GENERATING JAVA RMI CODE FOR THE DISTRIBUTED ASPECTS OF VDM-RT MODELS

Miran Hasanagic, Peter Gorm Larsen, Peter W. V. Tran-Jørgensen

AU AARHUS UNIVERSITET
SCIENCE AND TECHNOLOGY

AGENDA

Introduction

Distribution in VDM-RT

Java RMI

Code Generation VDM-RT models

Conclusion

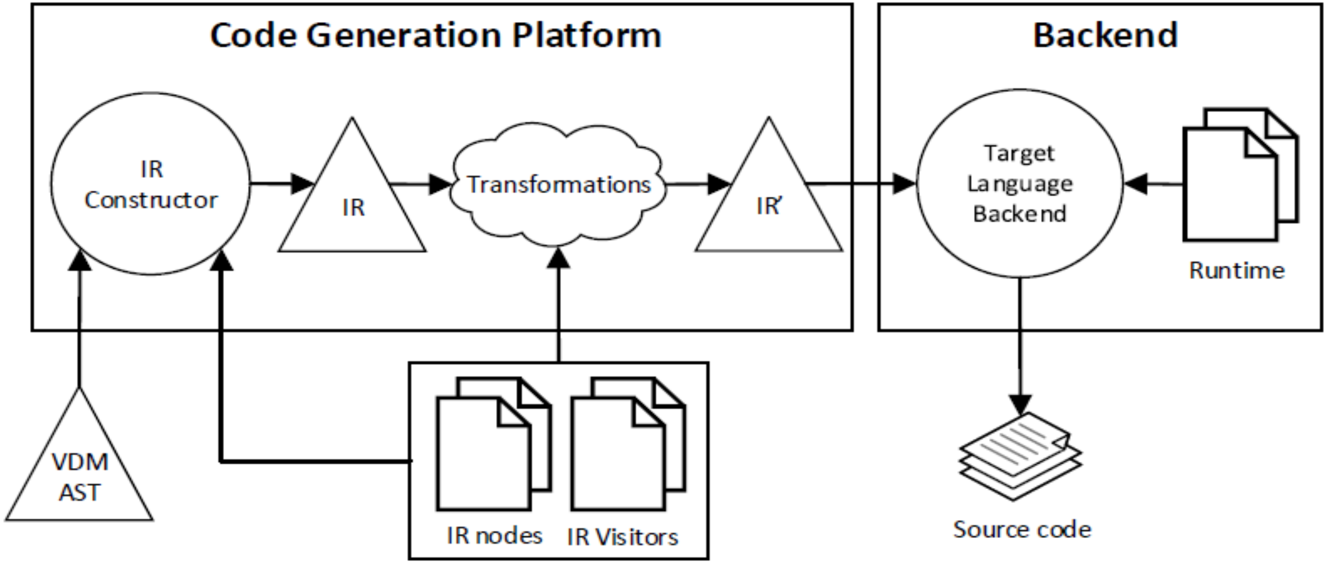
Future Work

Overture vision

INTRODUCTION

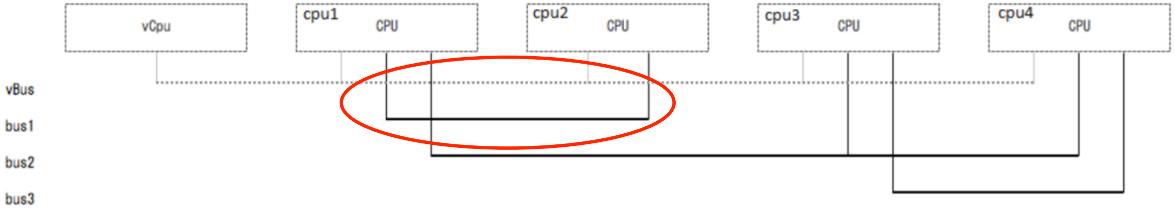
- Code Generation of VDM models → Implementation
- Dialects:
  - VDM-SL for sequential and functional modelling
  - VDM-PP for object oriented modelling
  - VDM-RT for modelling of time aspects and distributed architecture
- Focus is on the distributed aspects of VDM-RT models

CODE GENERATION PLATFORM



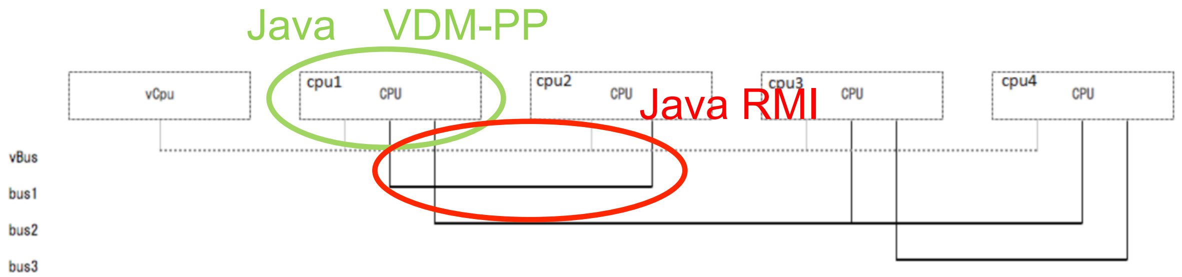
DISTRIBUTION IN VDM-RT MODELS

- Distribution is modelled inside the **system** definition



VDM-RT CODE GENERATION

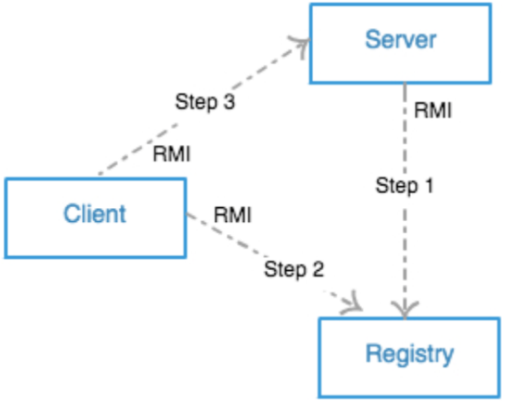
- Noval area of research
- First support the distributed aspects of VDM-RT
- Current version supports Java code generation for VDM-PP models
- Use Java Remote Method Invocation (RMI) in order to enable distributed communication between Java Virtual Machines

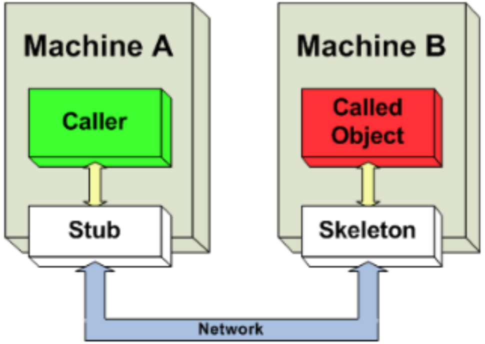


JAVA RMI MOTIVATION

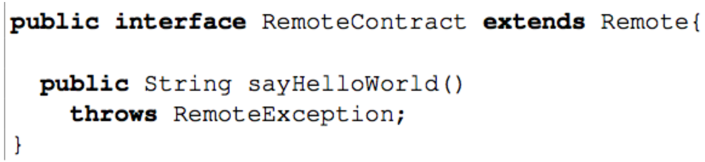
- Static Distributed System
- Object-Oriented Distribution
- Communication paradigm: Remote Method Invocation

JAVA RMI



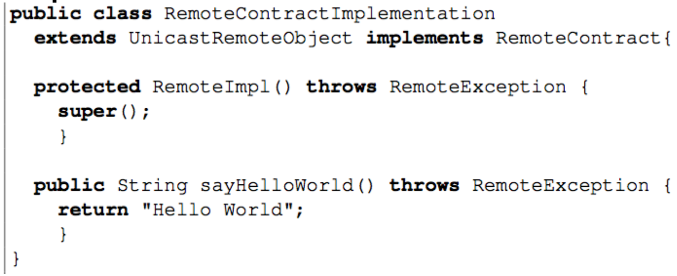


Interface→Client



```
public interface RemoteContract extends Remote{

  public String sayHelloWorld()
    throws RemoteException;
}
```

Implementation→Server

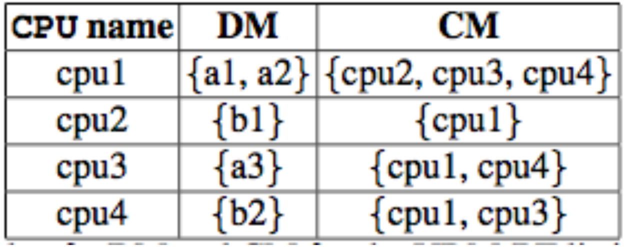


```
public class RemoteContractImplementation
  extends UnicastRemoteObject implements RemoteContract{

  protected RemoteImpl() throws RemoteException {
    super();
    }

  public String sayHelloWorld() throws RemoteException {
    return "Hello World";
    }
}
```

STEPS DURING CODE GENERATION

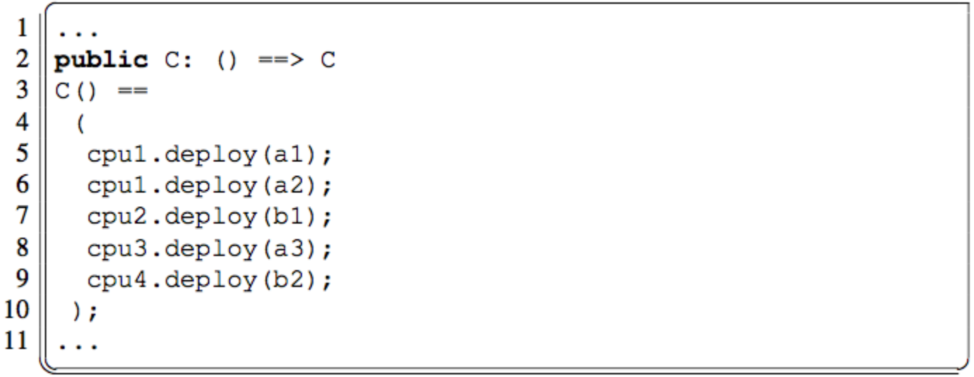
1. Extracting distribution information from a VDM-RT model
2. Code Generating VDM-RT classes
3. Transformation of method parameters and return values
4. Generating functionality of a single CPU
5. Enabling execution
6. Entry method of implemented model

1. DISTRIBUTION INFORMATION IN VDM-RT



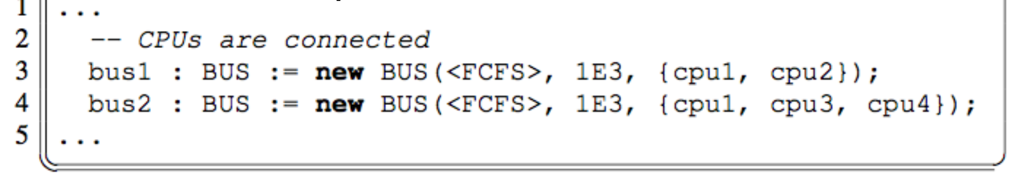
CPU name	DM	CM
cpu1	{a1, a2}	{cpu2, cpu3, cpu4}
cpu2	{b1}	{cpu1}
cpu3	{a3}	{cpu1, cpu4}
cpu4	{b2}	{cpu1, cpu3}

Distribution map



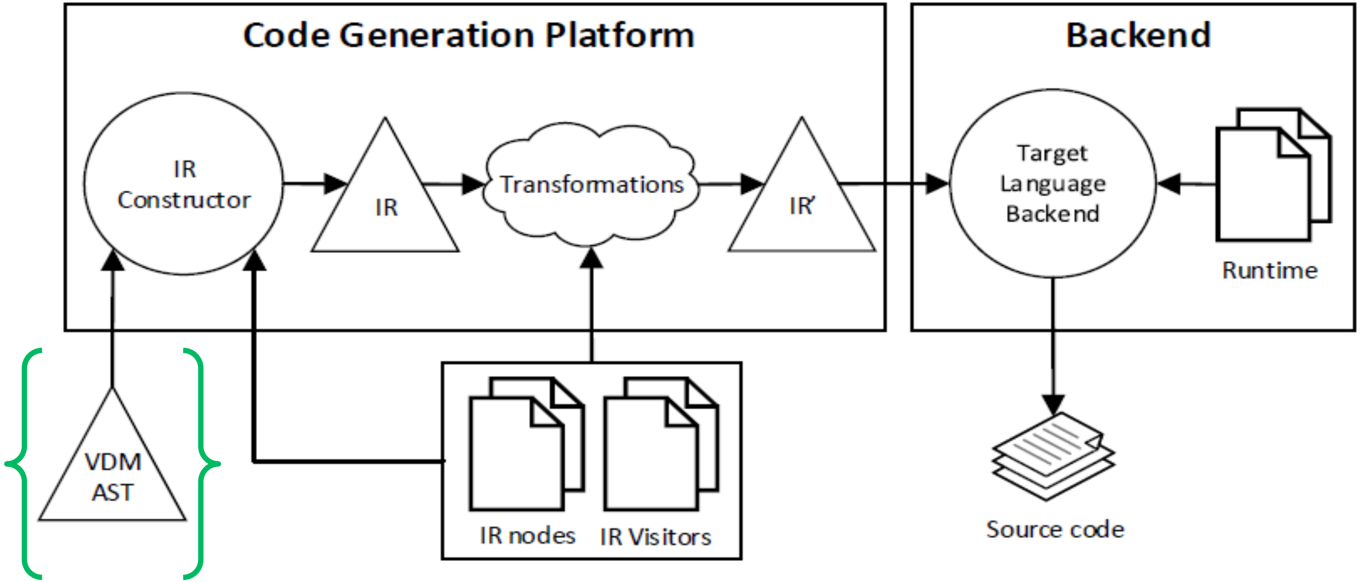
```
...
public C: () ==> C
C() ==
 (
  cpu1.deploy(a1);
  cpu1.deploy(a2);
  cpu2.deploy(b1);
  cpu3.deploy(a3);
  cpu4.deploy(b2);
 );
...
```

Connection map



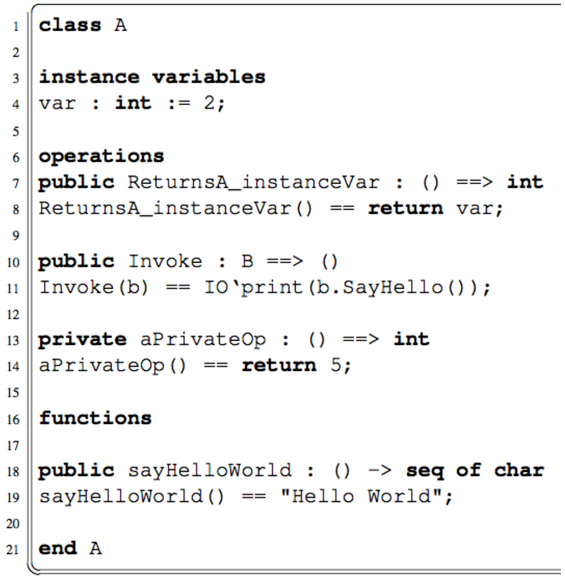
```
...
  -- CPUs are connected
  bus1 : BUS := new BUS(<FCFS>, 1E3, {cpu1, cpu2});
  bus2 : BUS := new BUS(<FCFS>, 1E3, {cpu1, cpu3, cpu4});
...
```

1. DISTRIBUTION INFORMATION IN VDM-RT



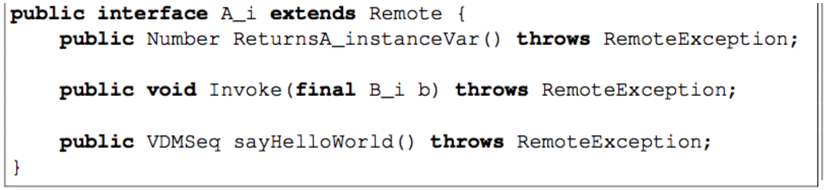
2. CODE GENERATING VDM-RT CLASSES

VDM-RT

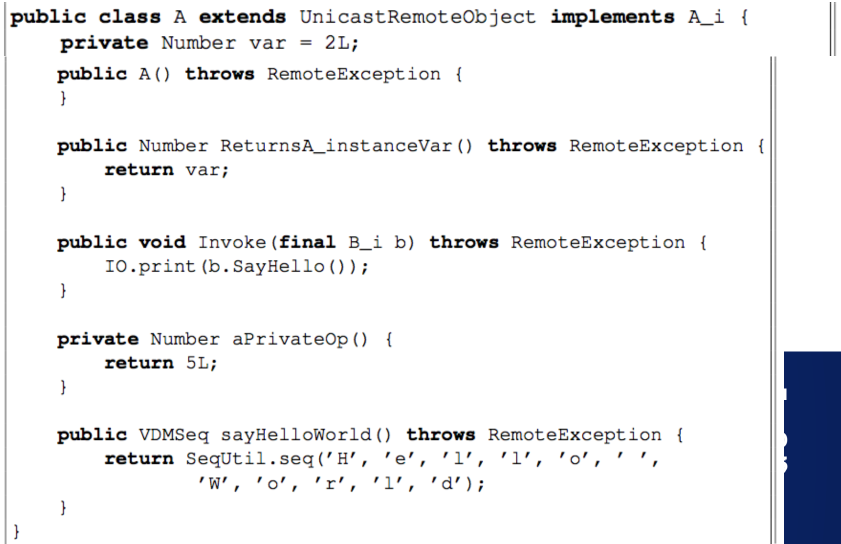


```
class A

instance variables
var : int := 2;

operations
public ReturnsA_instanceVar : () ==> int
ReturnsA_instanceVar() == return var;

public Invoke : B ==> ()
Invoke(b) == IO‘print(b.SayHello());

private aPrivateOp : () ==> int
aPrivateOp() == return 5;

functions

public sayHelloWorld : () -> seq of char
sayHelloWorld() == "Hello World";

end A
```

Java RMI code generator

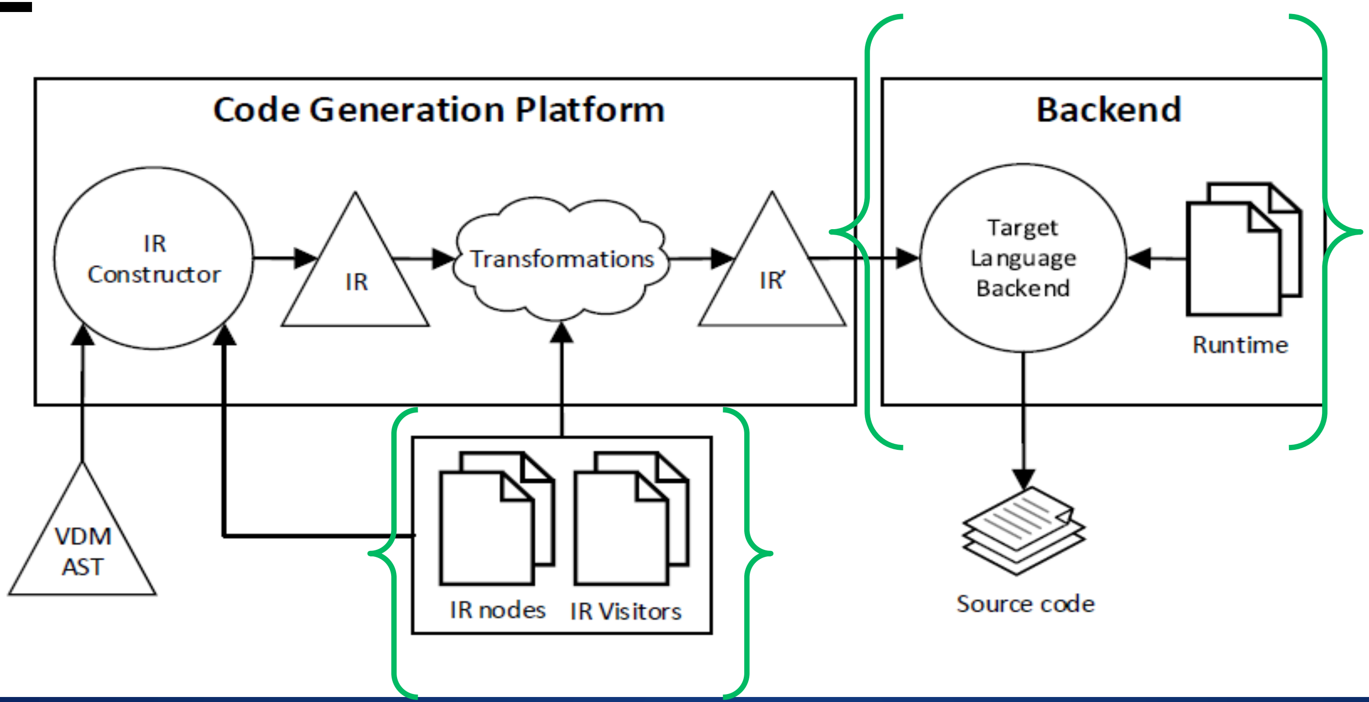


```
public interface A_i extends Remote {
    public Number ReturnsA_instanceVar() throws RemoteException;

    public void Invoke(final B_i b) throws RemoteException;

    public VDMSeq sayHelloWorld() throws RemoteException;
}
```



```
public class A extends UnicastRemoteObject implements A_i {
    private Number var = 2L;
    public A() throws RemoteException {
    }

    public Number ReturnsA_instanceVar() throws RemoteException {
        return var;
    }

    public void Invoke(final B_i b) throws RemoteException {
        IO.print(b.SayHello());
    }

    private Number aPrivateOp() {
        return 5L;
    }

    public VDMSeq sayHelloWorld() throws RemoteException {
        return SeqUtil.seq('H', 'e', 'l', 'l', 'o', ' ',
                'W', 'o', 'r', 'l', 'd');
    }
}
```

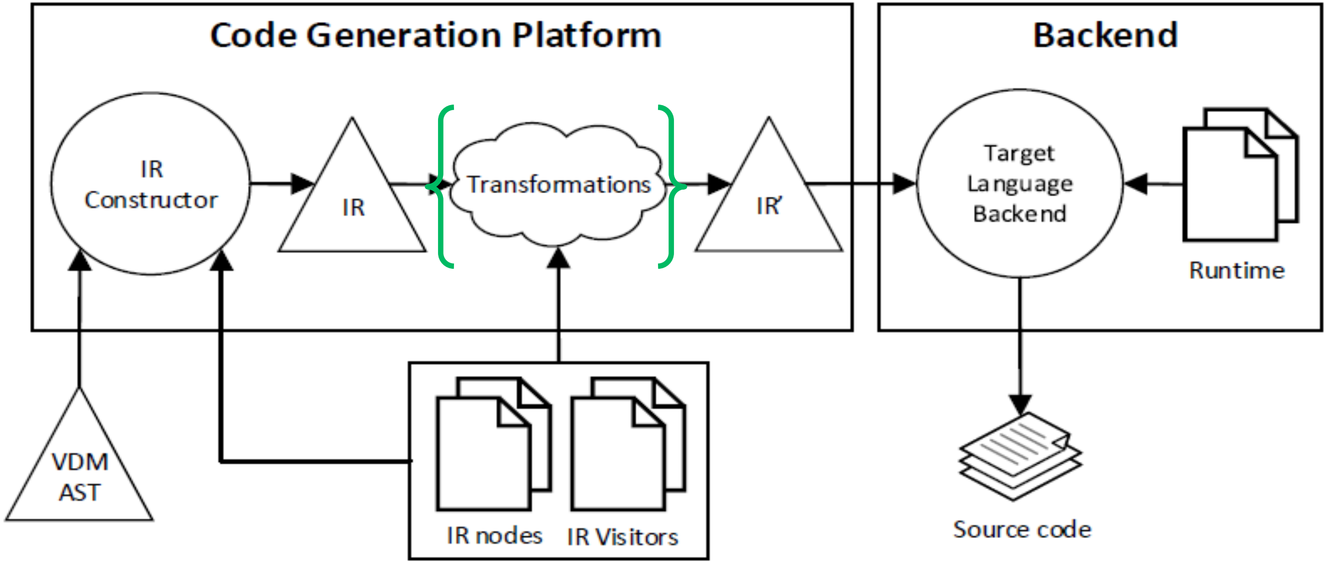
2. CODE GENERATING VDM-RT CLASSES



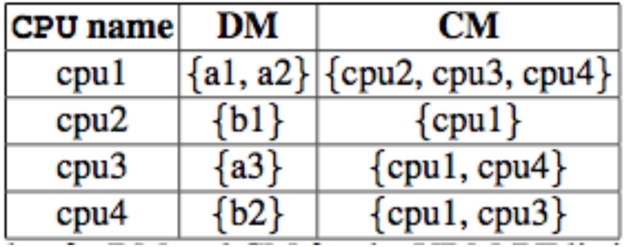
3. TRANSFORMATION

- Different representation between Java RMI and VDM-RT model
  - Implementation and interface
- For example if a method has a class type as a parameter

3. TRANSFORMATION

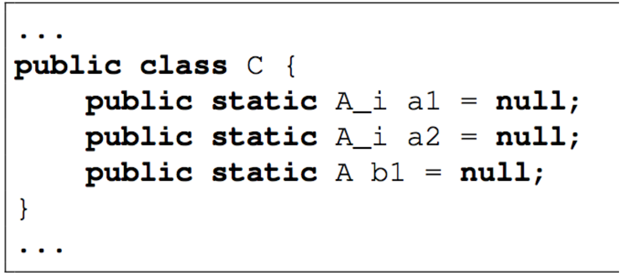


4. SINGLE CPU

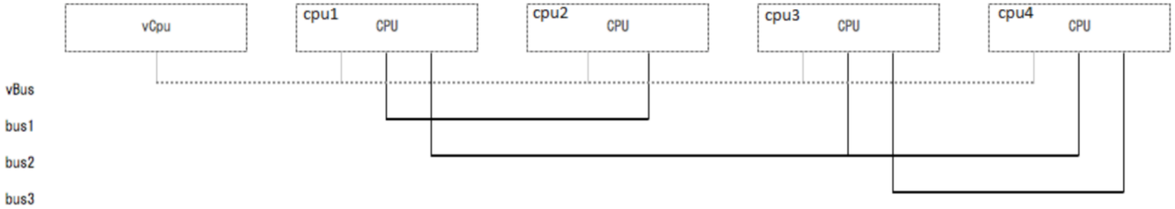


CPU name	DM	CM
cpu1	{a1, a2}	{cpu2, cpu3, cpu4}
cpu2	{b1}	{cpu1}
cpu3	{a3}	{cpu1, cpu4}
cpu4	{b2}	{cpu1, cpu3}

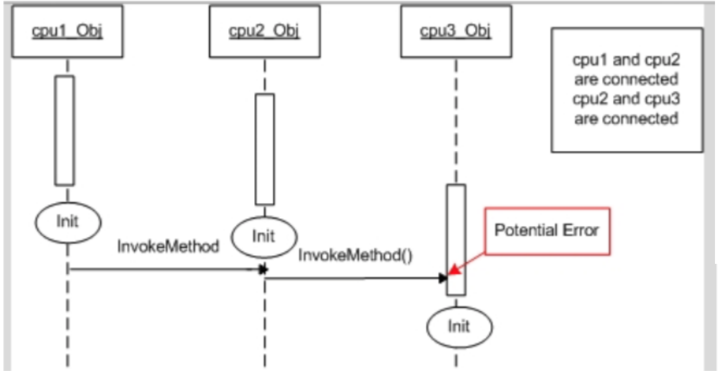
cpu2 – local system definition in Java

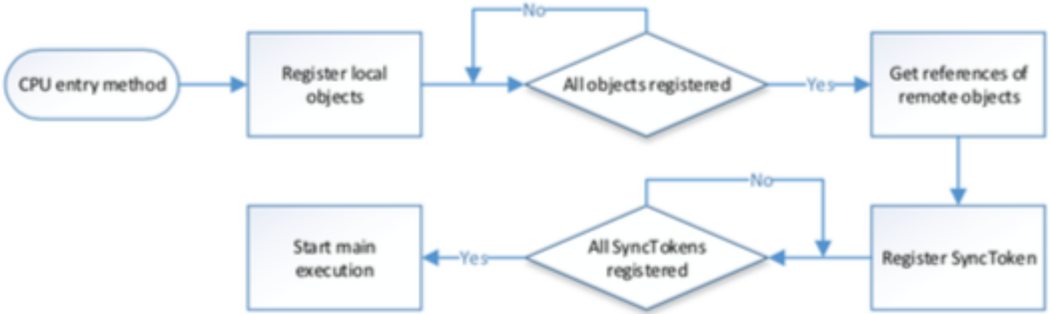


```
...
public class C {
    public static A_i a1 = null;
    public static A_i a2 = null;
    public static A b1 = null;
}
...
```



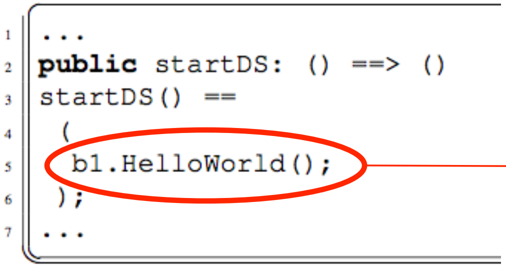
5. ENABLING EXECUTION



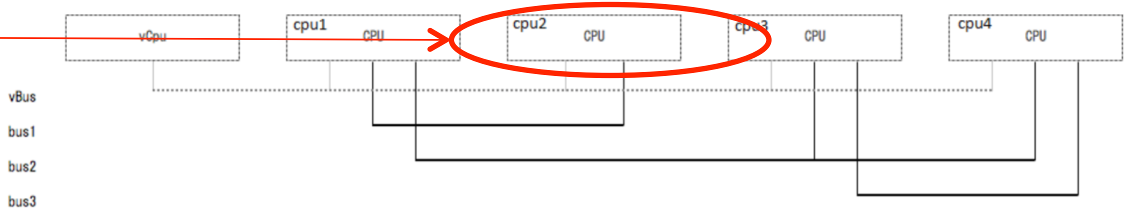


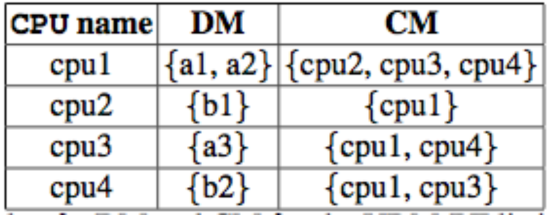
6. ENTRY METHOD OF IMPLEMENTED MODEL

- Provide **guidelines**



```
...
public startDS: () ==> ()
startDS() ==
  (
   b1.HelloWorld();
  );
...
```

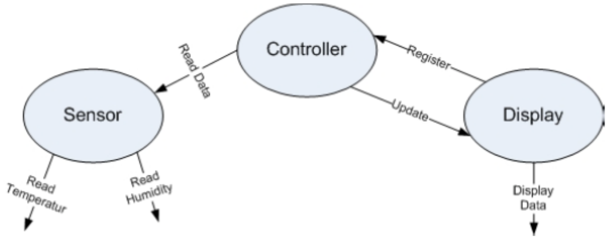




CPU name	DM	CM
cpu1	{a1, a2}	{cpu2, cpu3, cpu4}
cpu2	{b1}	{cpu1}
cpu3	{a3}	{cpu1, cpu4}
cpu4	{b2}	{cpu1, cpu3}

CONCLUDING REMARKS

- Used on a case study
- Representation of the distributed aspects in VDM-RT models
- Ensure same semantically meaning between model and implementation
- Initialization process of a DS
- Provide guidelines in order to support the code generation process



FUTURE WORK

- Research relationship between Real-Time aspects in VDM-RT models and the actual implementation
- Support code generation for a programming language suitable for Real-Time
- Research other suitable technologies for dynamic distributed systems

VISION

- 1 Year
  - Update VDM-RT for better modelling for distributed systems
- 5 Years
  - Enable code generation for target hardware (Software-in-the-Loop)
  - Use some of the extensions in industry case studies (INTO-CPS)
- 10 years
  - Used during different aspects of development cycle.

AU

AARHUS
UNIVERSITET