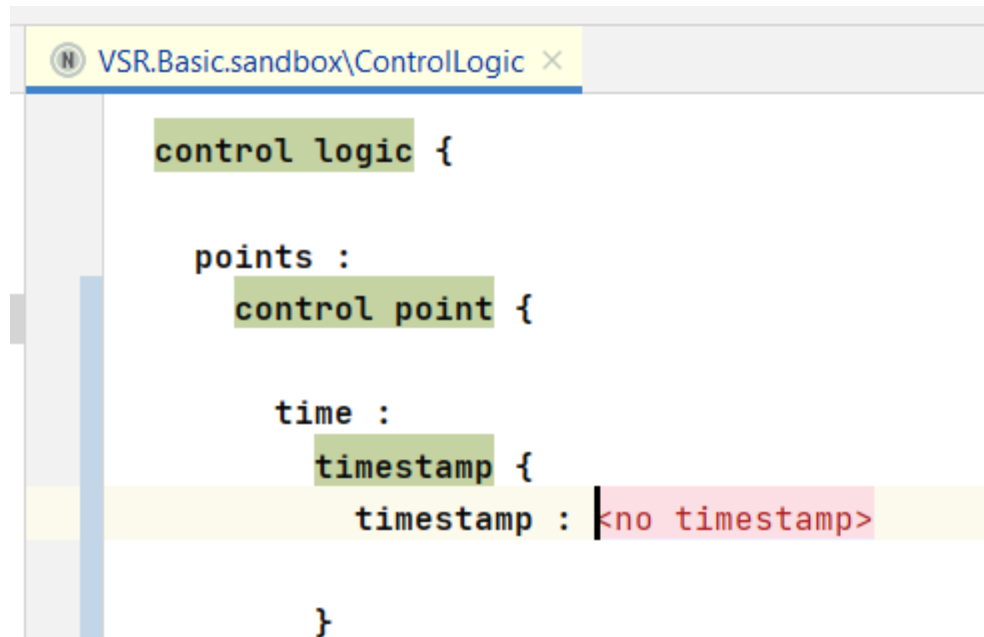
children:
points : ControlPoint[1..n]
```

The bottom of the IDE shows a toolbar with icons for adding new elements and switching between different views: 'ControlLogic' (selected), 'ControlLogic\_Editor', 'Constraints', and 'Behavior'.

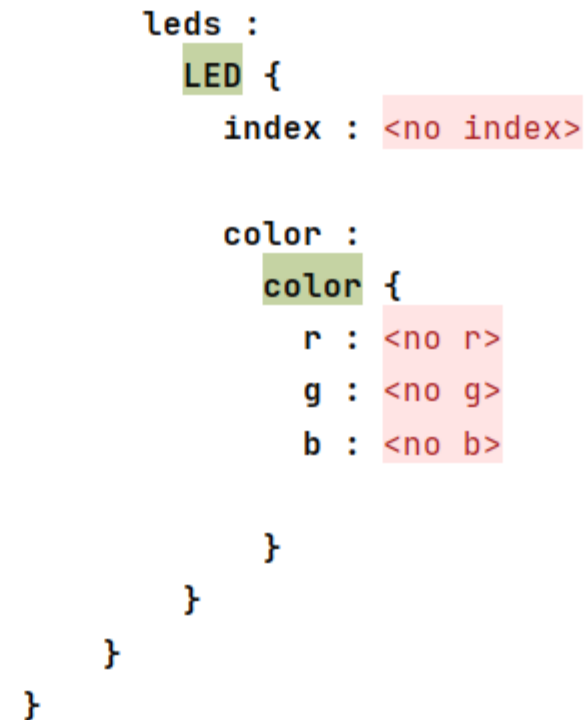
# MPS-VSR – SANDBOX



# MPS-VSR – DEFAULT EDITOR



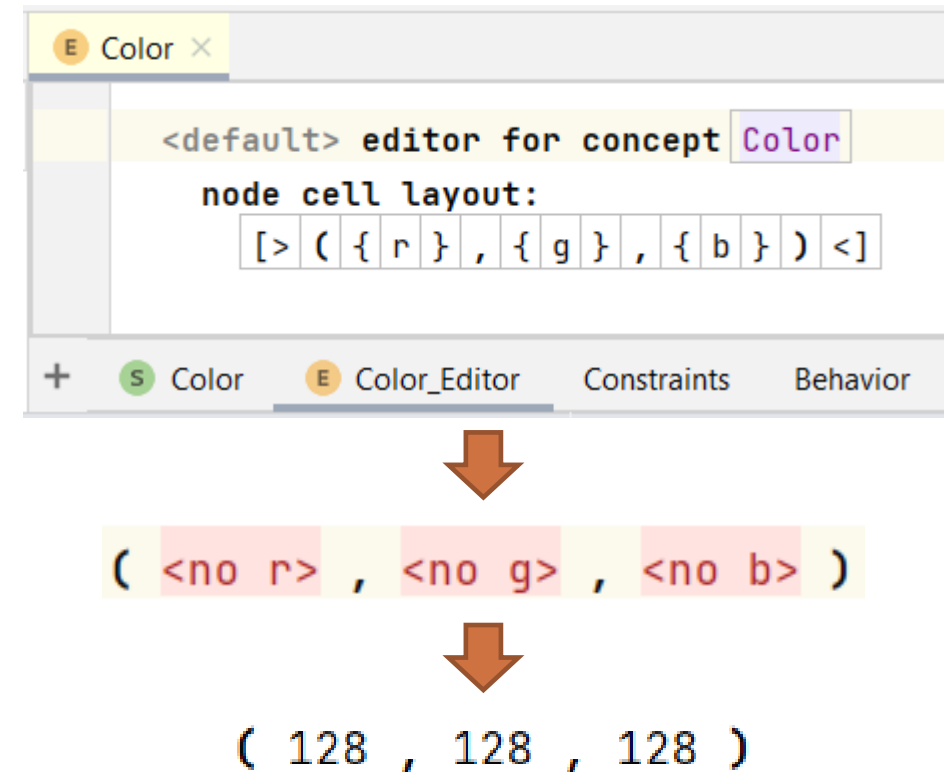
```
control logic {  
    points :  
        control point {  
            time :  
                timestamp {  
                    timestamp : <no timestamp>  
                }  
        }  
}
```



```
leds :  
    LED {  
        index : <no index>  
  
        color :  
            color {  
                r : <no r>  
                g : <no g>  
                b : <no b>  
            }  
        }  
    }  
}
```

# MPS-VSR – SAJÁT EDITOR

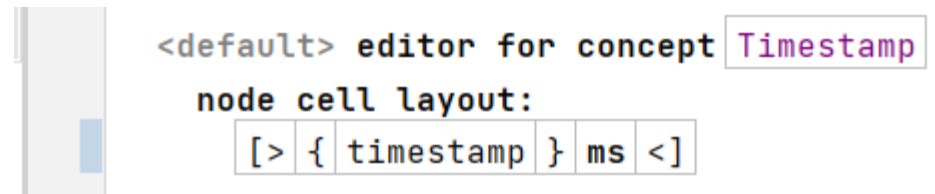
- Editor:
  - Projekcionális ☺
  - Minden concept-nek van sajátja
  - Publikus interface
  - Szöveges, grafikus, interaktív elemek
- Advanced: Editor components, öröklés



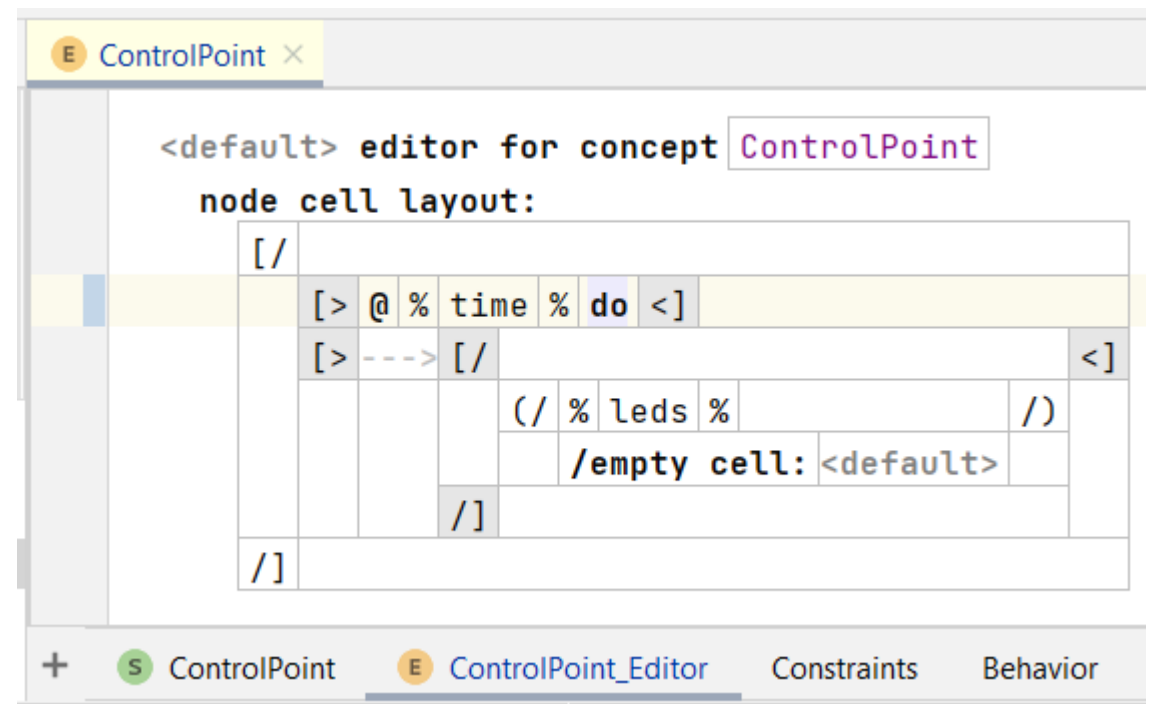


# MPS-VSR – EDITOR PÉLDÁK

- Editor:
  - Mértékegységek
  - Timestamp » ms
- Összetett editorok:
  - Egymásba ágyazás
  - Indent, layout
  - Kollekción



```
<default> editor for concept Timestamp  
node cell layout:  
[> { timestamp } ms <]
```



E ControlPoint x

```
<default> editor for concept ControlPoint  
node cell layout:  
[/  
[> @ % time % do <]  
[> ---> [/ <]  
    (/ % leds % /)  
    /empty cell: <default>  
    /]  
/]
```

+ S ControlPoint E ControlPoint\_Editor Constraints Behavior

# MPS-VSR - SANDBOX

The screenshot displays the MPS-VSR IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Build, Run, Tools, Migration, Git, Window, and Help. The breadcrumb path is `vsr_basic > VSR.Basic.sandbox > sandbox > ControlLogic > ms`. The left sidebar shows the Project view with a tree structure: `vsr_basic` (C:\Dev\Projects\VSR\vsr\_demo\basic) contains `VSR.Basic.sandbox` (generation required), which contains `VSR.Basic` (generation required), which contains `sandbox` (generation required). The `sandbox` folder contains `ControlLogic` and `ControlLogic`. The `ControlLogic` module is selected, and its Logical View is shown in the main editor. The Logical View displays the following code:

```
LED visualisation
Controlled Points:
@ <no timestamp> ms do
  LED <no index> » ( <no r> , <no g> , <no b> )
```

# MPS-VSR - SANDBOX

The screenshot displays the MPS-VSR IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Build, Run, Tools, Migration, Git, Window, and Help. The breadcrumb path at the top reads: `vsr_basic > VSR.Basic.sandbox > sandbox > ControlLogic > ms`.

The left sidebar shows the Project and Pull Requests views. The Project view is expanded, showing the following structure:

- `vsr_basic` (C:\Dev\Projects\VSR\vsr\_demo\basic)
  - `VSR.Basic.sandbox` (generation required)
    - `VSR.Basic` (generation required)
      - `sandbox` (generation required)
        - `ControlLogic` (selected)
  - `VSR.Basic` (generation required)
    - `structure`
      - `Color`
      - `ControlLogic`
      - `ControlPoint`
      - `LED`
      - `Timestamp`

The main editor window displays the `ControlLogic` module. The code is as follows:


```
LED visualisation
Controlled Points:
@ 0 ms do
  LED 10 » ( 0 , 0 , 0 )
@ 5000 ms do
  LED 20 » ( 32 , 32 , 0 )
  LED 70 » ( 100 , 100 , 100 )
@ 15000 ms do
  LED <no index> » ( <no r> , <no g> , <no b> )
```

# MPS - KÓDGENERÁLÁS

- Minden nyelvhez tartozhat saját generátor
- Bemenet: AST » Input model
- Kimenet: AST » Output model (» forráskód)
- Általunk definiált transzformáció
  - Átalakítás » redukciós szabályok
  - Beillesztés » makrók

# MPS-VSR – GENERÁTOR KONFIGURÁLÁSA

- Kimenet: XML fájl » *jetbrains.mps.core.xml* natív nyelv
- Minden ControlLogic conceptből legyen XML » Root mapping rule



The screenshot displays the MPS IDE interface. On the left, the 'Pull Requests' panel shows a tree structure of VSR components. The 'generator/VSR.Basic/main' component is selected, and its sub-component 'VSR.Basic.generator' is expanded, showing the 'templates@generator' folder. The 'VSR\_XML' template is highlighted. On the right, the 'is applicable:' section shows '<always>'. The 'conditional root rules:' section shows '<< ... >>'. The 'root mapping rules:' section shows a table with the following content:

concept	ControlLogic	-->	VSR_XML
inheritors	false		
condition	<always>		
keep input root default			

# MPS-VSR – GENERÁTOR BEMENETE

The screenshot displays the MPS-VSR IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Build, Run, Tools, Migration, Git, Window, and Help. The title bar shows the current file path: `vsr_basic - VSR.Basic.generator\VSR.Basic.generator.templates@generator\Output`.

The left sidebar shows the Project view with the following structure:

- vsr\_basic** (C:\Dev\Projects\VSR\vsr\_demo\basic)
  - VSR.Basic.sandbox (generation required)
  - VSR.Basic (generation required)
    - structure
    - editor (generation required)
    - constraints
    - behavior
    - typesystem
    - generator/VSR.Basic/main (generation required)
      - VSR.Basic.generator (generation required)
        - templates@generator (generation required)
          - Output**
          - VSR\_XML
          - main
    - all models
    - Modules Pool
    - checkpoints (read only)

The main editor area shows the `ControlLogic` template. The `input` section is highlighted, showing a list of concepts and their declarations:

```
[root template]
input Con
xml Out
<no pro
<no ele
```

Concept	Declaration
<code>ControlLogic</code>	<code>^(BaseConcept in VSR.Basic)</code>
<code>ControlPoint</code>	<code>^(BaseConcept in VSR.Basic)</code>
<code>BaseConcept</code>	<code>^ConceptDeclaration (j.m.l.core.structure)</code>
<code>IContainer</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>INamedConcept</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>IOldCommentContainer</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>IPlaceholderContent</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>ISkipConstraintsChecking</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>IStubForAnotherConcept</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>ImplementationContainer</code>	<code>^InterfaceConceptDeclaration (j.m.l.core.structure)</code>
<code>ScopeConcept</code>	<code>^InterfaceConceptDeclaration (j.m.l.traceable.structure)</code>

Press Ctrl+Alt+B to Show item trace

# MPS-VSR – XML FEJLÉC

- Projekcionális – nincs szintaktikai hibalehetőség!

The screenshot displays the MPS-VSR IDE interface. On the left, a project tree shows the structure of a project named 'vsr\_basic'. The tree includes a 'Pull Requests' sidebar, a 'VSR.Basic.sandbox' folder, and a 'VSR.Basic' folder. The 'VSR.Basic' folder is expanded, showing subfolders like 'structure', 'editor', 'constraints', 'behavior', 'typesystem', and 'generator/VSR.Basic/main'. The 'generator/VSR.Basic/main' folder is further expanded, showing 'VSR.Basic.generator' and 'templates@generator'. The 'templates@generator' folder is expanded, showing 'Output' and 'VSR XML'.

On the right, a code editor displays the XML output of the 'VSR XML' template. The code is as follows:

```
[root template  
input ControlLogic  
]  
  
xml Output.xml
```

A tooltip is visible over the code editor, listing XML-related items with their corresponding icons:

- (j.m.core.xml)
- xml doctype declaration
- processing instruction
- xml declaration

Press Ctrl+Alt+B to Show item trace

# MPS-VSR – XML ELEMENT

- **vsr\_basic** (C:\Dev\Projects\VSR\vsr\_demo\basic)
  - > S VSR.Basic.sandbox (generation required)
  - ▼ L VSR.Basic (generation required)
    - > S structure
    - > E editor (generation required)
    - > C constraints
    - > B behavior
    - > T typesystem
    - ▼ G generator/VSR.Basic/main (generation required)
      - ▼ VSR.Basic.generator (generation required)
        - ▼ T templates@generator (generation required)
          - > N Output
          - > N VSR\_XML
          - > ⇒ main

```
[root template  
input ControlLogic]
```

xml Output.xml

```
<?xml version = "1.0" encoding = "default" standalone = "default" ?>
```

```
<no element>
```

```
N <element/> (j.m.core.xml)
```

Press Ctrl+Alt+B to Show item trace



# MPS-VSR – GENERÁTOR MAKRÓK

## ■ Gyakori makrók

- Property
- Copy
- Loop

## ■ Példa:

- Minden ControlPoint-hoz új XML sor

```
[root template  
input ControlLogic]
```

xml Output.xml

```
<?xml version = "1.0" encoding = "default" standa  
<ControlLogic>
```

```
<ControlPoint></ControlPoint>
```

```
</
```

Intentions

- ✎ Add LOOP macro over node.points >
- ✎ Add LOOP macro over node.smodelAttribute >
- ✎ Add Node Macro >
- ✎ Apply COPY\_SRCL over node.points >
- ✎ Apply COPY\_SRCL over node.smodelAttribute >
- ✎ Turn to Element >
- Attach Mapping Label

# MPS-VSR – PROPERTY MAKRÓ

- Property makró
  - Koncepció példányából
  - Generál kódot
  - » string kimenetű függvény
- Példa:
  - Timestamp

The screenshot shows the MPS IDE interface. At the top, a code editor displays a snippet of a `<ControlLogic>` macro with a `$LOOP$` loop containing a `<ControlPoint>` block. Inside the `<ControlPoint>` block, there is a `<Timestamp>` block with a value of `1000`. Below the code editor, the 'Inspector' panel is visible, showing the class `jetbrains.mps.lang.generator.structure.PropertyMacro`. The 'property value' section is expanded, showing the implementation of the `value` property. The implementation is a lambda function that takes `templateValue`, `genContext`, and `node` as arguments and returns a string. The string is generated by calling `Integer.toString(node.time.timestamp)`.

```
<ControlLogic>
$LOOP$ <ControlPoint>
    <Timestamp>$[1000]</Timestamp>
</ControlLogic>
```

Inspector

jetbrains.mps.lang.generator.structure.PropertyMacro

```
property value

comment : <none>
value : (templateValue, genContext, node)->string {
    string str = Integer.toString(node.time.timestamp);
    return str;
}
```

# MPS-VSR – LOOP MAKRÓ

- Loop makró
  - Kollekciónak bejárása
  - Bejárás felüldefiniálható
  - » sequence visszatérésű függvény

- Példa:

- Minden LED színét kiírjuk

```
<ControlLogic>
  $LOOP$ [<ControlPoint>
    <Timestamp>${1000}</Timestamp>
    $LOOP$ [<Color>${(R=100,G=100,B=100)}</Color> ]
  ]
</ControlLogic>
```

+ Logic   E ControlLogic\_Editor   Constraints   Behavior   Typesystem   Actions   Refa

Inspector

jetbrains.mps.lang.generator.structure.LoopMacro

```
iterate over sequence of nodes

comment      : <none>
mapping label : <no label>
iteration sequence : (genContext, node)->sequence<node<>> {
    node.leds;
}
```

# MPS-VSR – FORDÍTÁS

```
ControlLogic x
```

LED visualisation  
Controlled Points:

```
@ 0 ms do
  LED 10 » ( 0 , 0 , 0 )
@ 5000 ms do
  LED 20 » ( 32 , 32 , 0 )
  LED 70 » ( 100 , 100 , 100 )
@ 15000 ms do
  LED <no index> » ( <no r> , <no g> , <no b> )
```



```
ControlLogic x VSR_XML.xml x
```

```
1 |<?xml version = "1.0"?>
2 |<ControlLogic>
3 |  <ControlPoint>
4 |    <Timestamp>0</Timestamp>
5 |    <Color>(R=0,G=0,B=0)</Color>
6 |  </ControlPoint>
7 |  <ControlPoint>
8 |    <Timestamp>5000</Timestamp>
9 |    <Color>(R=32,G=0,B=0)</Color>
10 |    <Color>(R=100,G=100,B=100)</Color>
11 |  </ControlPoint>
12 |  <ControlPoint>
13 |    <Timestamp>15000</Timestamp>
14 |    <Color>(R=0,G=0,B=0)</Color>
15 |  </ControlPoint>
16 |</ControlLogic>
```

# VSR DEMÓ

- VSR-DSL



- VSR-XML



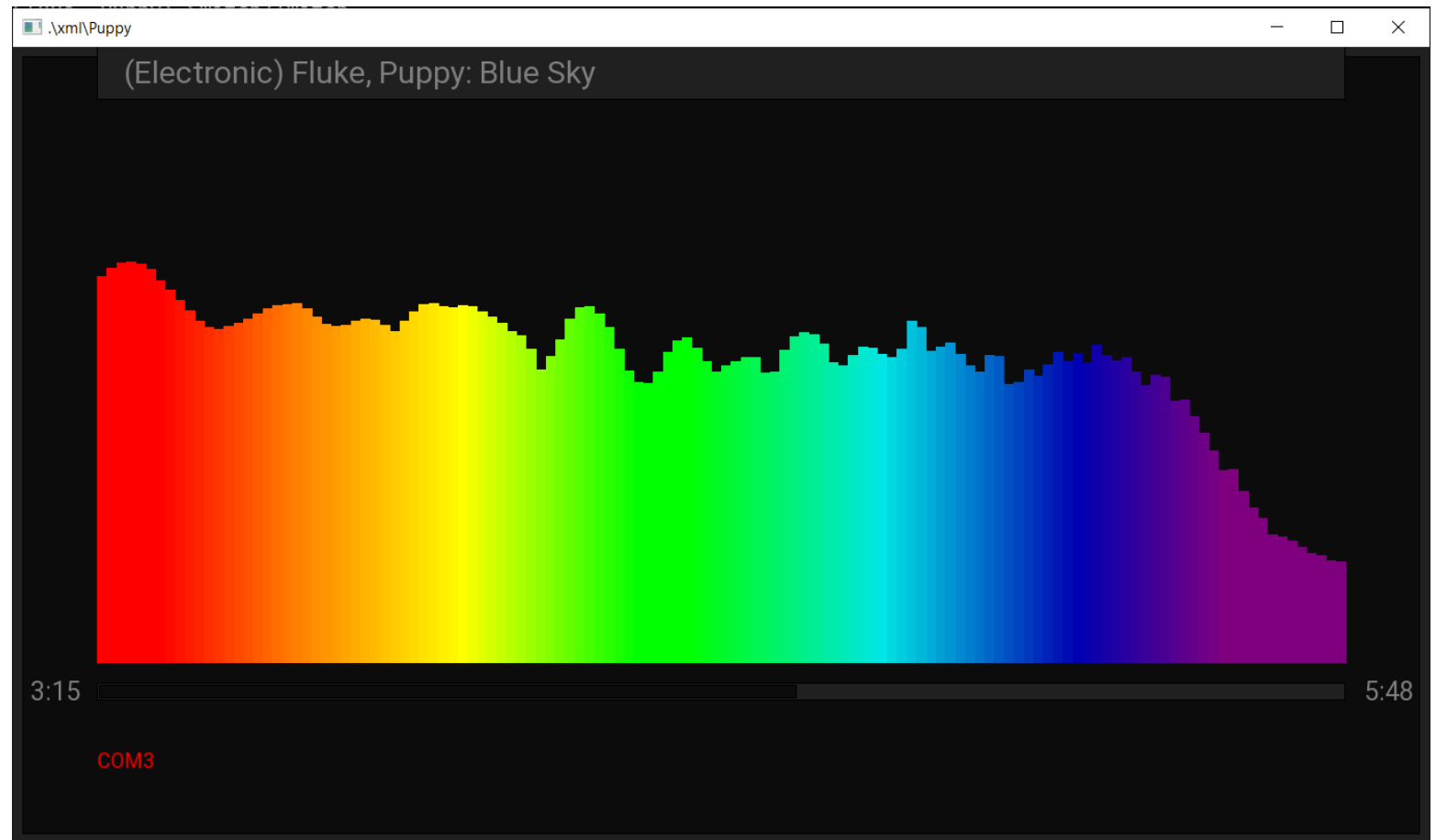
- C++ szimulátor



- STM32 MCU



- RGB szalag



# MPS KITEKINTÉS

- Felhasználási lehetőségek
  - Okosóra firmware konfigurációja, pl. menürendszer DSL » C kód
  - Rendszerek tesztjeinek modellezése, pl. unit-teszt DSL » tesztesetek generálása
  - Adminisztratív feladatok automatizálása, pl. pénzügyi jelentések generálása
  - GPGPU modellező DSL » pl. GPU kernelek generálása és debuggolása
  - Matematikai DSL » pl. tételbizonyító környezet
  - Szakorvosi DSL » pl. diagnózist segítő környezet



KÖSZÖNJÜK A FIGYELMET!