ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

ModHel'X

A component-oriented approach to multi-formalism modeling

Cécile Hardebolle, Frédéric Boulanger

2 October 2007

Cécile HARDEBOLLE
Supélec – Computer Science Department
✉ cecile.hardebolle@supelec.fr

SYSTEM@TIC PARIS-REGION Pôle de compétitivité

Agenda

1. Context, existing approaches & motivations
2. ModHel'X: underlying concepts
3. The coffee machine example
4. Discussion & conclusion

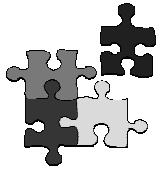
Introduction

- Context
  - Heterogeneous systems: software/hardware, digital/analog, IPs…
  - Multiple modeling formalisms: level of refinement, aspect, domain…

Русский

精忠

- Objective: having a global model of the designed system all along the design cycle
  - Maximize model reuse, facilitate and optimize designers collaboration
  - Simulation, code generation, verification, validation, tests



Multi-formalism modeling =
allow the use of several modeling languages in a model

- Main issues
  - Describe the semantics of a modeling language precisely
  - Define the semantics of a combination of modeling languages in a model

Existing approaches & motivations

- Defining the semantics of modeling languages

[Kermeta] ▸ UML meta-model + execution operations (imperative semantics)

[PtolemyII] ▸ Fixed component-oriented abstract syntax
+ Model of Computation (MoC)

Existing approaches & motivations

- Defining the semantics of modeling languages

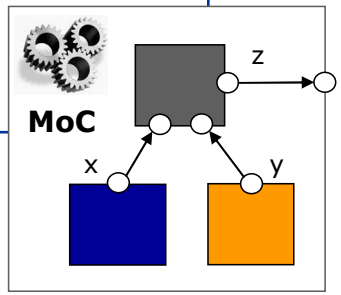
[Kermeta] UML meta-model + execution operations (imperative semantics)

[PtolemyII] Fixed component-oriented abstract syntax

+ Model of Computation (MoC)

Set of rules that define the behavior of the model by combining the behaviors of its components

= "way of interpreting the model"



Existing approaches & motivations

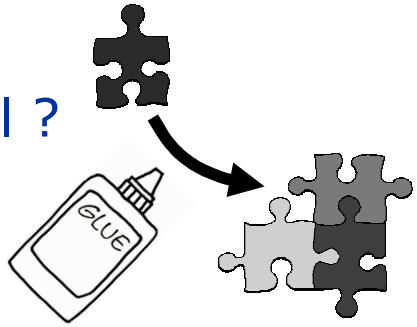
- Defining the semantics of modeling languages

[Kermeta] ▸ UML meta-model + execution operations (imperative semantics)

[PtolemyII] ▸ Fixed component-oriented abstract syntax

+ Model of Computation (MoC)

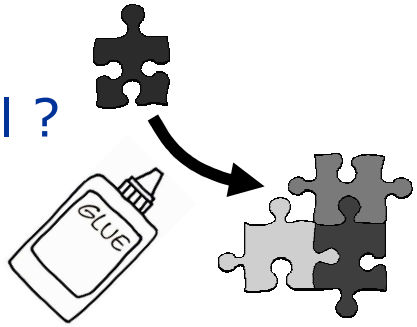
➡ How to “glue” heterogeneous parts of a model ?



Existing approaches & motivations

- Defining the semantics of modeling languages
  - [Kermeta] UML meta-model + execution operations (imperative semantics)
  - [PtolemyII] Fixed component-oriented abstract syntax
    + Model of Computation (MoC)

➡ How to "glue" heterogeneous parts of a model ?

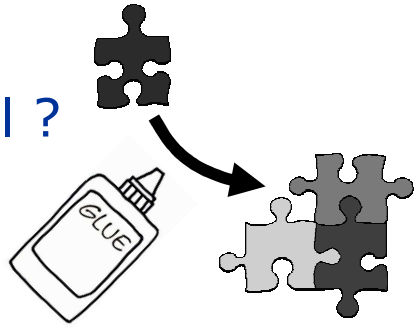


- Combining modeling languages in a model
  - Transformation toward a union meta-model
  - [ATOM^3] Transformation toward one of the modeling languages
  - [PtolemyII] Hierarchical layers using different Models of Computation (MoCs)

Existing approaches & motivations

- Defining the semantics of modeling languages
  - [Kermeta] UML meta-model + execution operations (imperative semantics)
  - [PtolemyII] Fixed component-oriented abstract syntax
    + Model of Computation (MoC)

➡ How to "glue" heterogeneous parts of a model ?



- Combining modeling languages in a model
  - Transformation toward a union meta-model
  - [ATOM^3] Transformation toward one of the modeling languages
  - [PtolemyII] Hierarchical layers using different Models of Computation (MoCs)

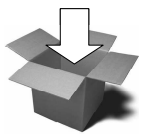
ModHel'X's goal = provide support for the explicit specification of interactions between hierarchical layers using different MoCs

Agenda

1. Context, existing approaches & motivations
2. ModHel'X: underlying concepts
3. The coffee machine example
4. Discussion & conclusion

ModHel'X: main principles

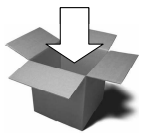
Encapsulation



- Components of a model = black-boxes with well defined interfaces
  - Goal: decouple the internal mechanism of a component from the model in which it is used

ModHel'X: main principles

- Encapsulation



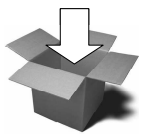
  - Components of a model = black-boxes with well defined interfaces
    - Goal: decouple the internal mechanism of a component from the model in which it is used

- Observation

  - Execution of a model = observations of its behavior = snapshots
    - Triggered by time, environment changes and by components of the model
  - Snapshot = combination of observations of the components of the model according to the MoC
  - Observation of a component (black-box) = update of its interface

ModHel'X: main principles

Encapsulation



- Components of a model = black-boxes with well defined interfaces
  - Goal: decouple the internal mechanism of a component from the model in which it is used

Observation

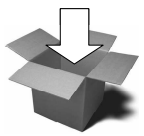
- Execution of a model = observations of its behavior = snapshots
  - Triggered by time, environment changes and by components of the model
- Snapshot = combination of observations of the components of the model according to the MoC
- Observation of a component (black-box) = update of its interface

Hierarchy & delegation

- Behavior of a component = internal model + internal MoC
- Update of a component = update of its internal model
- Semantic adaptation at the border of the component

ModHel'X: main principles

Encapsulation



- Components of a model = black-boxes with well defined interfaces
  - Goal: decouple the internal mechanism of a component from the model in which it is used

Observation

- Execution of a model = observations of its behavior = snapshots
  - Triggered by time, environment changes and by components of the model
- Snapshot = combination of observations of the components of the model according to the MoC
- Observation of a component (black-box) = update of its interface

Hierarchy & delegation

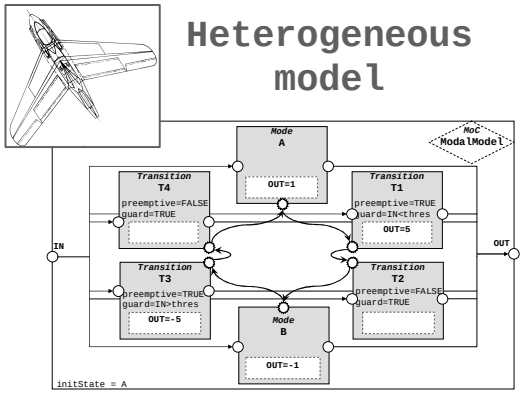
- Behavior of a component = internal model + internal MoC
- Update of a component = update of its internal model
- Semantic adaptation at the border of the component

one MoC by layer + local heterogeneity = reduced complexity

General architecture of ModHel'X

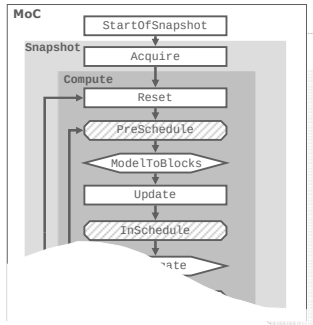
Executable descriptions of each MoC =

Heterogeneous model

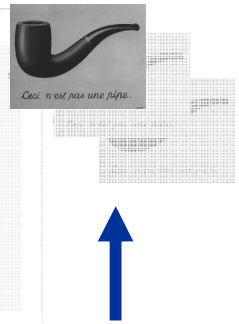


Generic execution engine

Generic model of execution



Semantics of the MoC



Generic meta-model for the representation of the structure of heterogeneous models

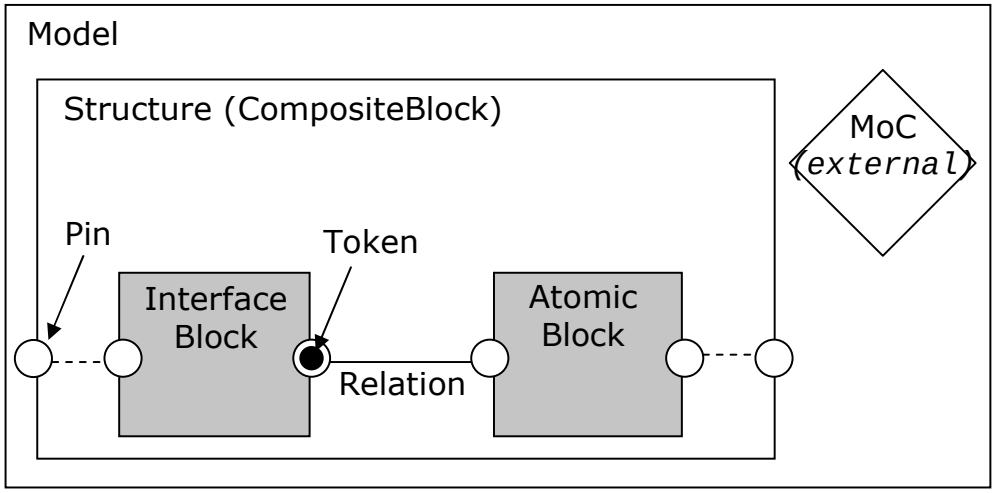
Generic algorithm for executing heterogeneous models

Language for describing

- The operational semantics of a MoC
- The interactions between MoCs

Representing an heterogeneous model

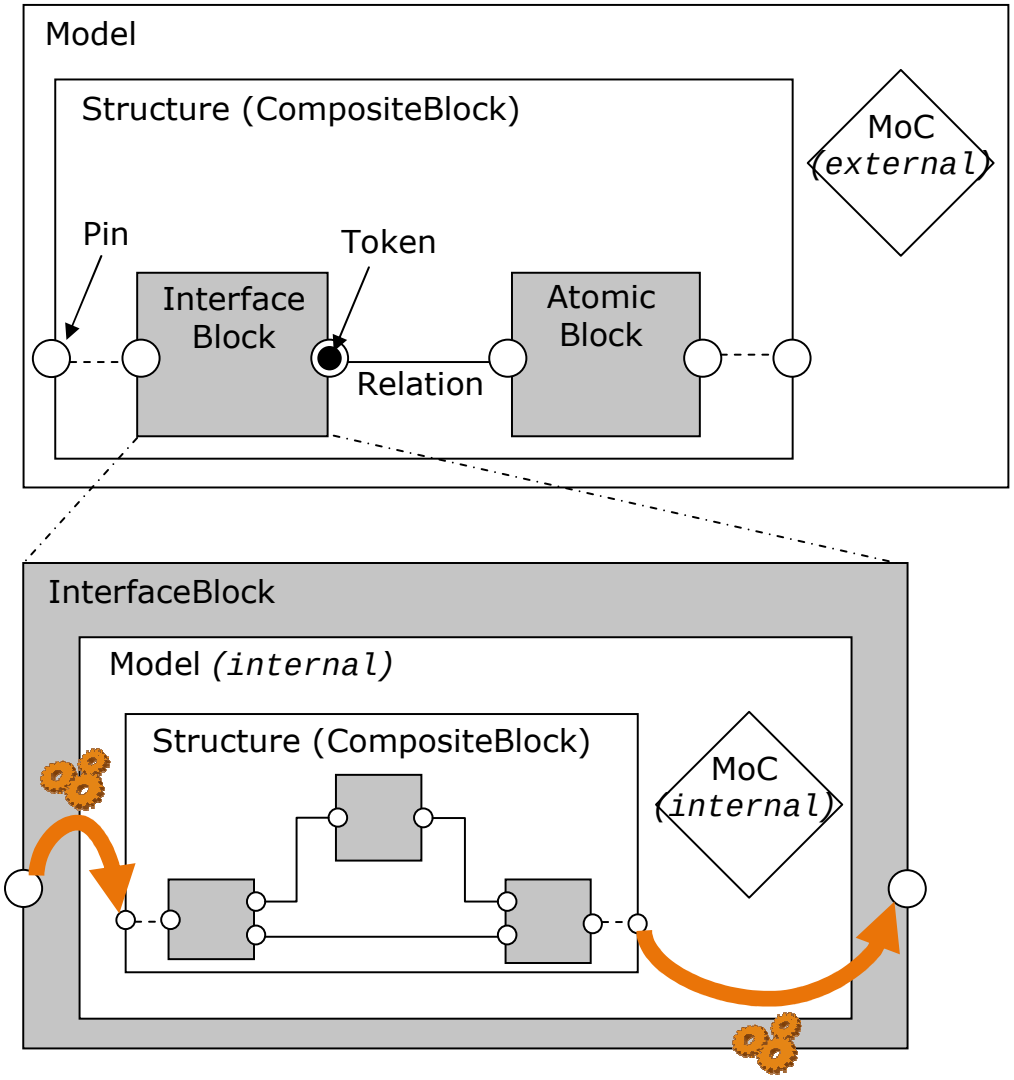
- A set of few basic and generic objects



- Two separate aspects
  - Structural: Blocks, Pins, Relations, Tokens
  - Behavioral: Model of computation
- Specialization of these objects for each MoC

Representing an heterogeneous model

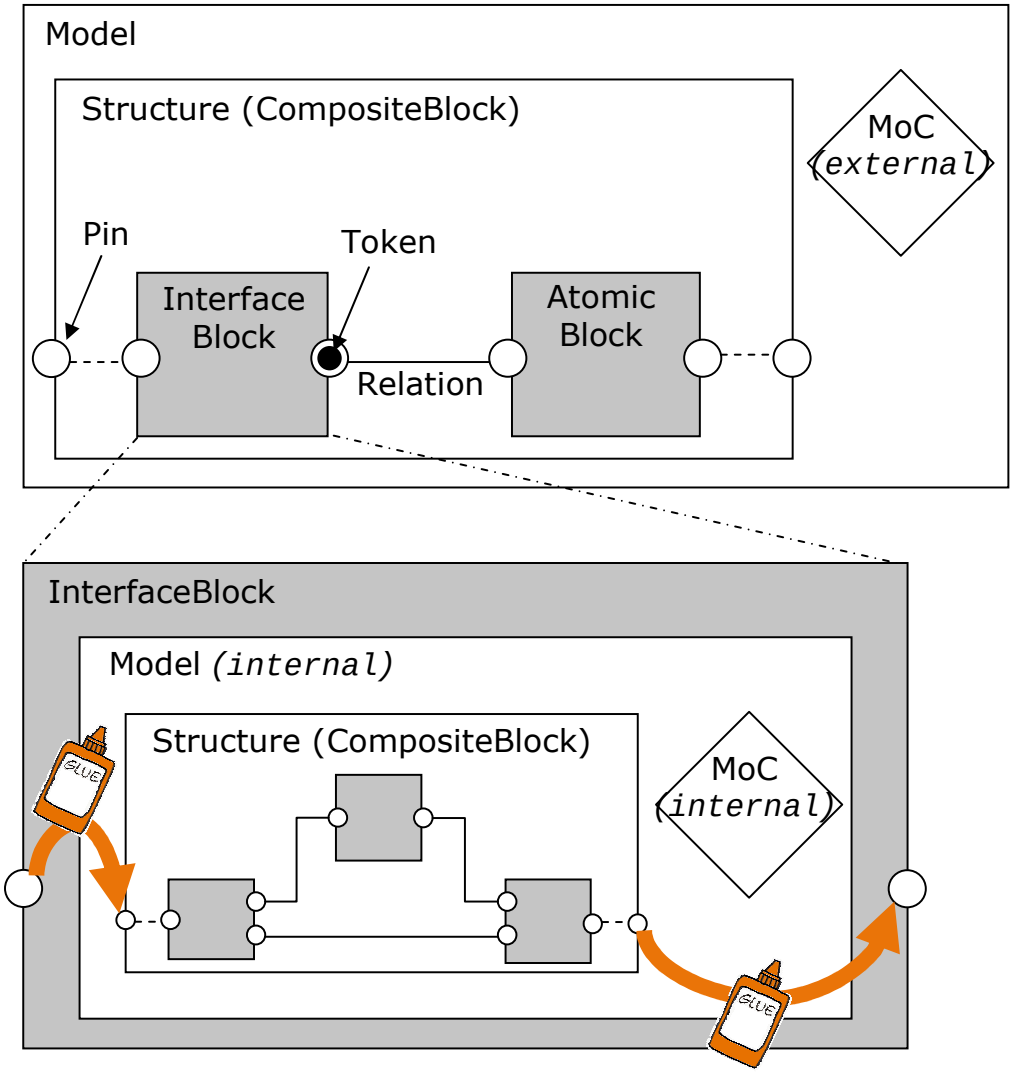
- A set of few basic and generic objects



- Two separate aspects
  - Structural: Blocks, Pins, Relations, Tokens
  - Behavioral: Model of computation
- Specialization of these objects for each MoC
- Hierarchical heterogeneity
  - An InterfaceBlock has an internal model
  - The internal & the external MoCs can be different
  - The InterfaceBlock realizes the semantic adaptation

Representing an heterogeneous model

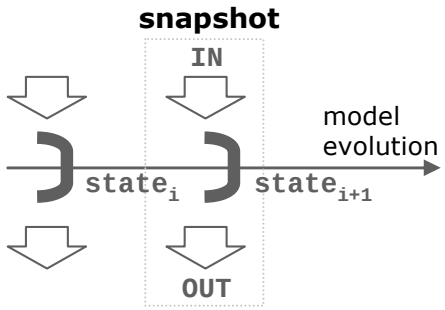
- A set of few basic and generic objects



- Two separate aspects
  - Structural: Blocks, Pins, Relations, Tokens
  - Behavioral: Model of computation
- Specialization of these objects for each MoC
- Hierarchical heterogeneity
  - An InterfaceBlock has an internal model
  - The internal & the external MoCs can be different
  - The InterfaceBlock realizes the semantic adaptation

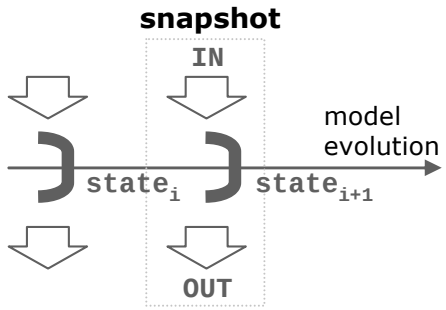
Executing an heterogeneous model

- One execution =
  - Sequence of successive snapshots of the model

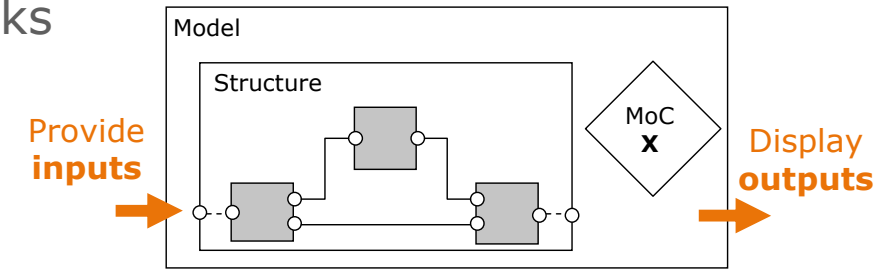


Executing an heterogeneous model

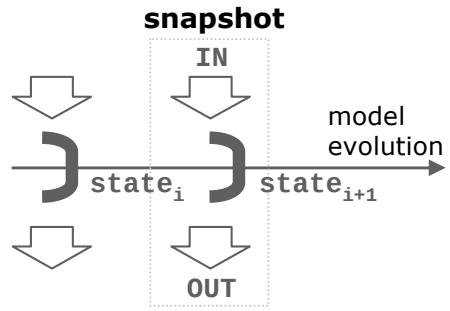
- One execution =
  - Sequence of successive snapshots of the model



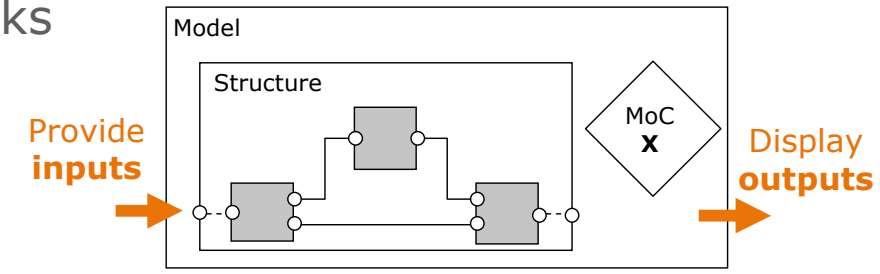
- One snapshot =
  - Causal: $state_i$ + $inputs_i$ ➡ $outputs_i$ + $state_{i+1}$
  - Gradual update of the model blocks
    - Schedule of the block to update
    - Update of the block
    - Propagation of the produced data



Executing an heterogeneous model



- One execution =
  - Sequence of successive snapshots of the model
- One snapshot =
  - Causal: $state_i$ + $inputs_i$ ➡ $outputs_i$ + $state_{i+1}$
  - Gradual update of the model blocks
    - Schedule of the block to update
    - Update of the block
    - Propagation of the produced data

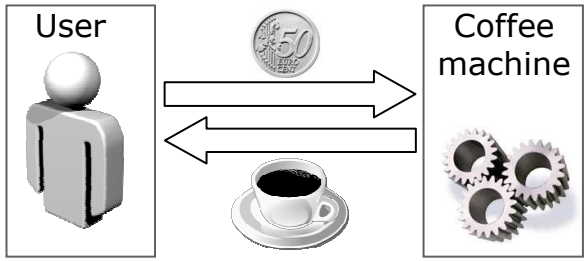


- Generic execution algorithm
  - A set of generic operations
  - Semantics specified using our language
  - Hierarchical execution
    - InterfaceBlocks have special operations in order to adapt the semantics between MoCs

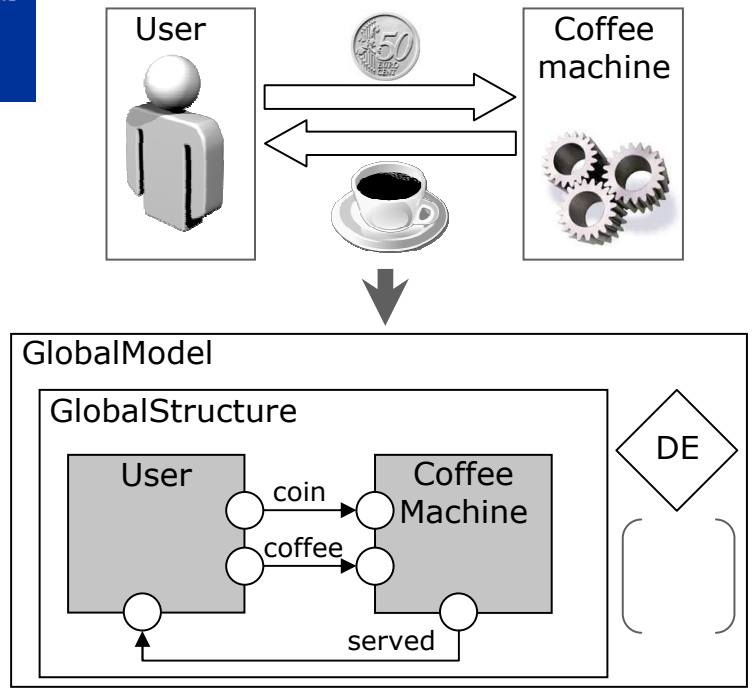
Agenda

1. Context, existing approaches & motivations
2. ModHel'X: underlying concepts
3. The coffee machine example
4. Discussion & conclusion

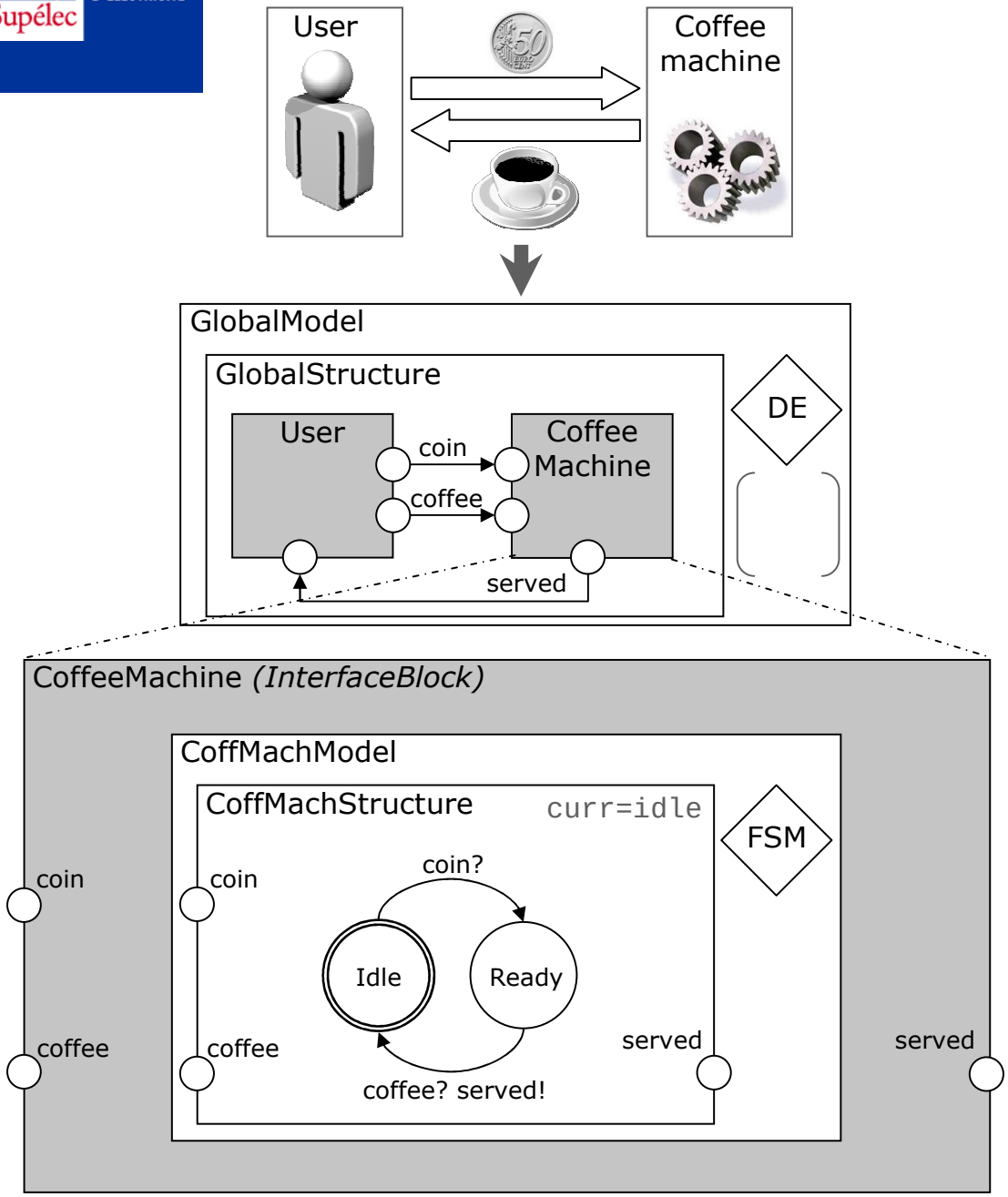
The coffee machine example



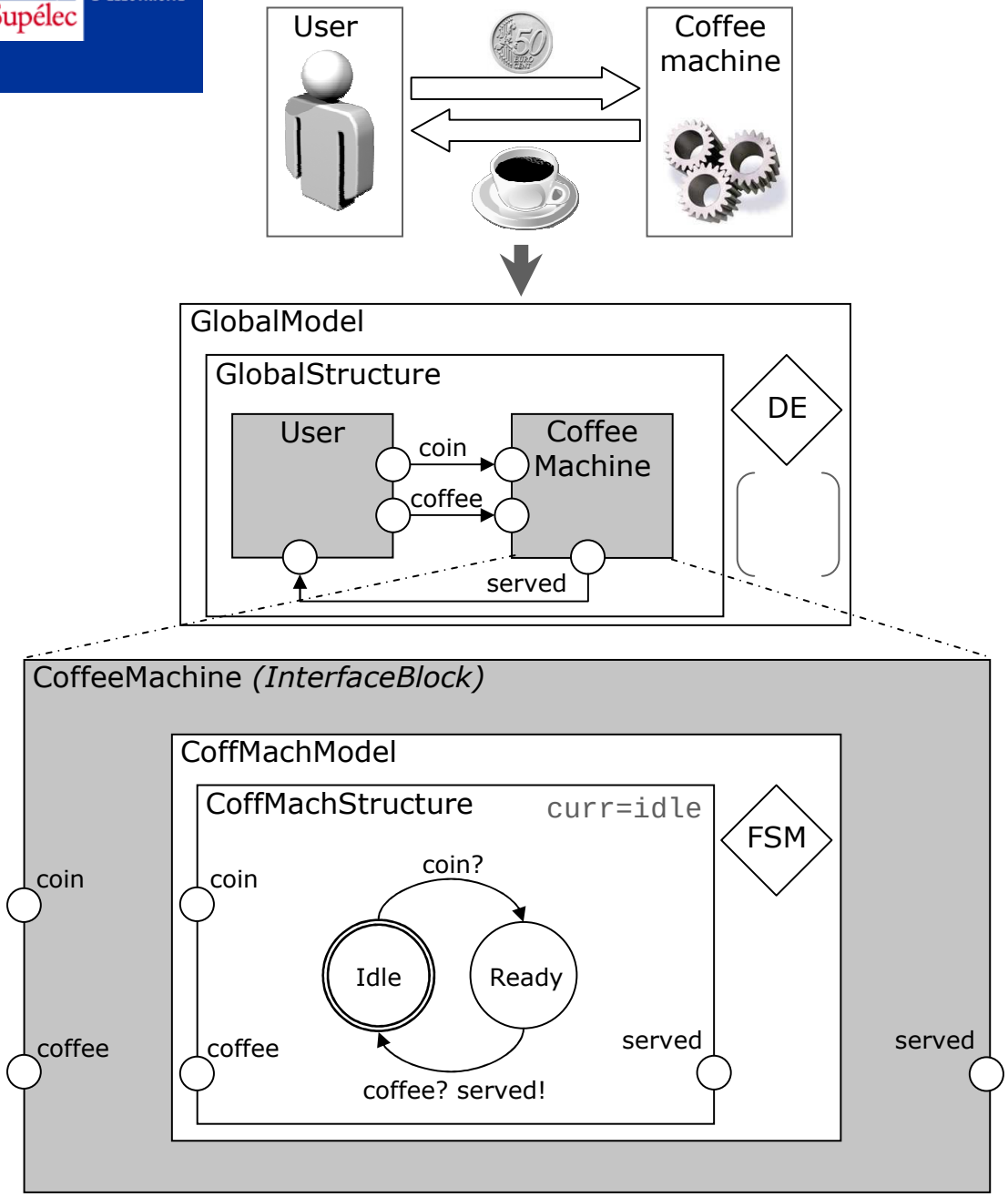
The coffee machine example



The coffee machine example

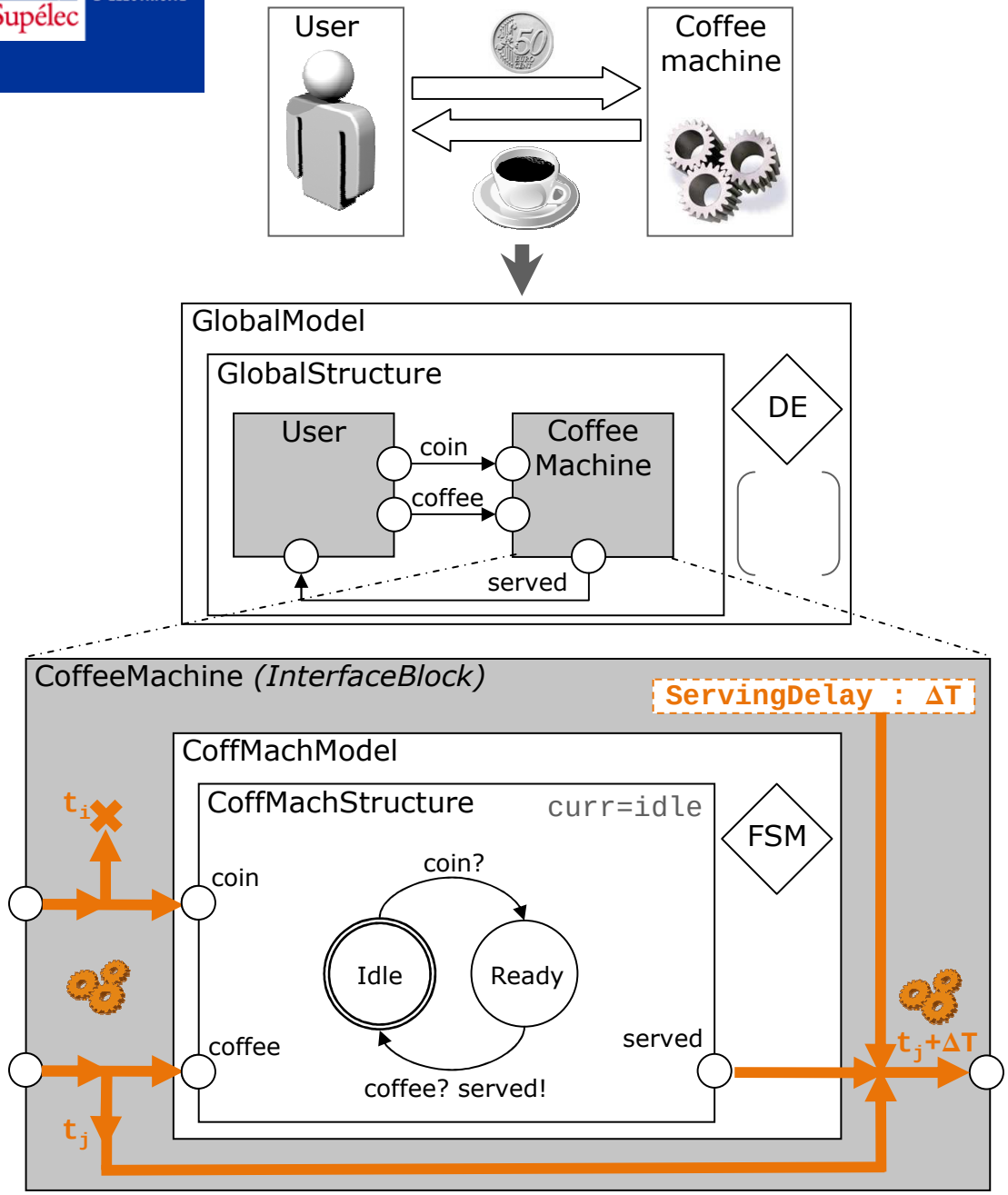


The coffee machine example



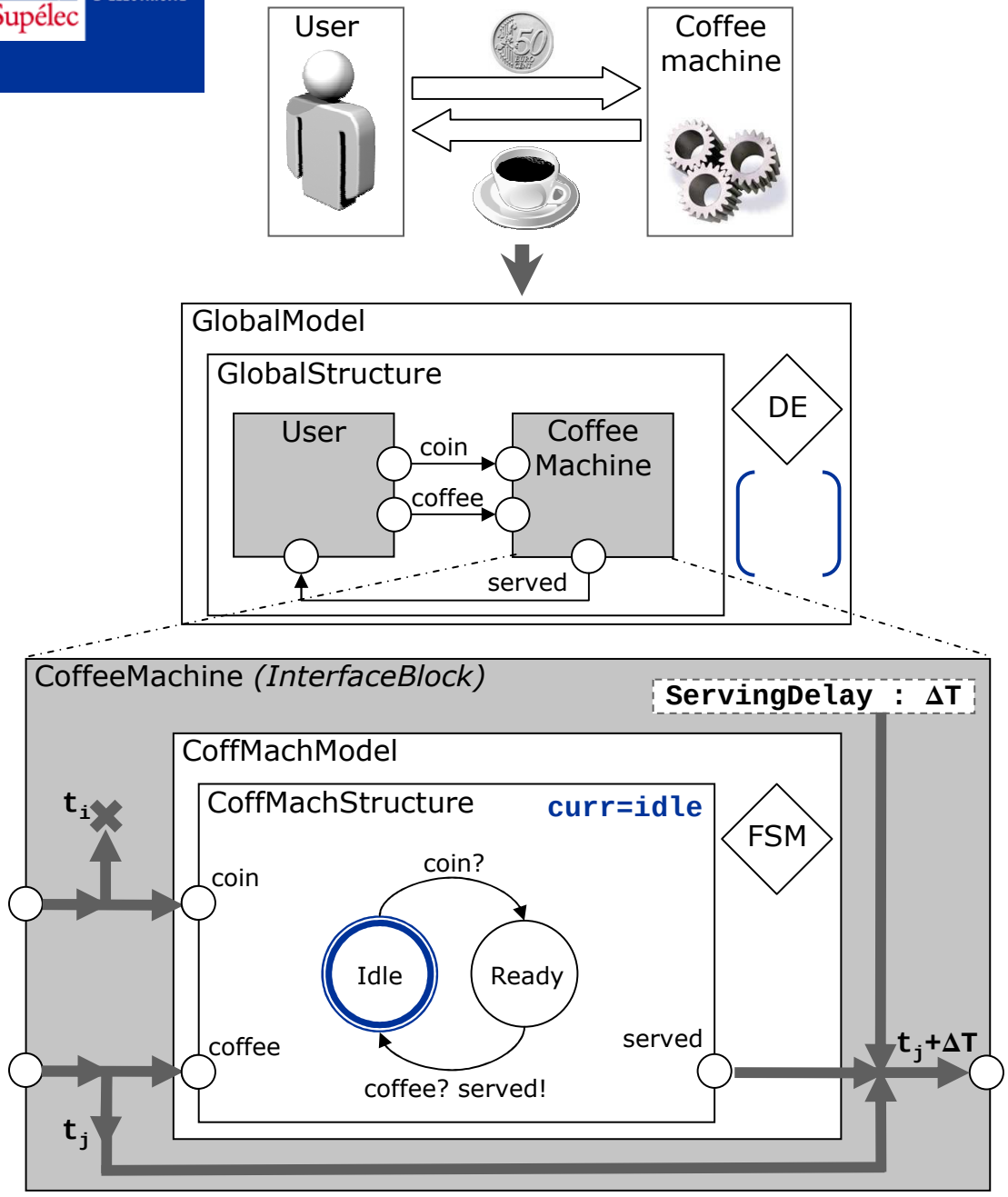
- Semantic adaptation: time "gluing"?

The coffee machine example

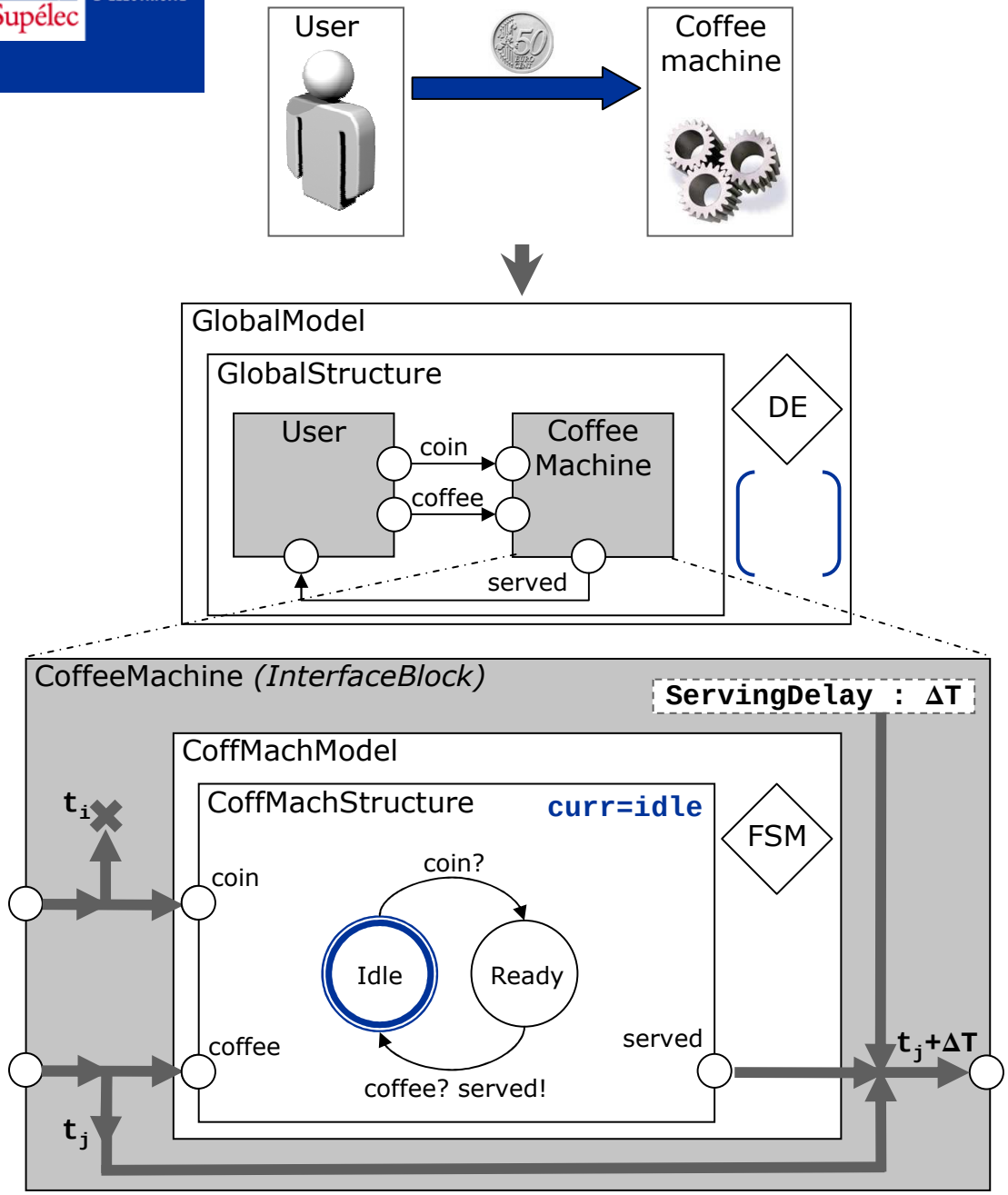


- Semantic adaptation: time "gluing"?
  - in: remove timestamps
  - out: add timestamps
    - ➡ which ones?

Running the model

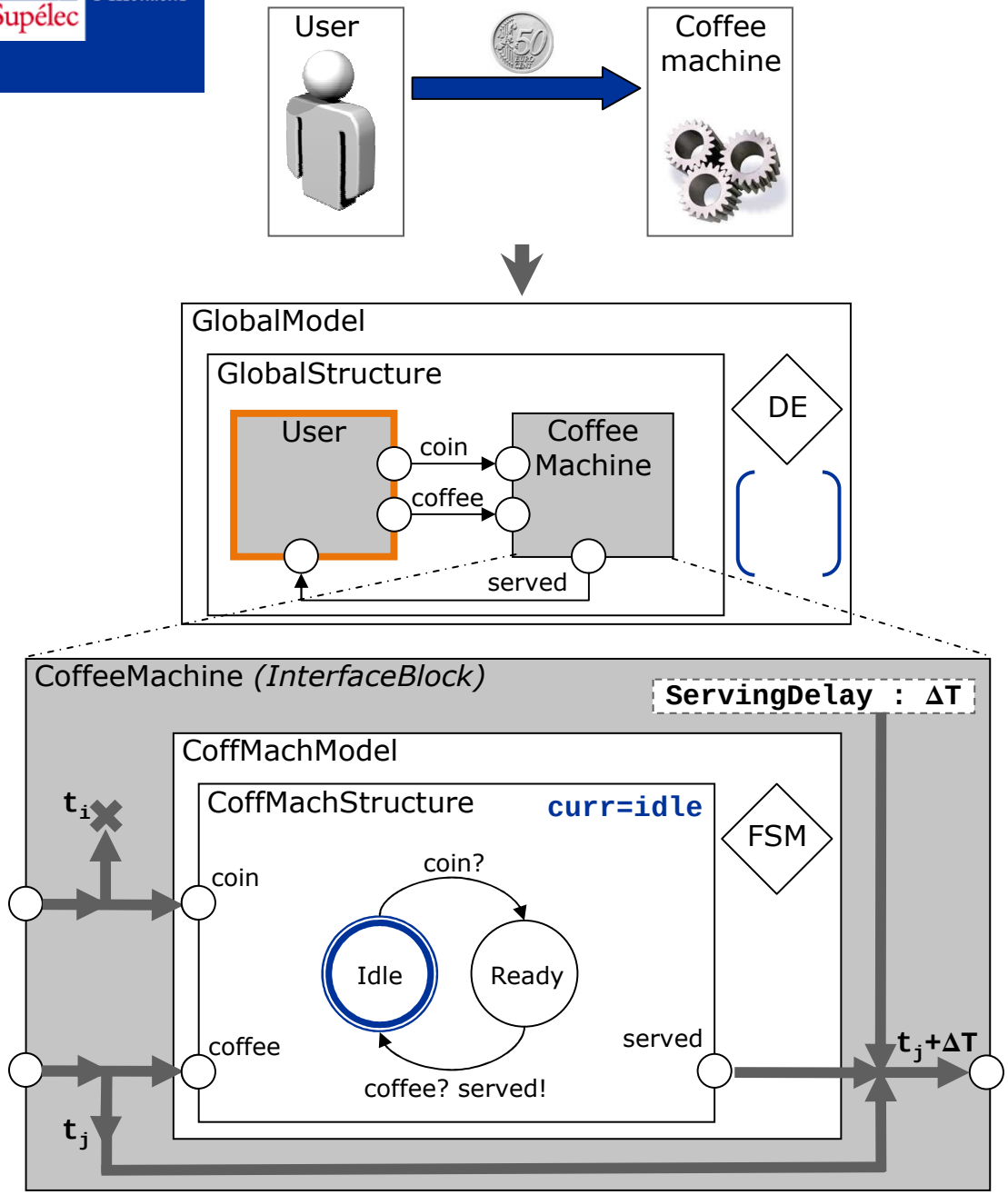


Running the model



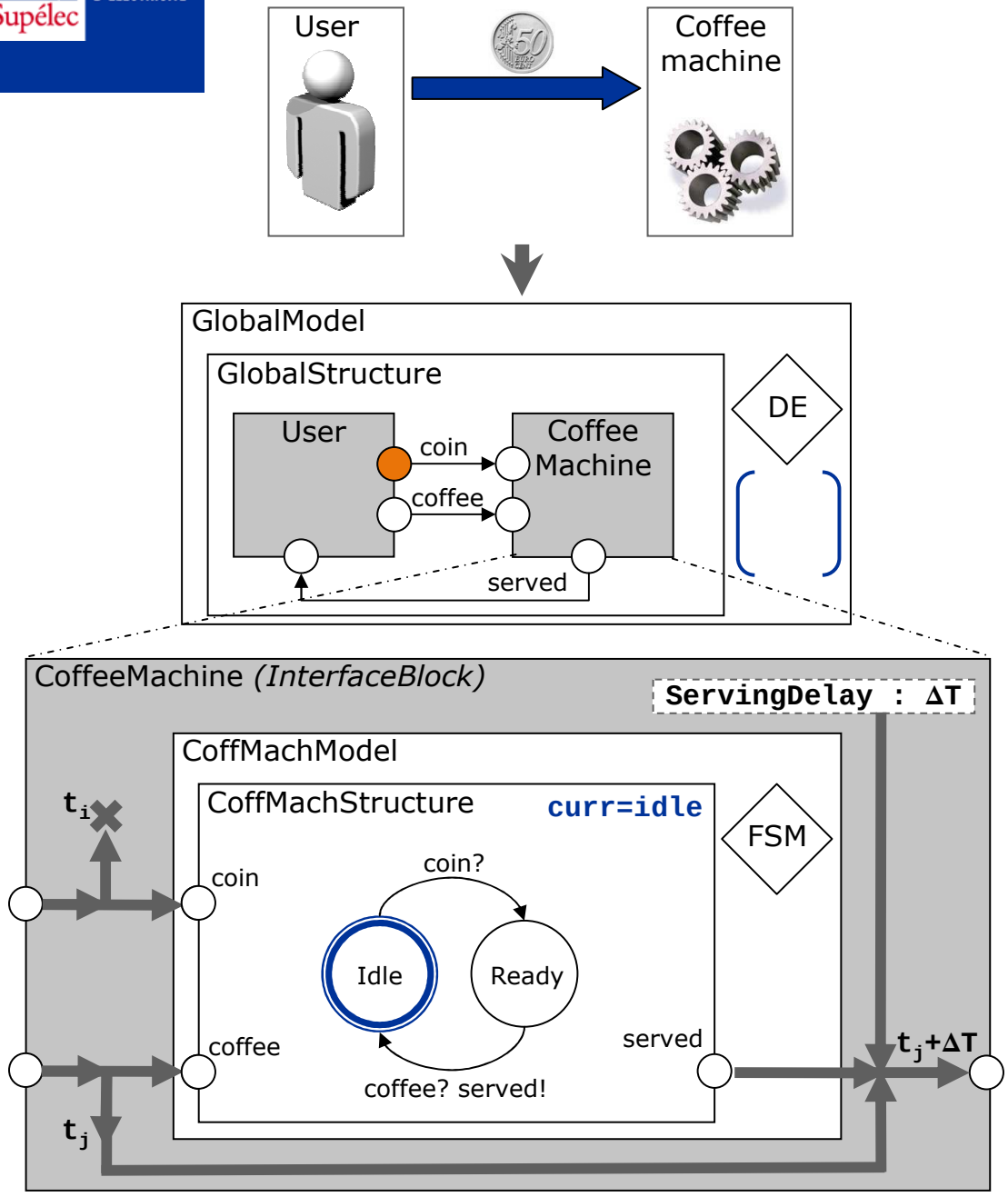
1. The user inserts the coin
   - 1rst snapshot at t_1

Running the model



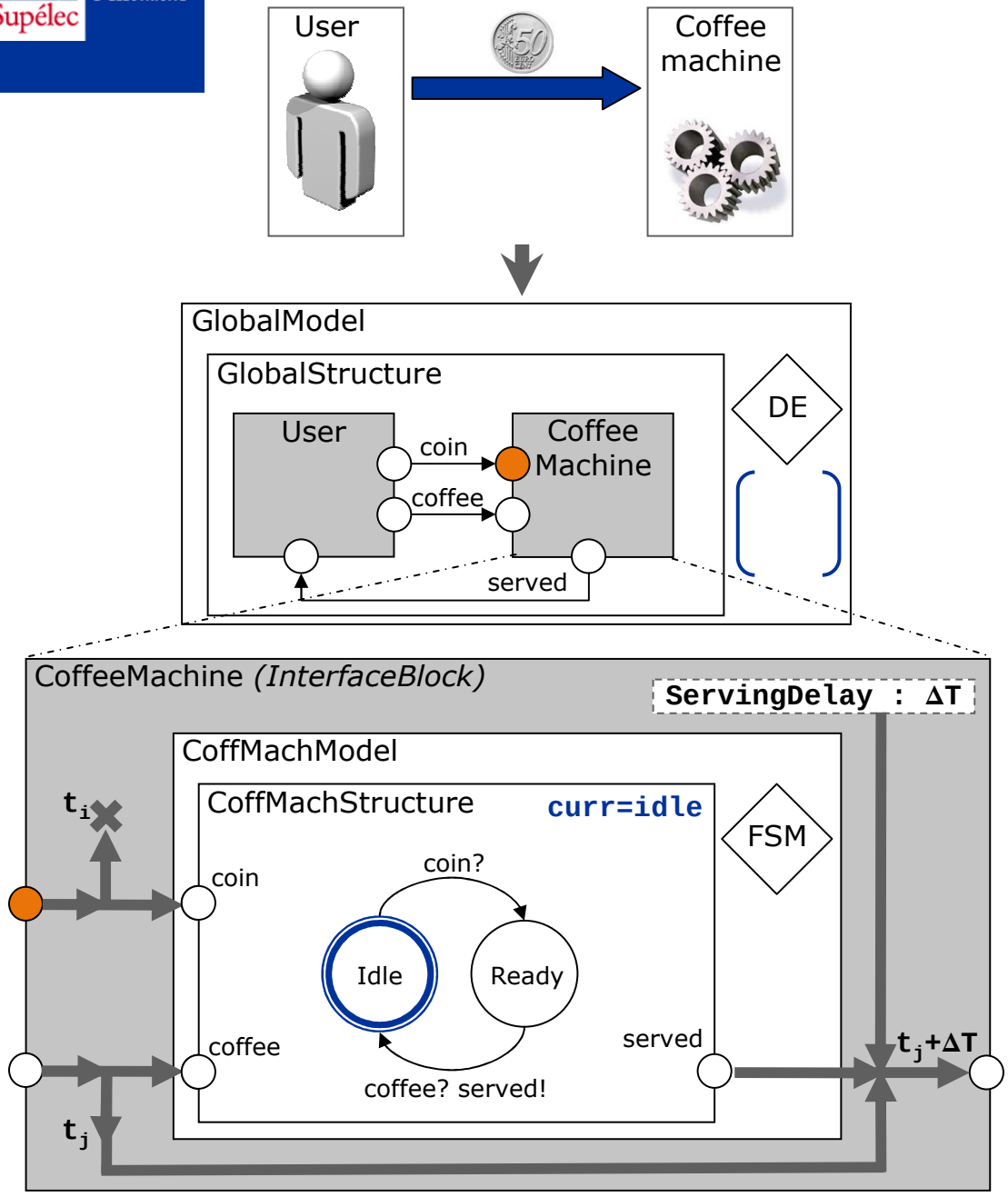
1. The user inserts the coin
   - 1rst snapshot at t_1

Running the model



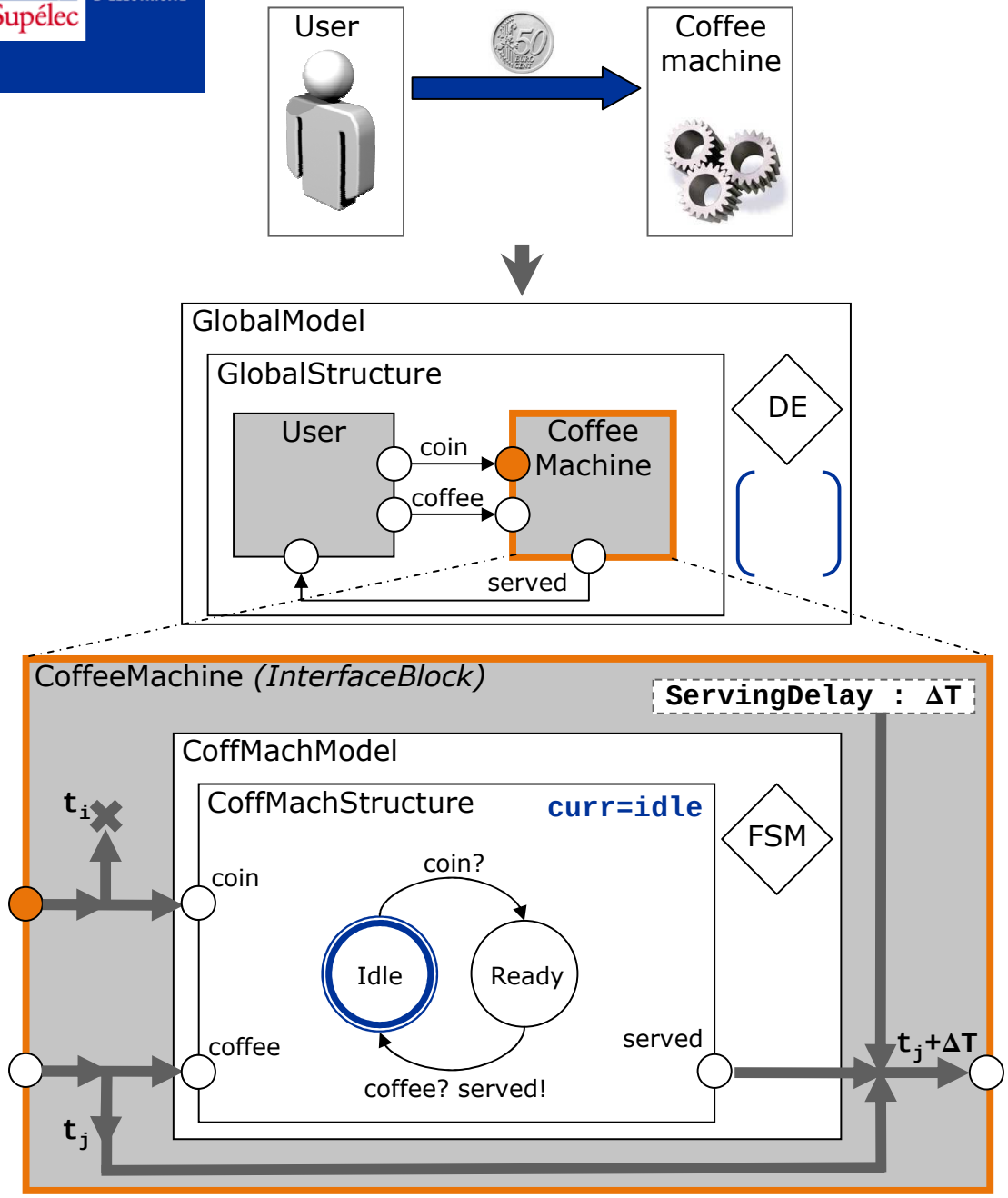
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



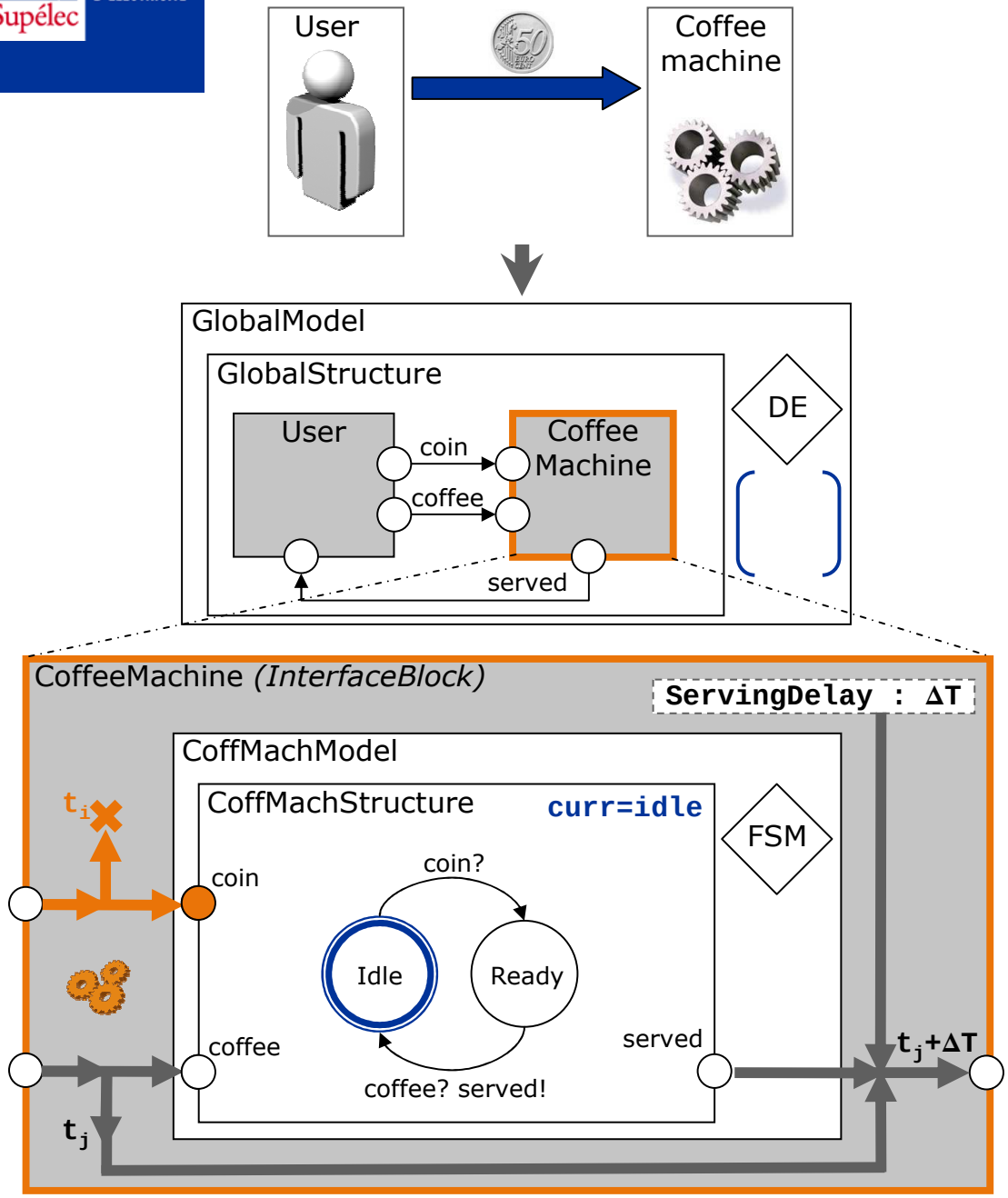
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



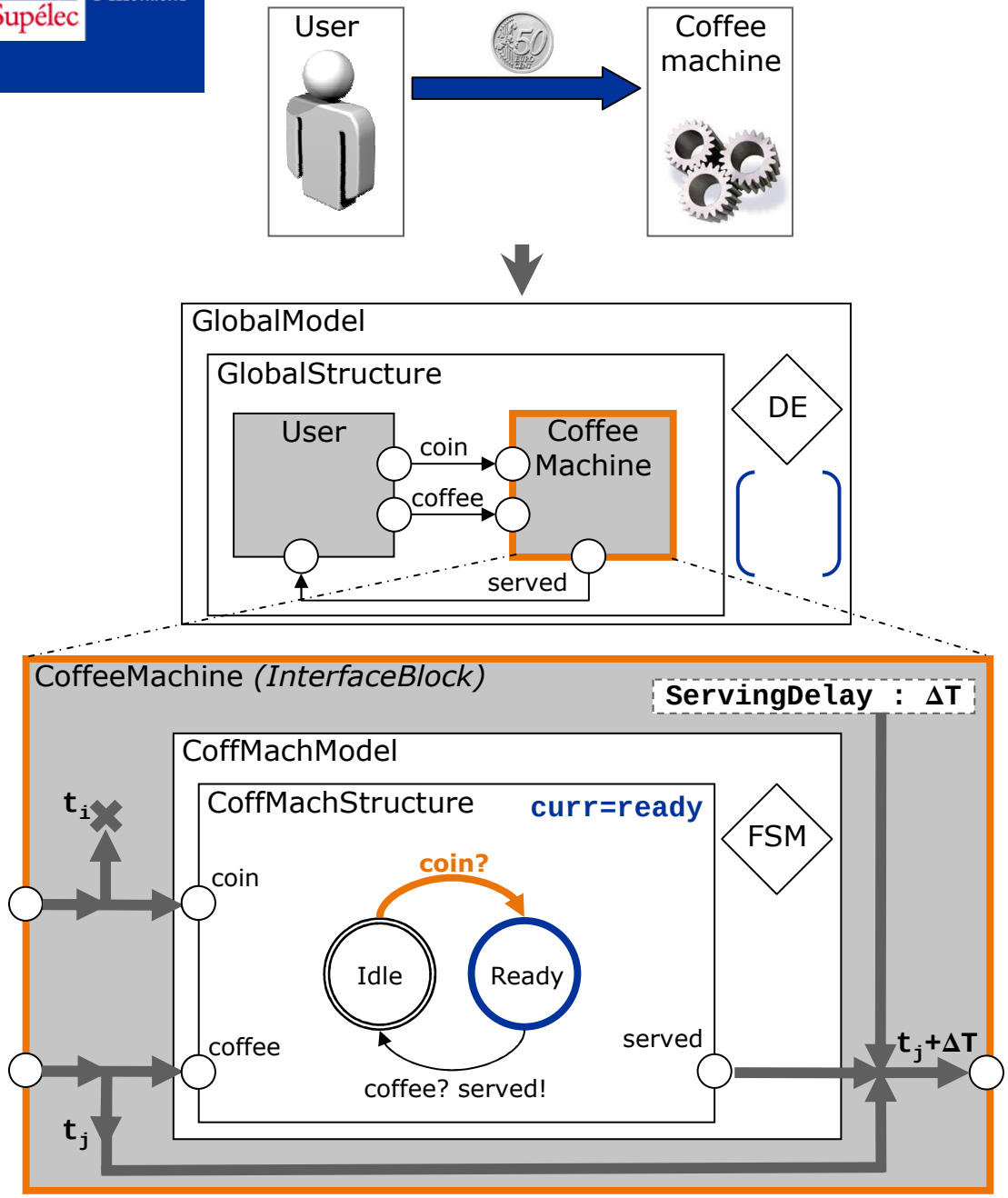
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



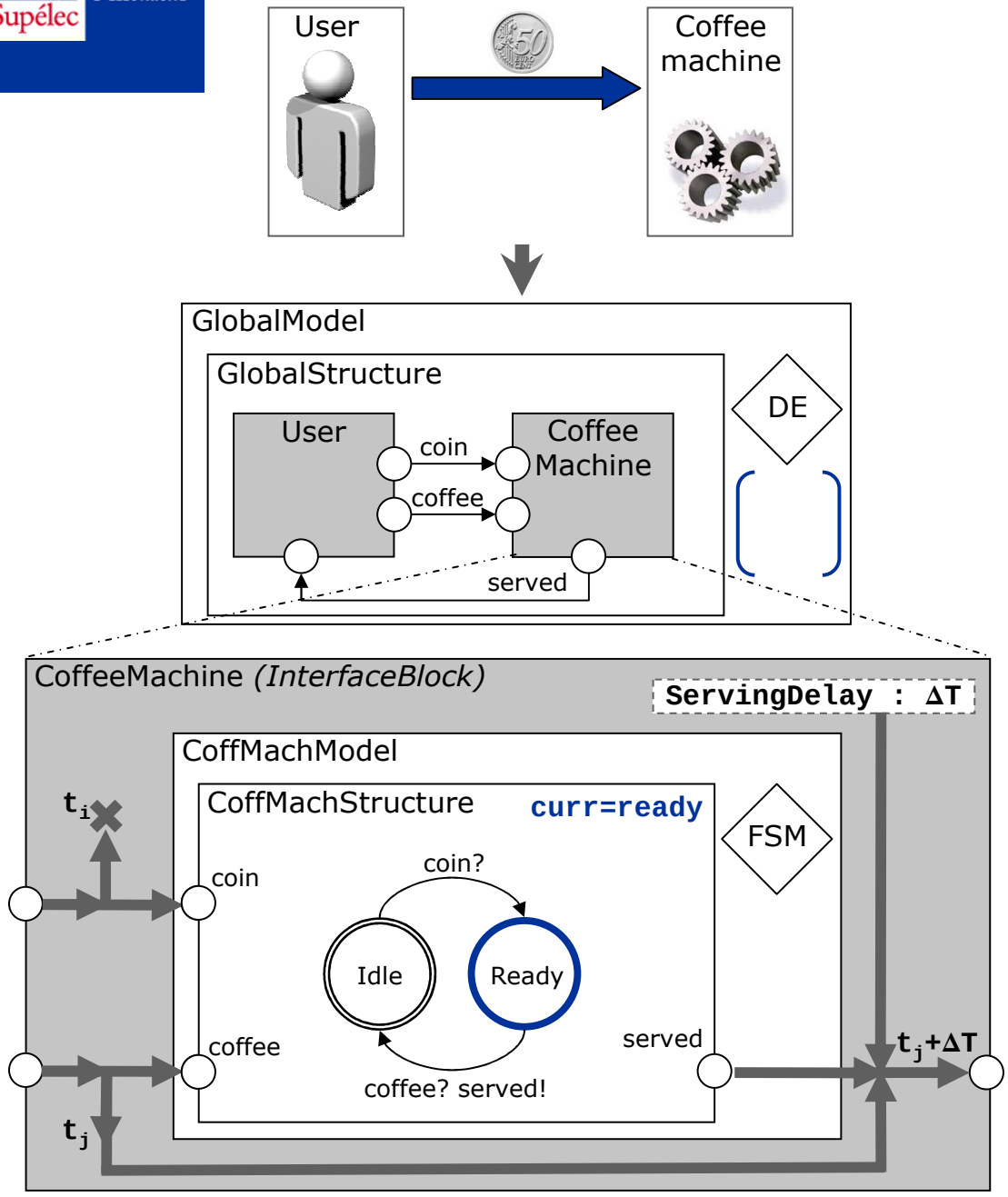
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



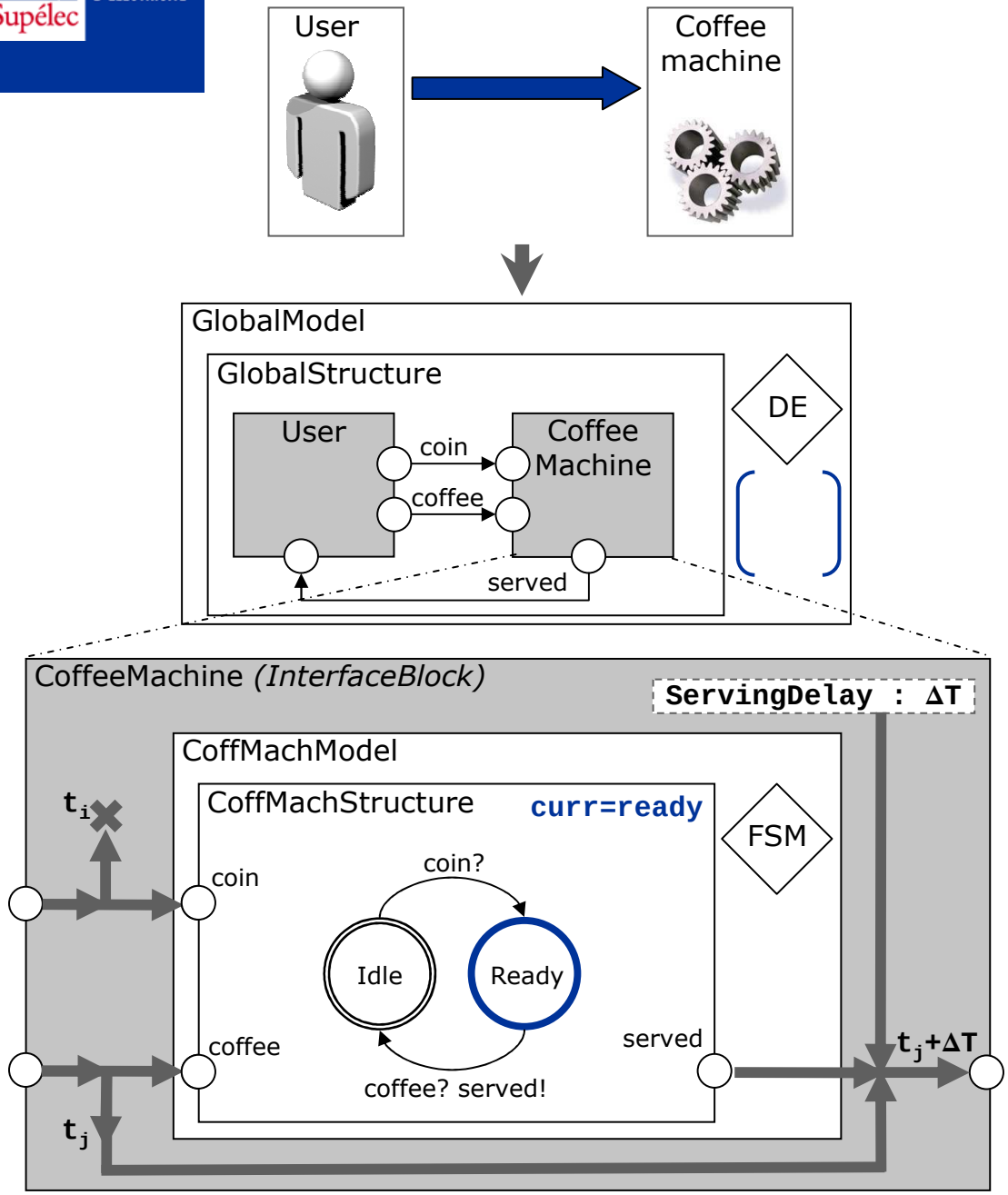
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



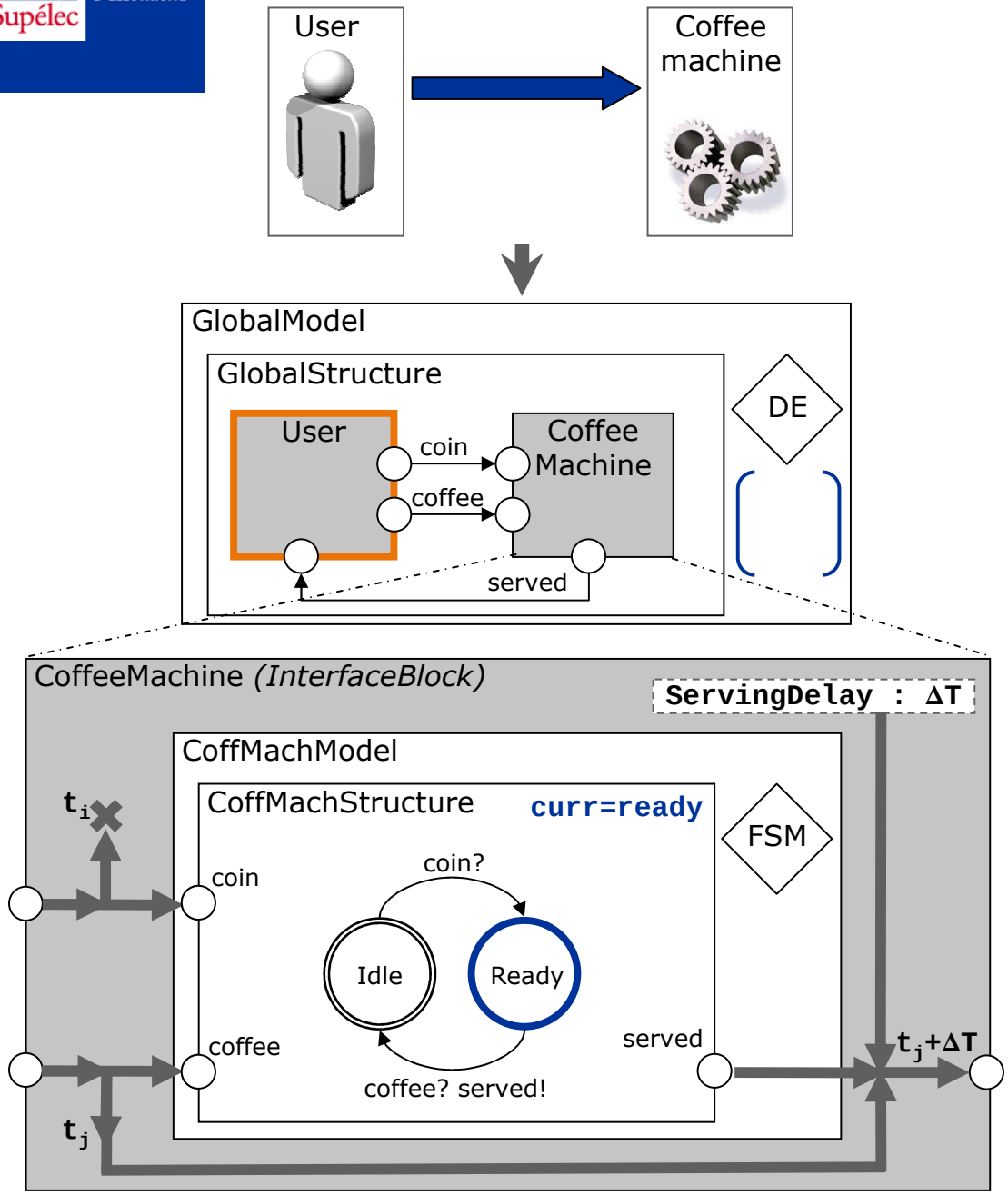
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1

Running the model



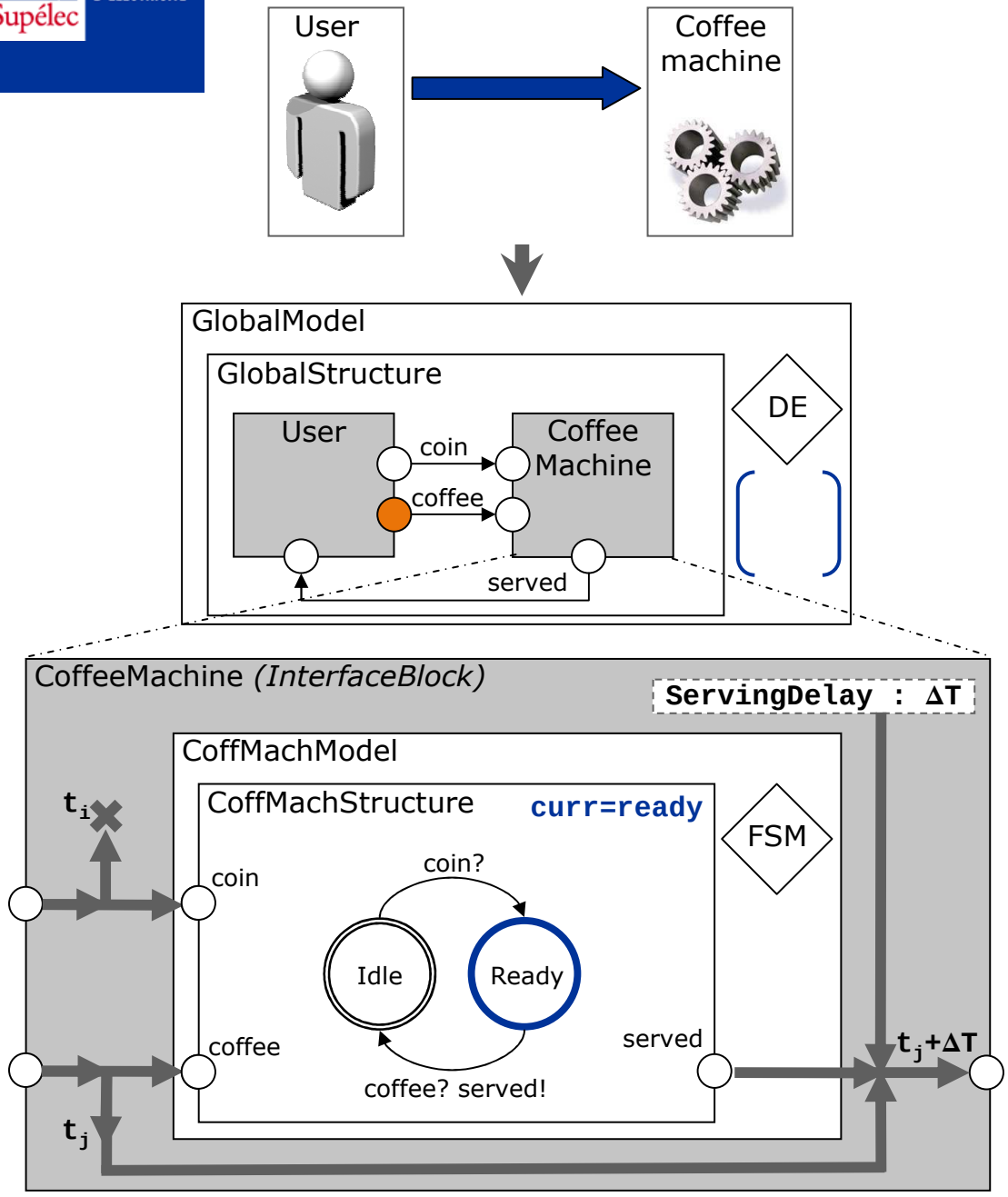
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2

Running the model



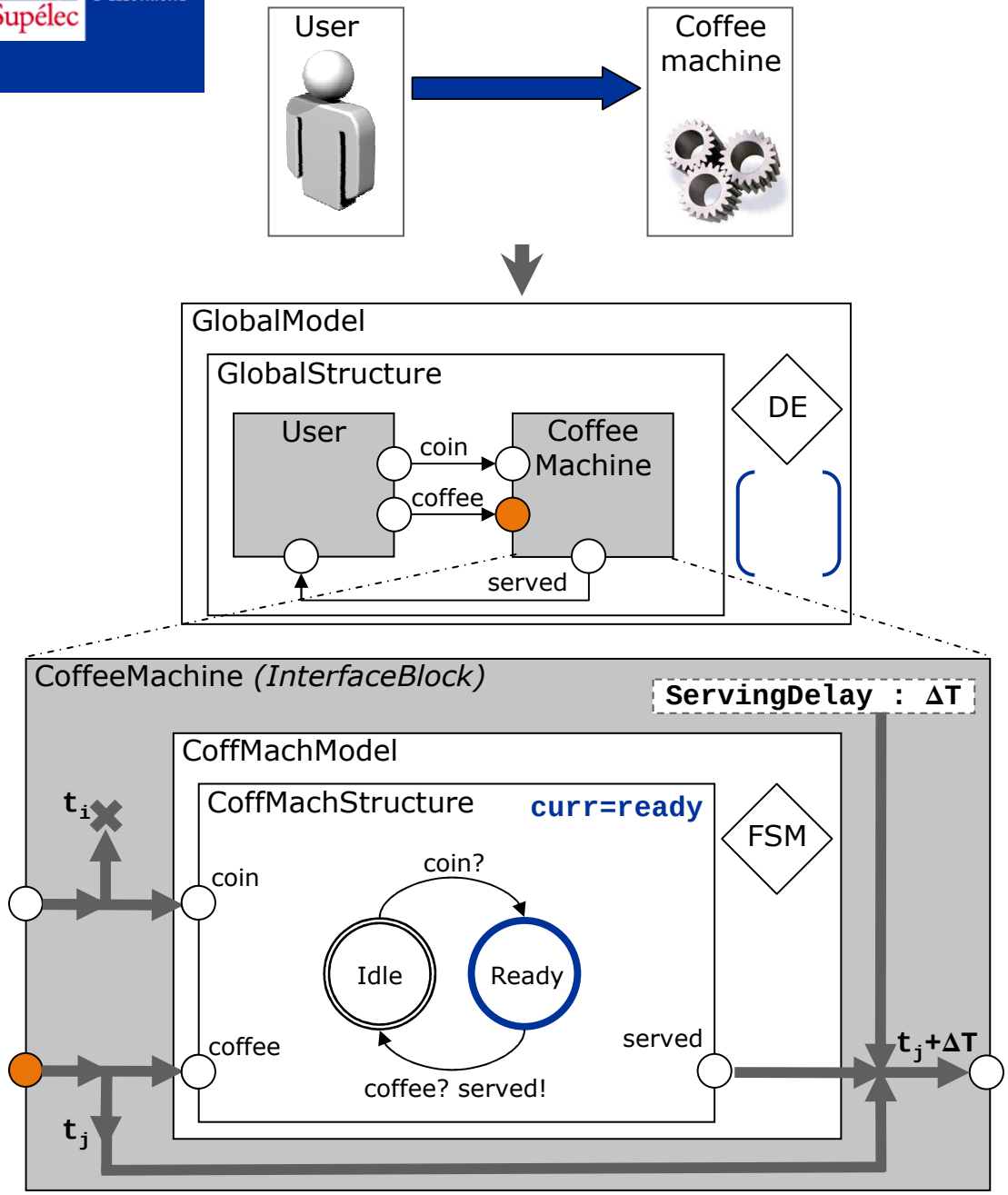
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2

Running the model



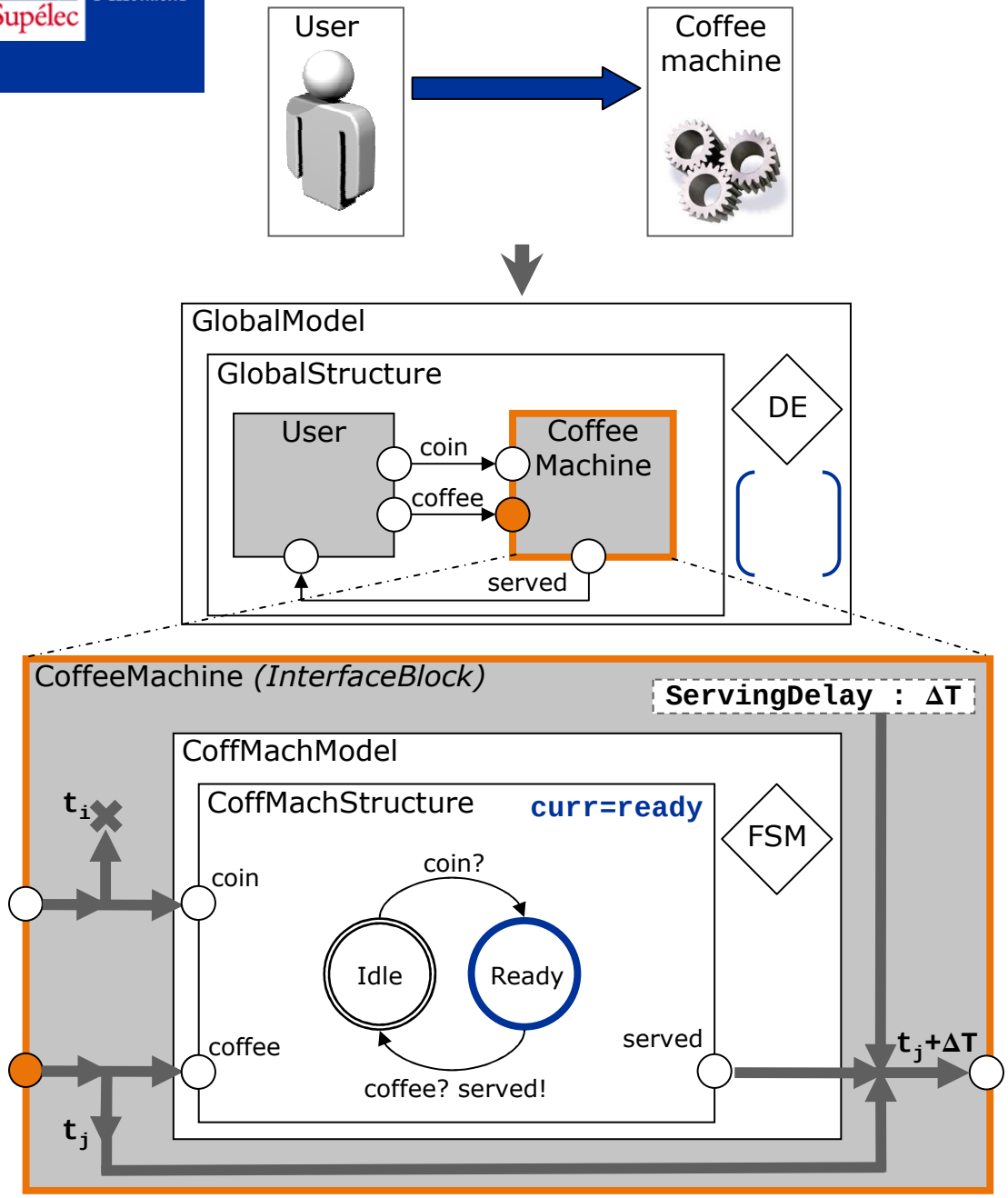
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2

Running the model



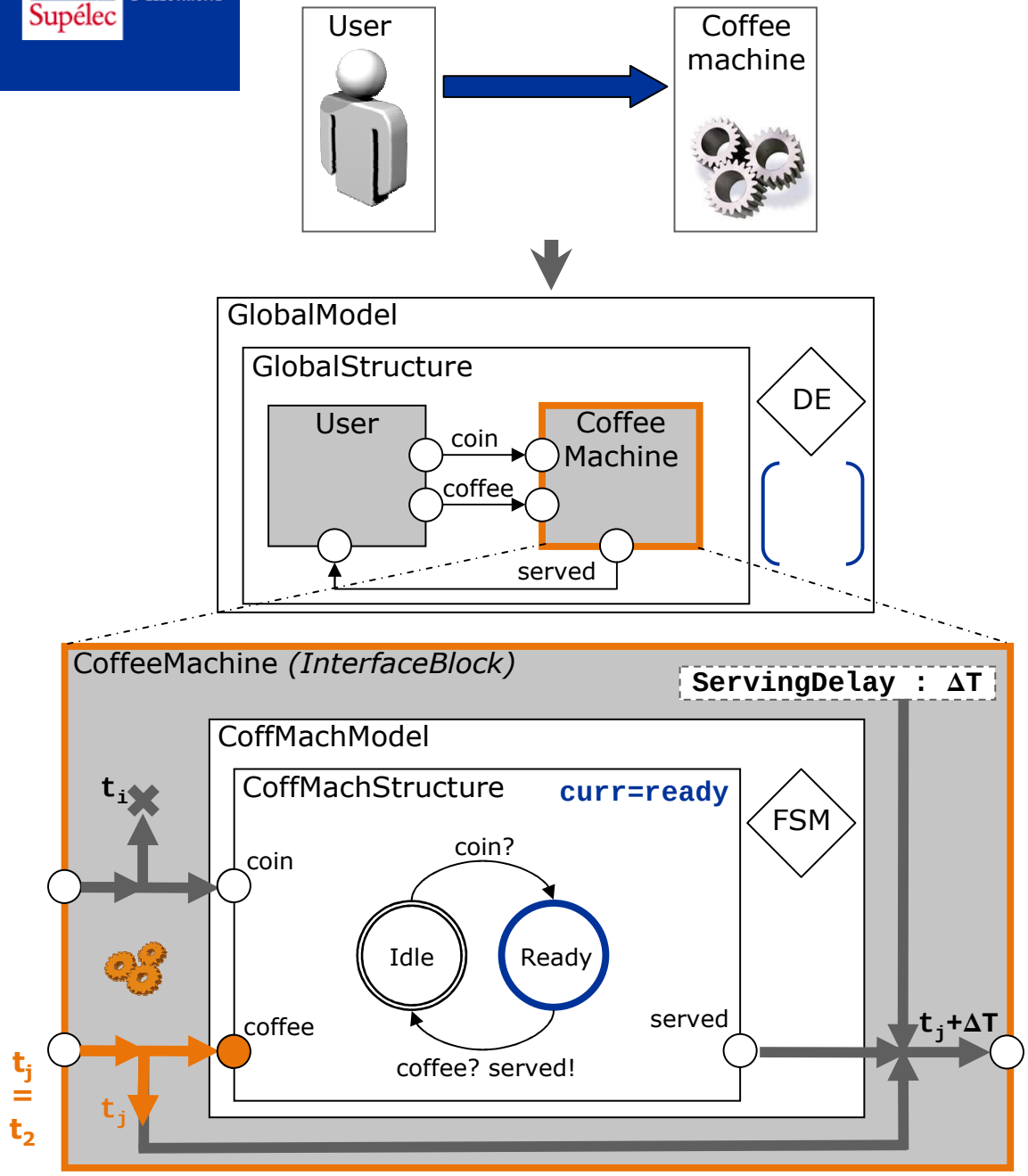
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2

Running the model



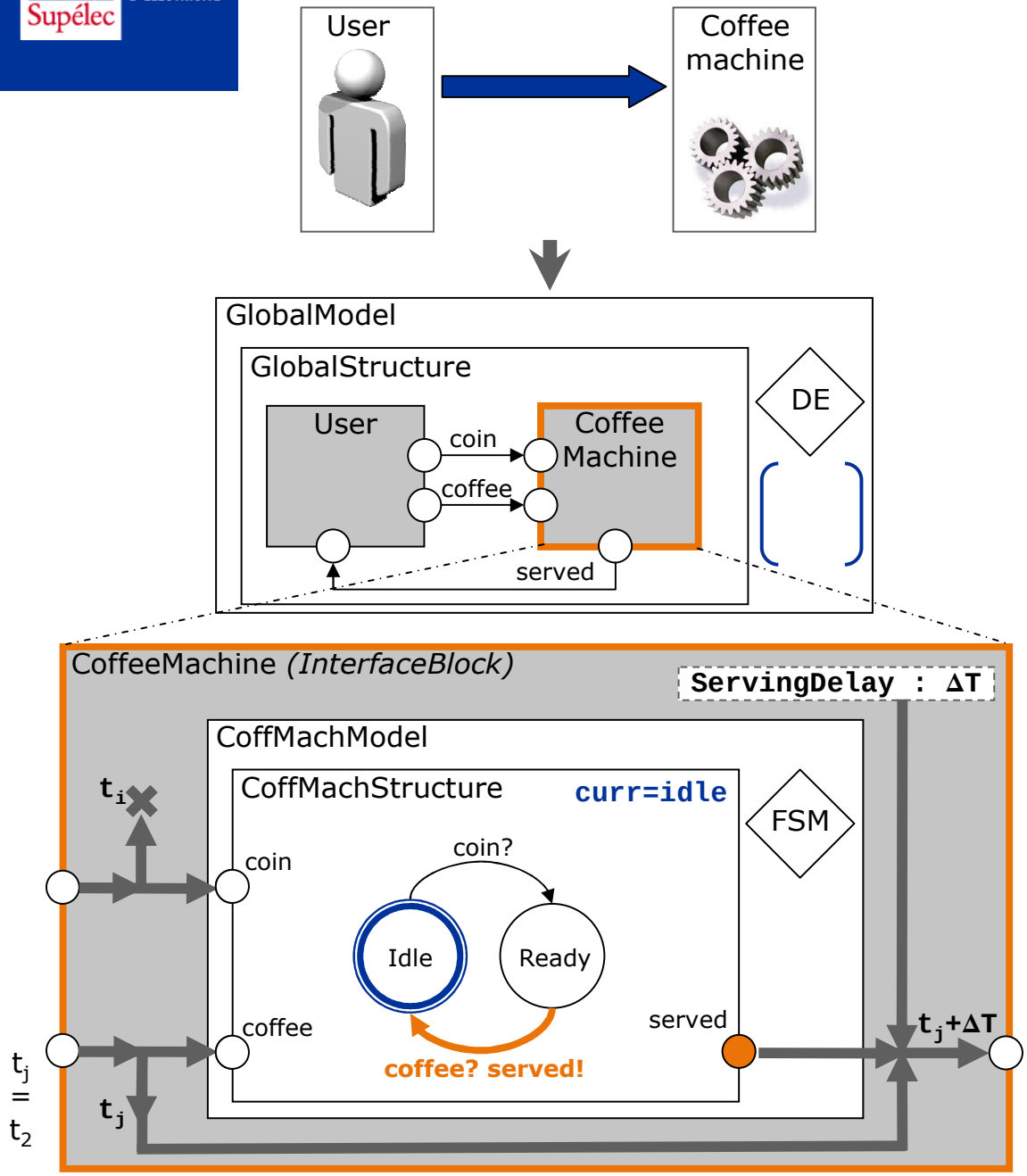
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2

Running the model



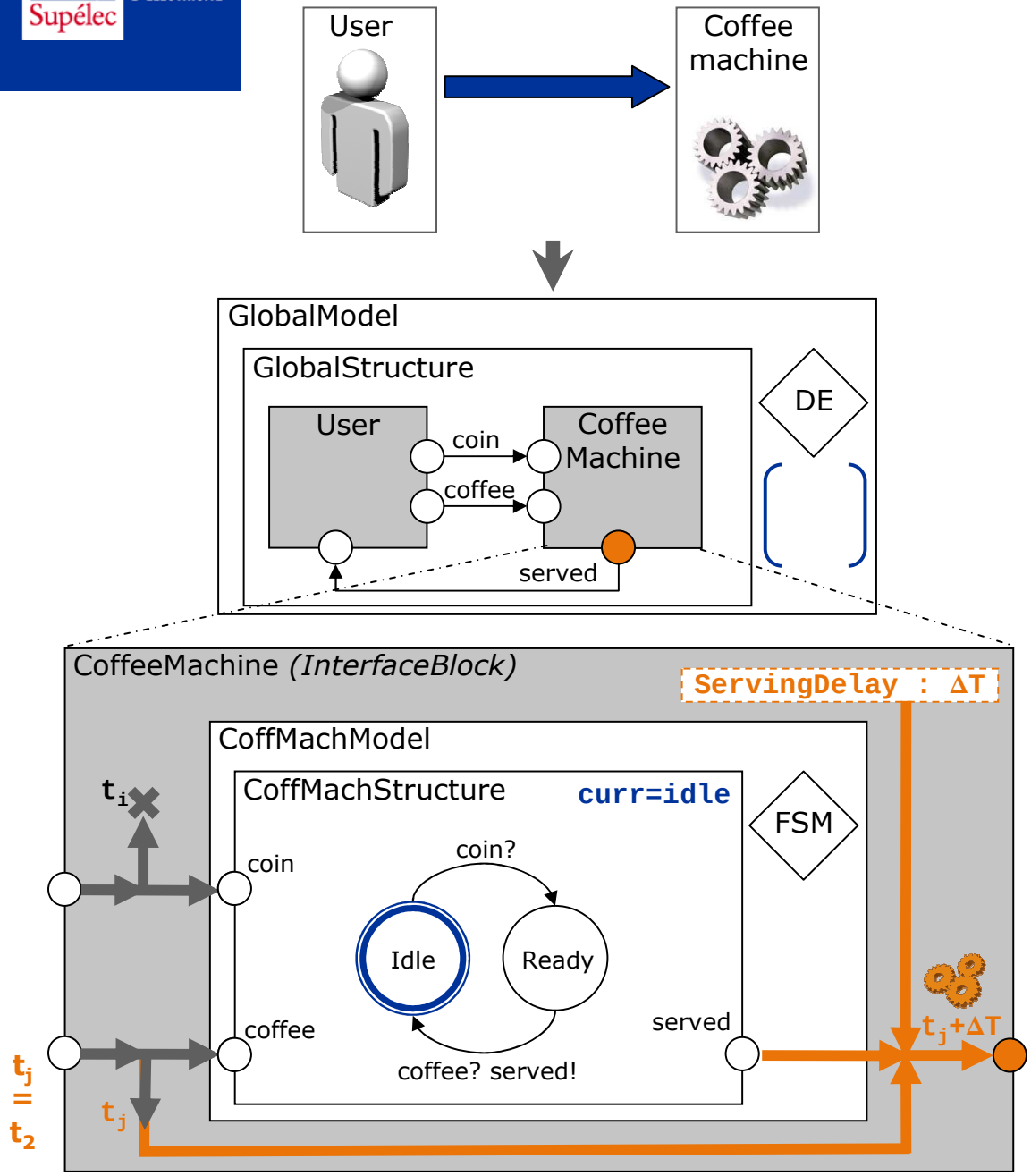
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2

Running the model



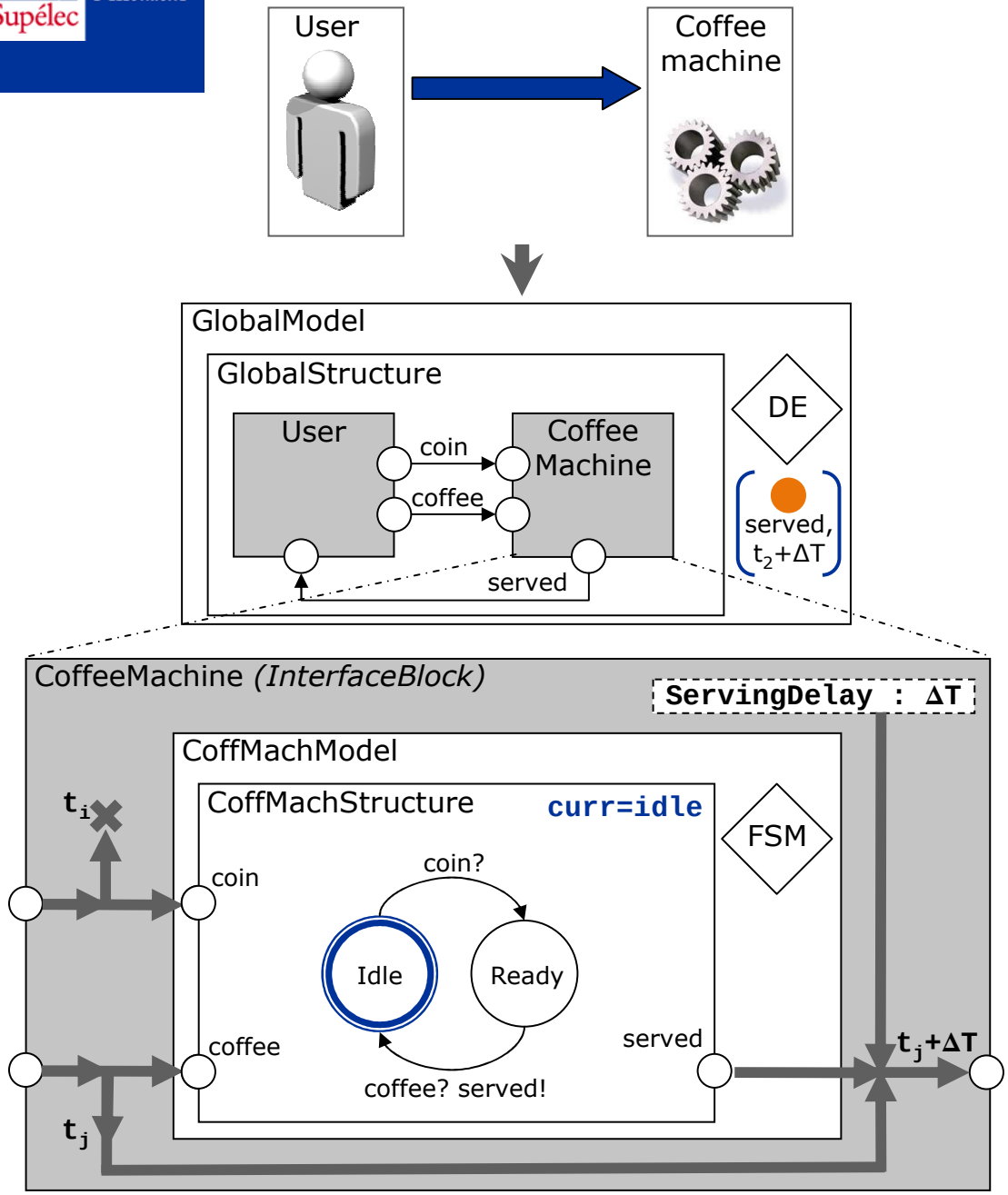
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2

Running the model



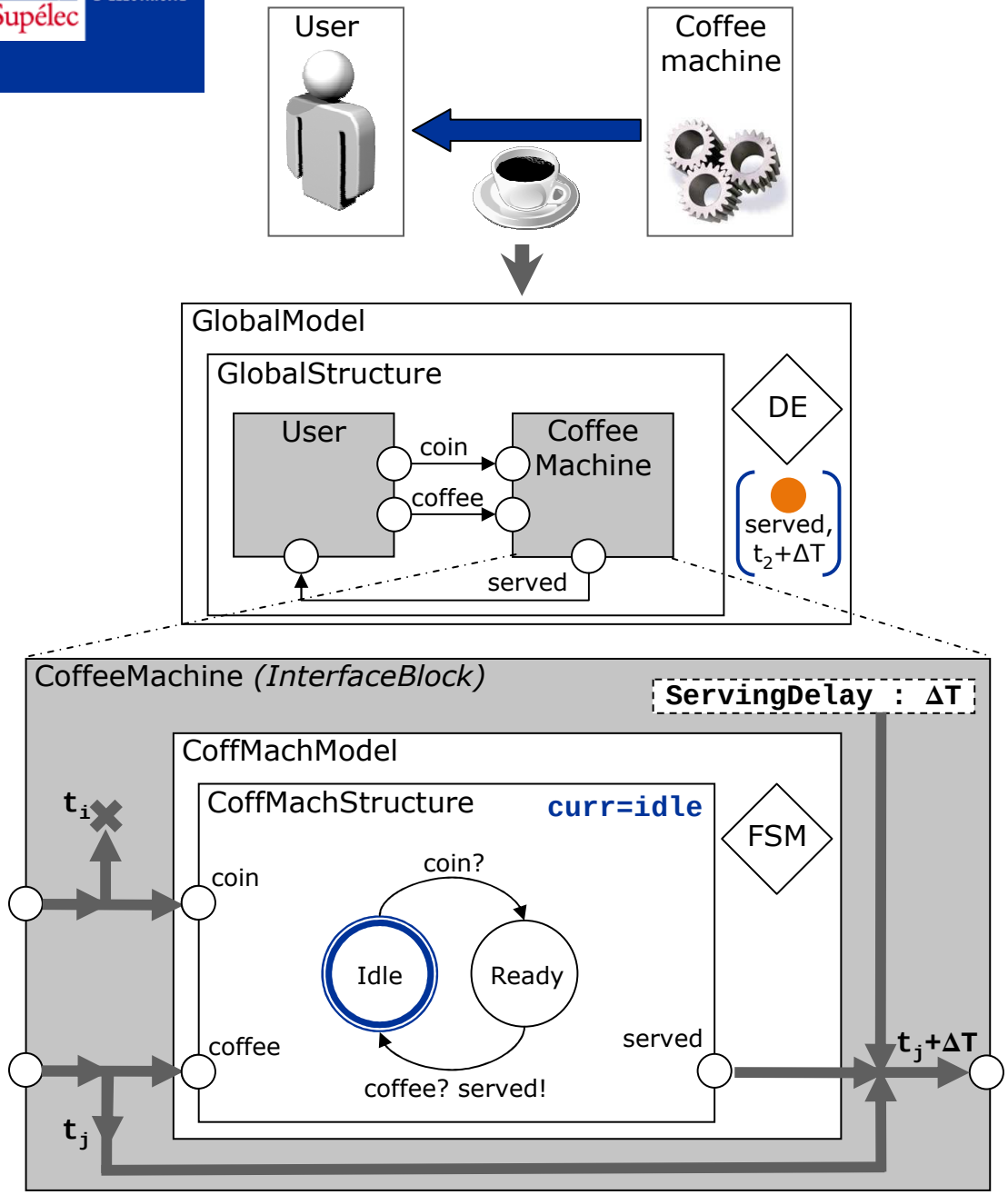
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time $t_3 = t_2 + \Delta T$

Running the model



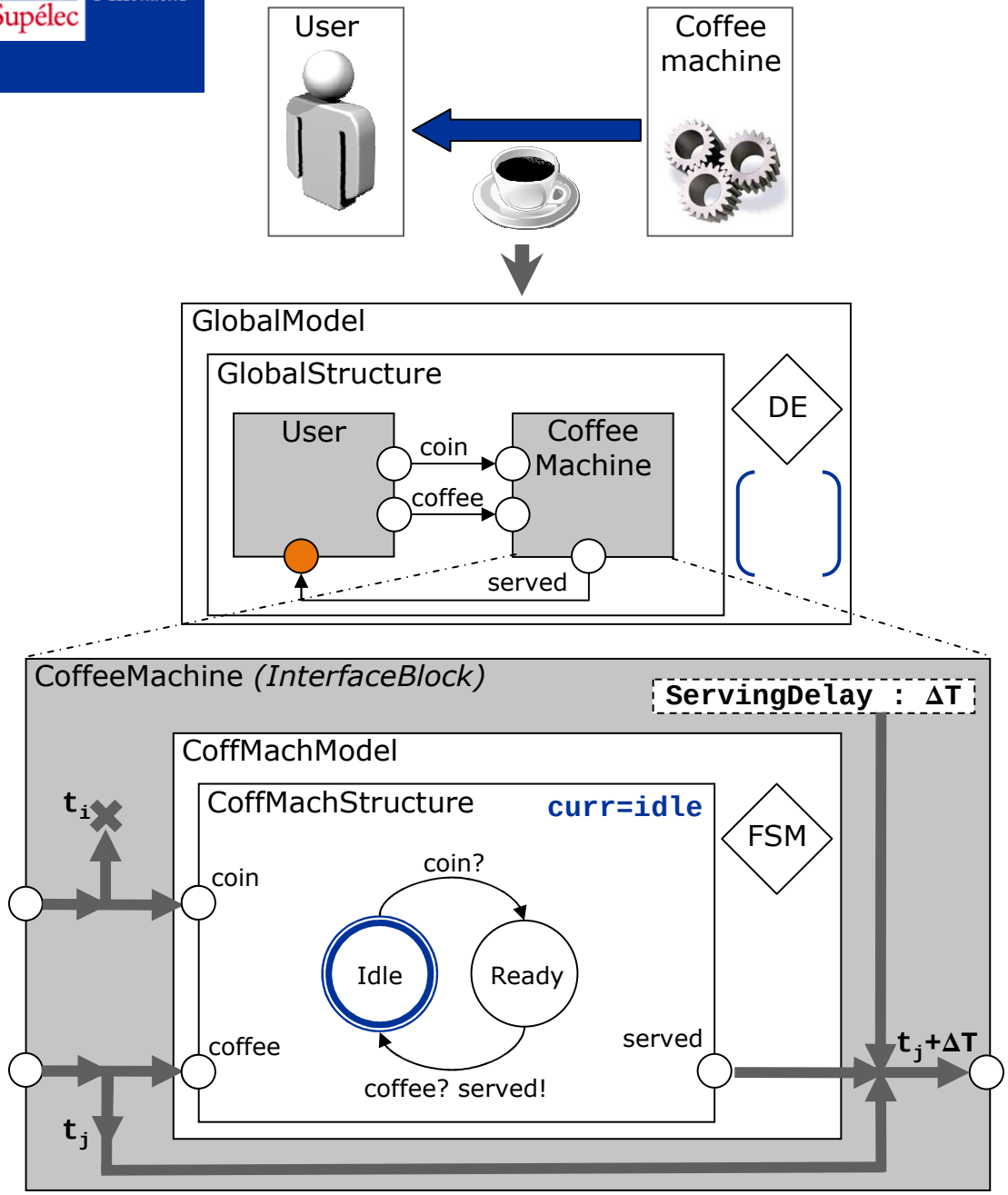
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time $t_3 = t_2 + \Delta T$

Running the model



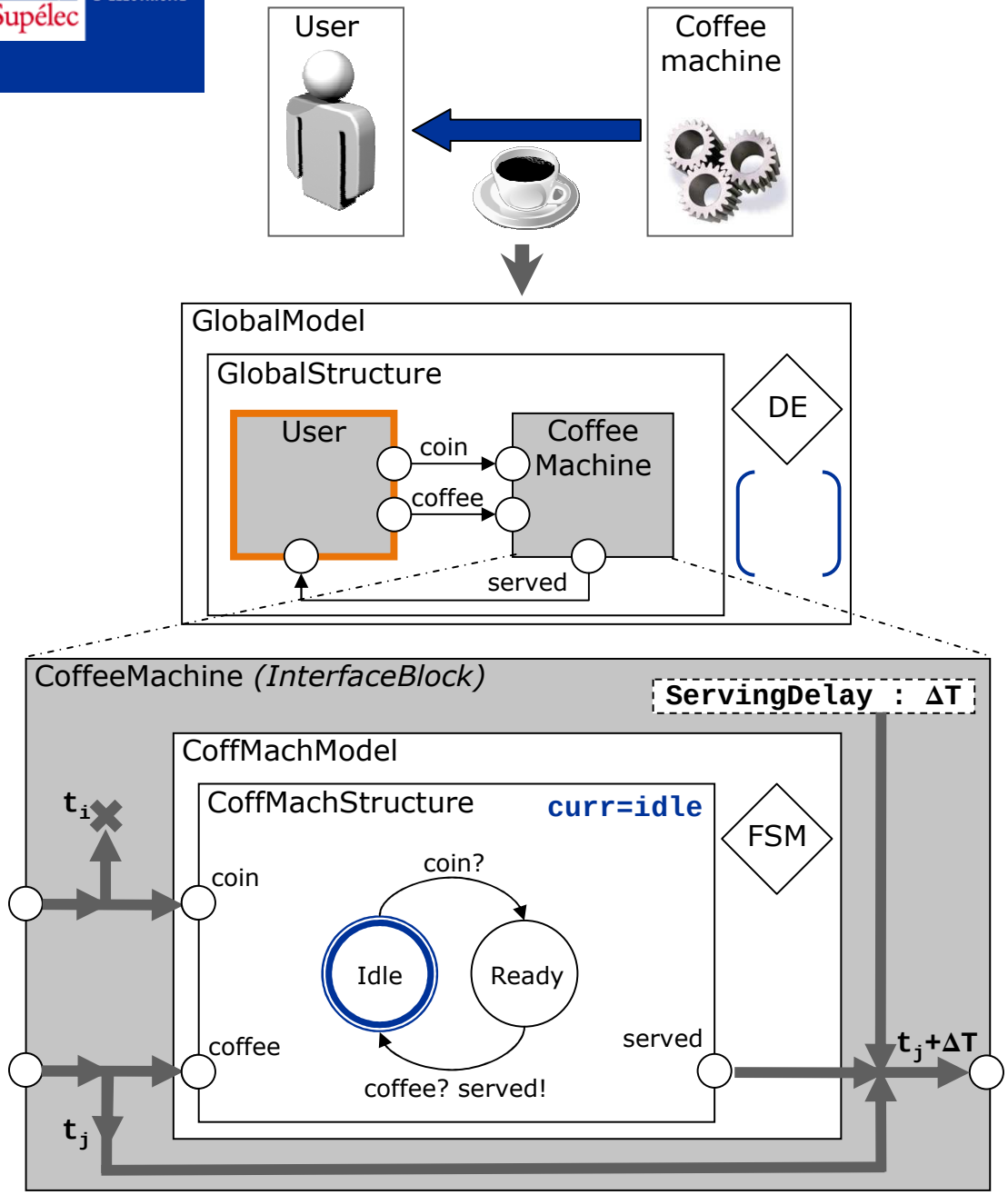
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time $t_3 = t_2 + \Delta T$
3. The machine delivers the coffee
   - 3rd snapshot at t_3

Running the model



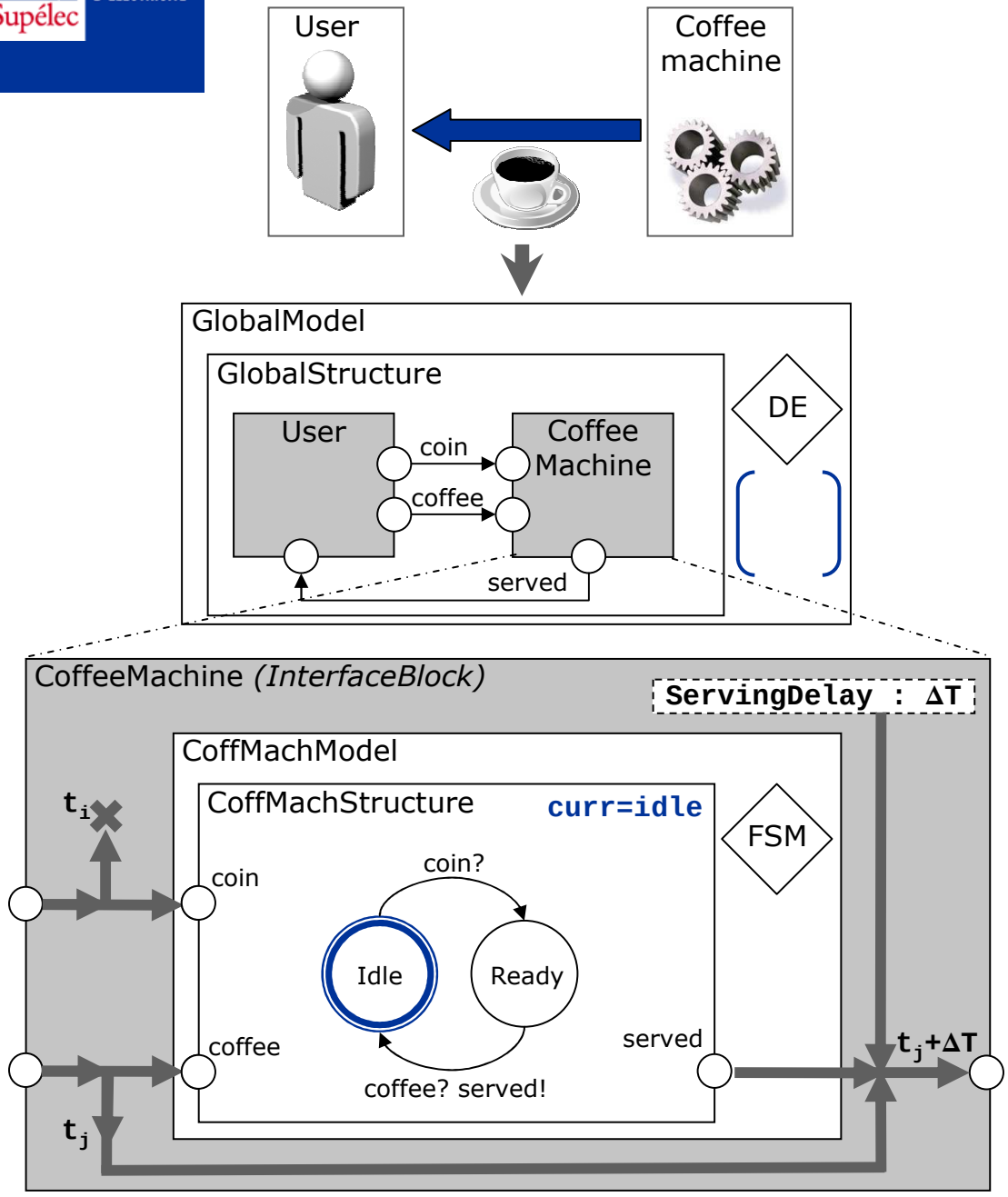
1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time $t_3 = t_2 + \Delta T$
3. The machine delivers the coffee
   - 3rd snapshot at t_3
   - the user receives the served event at time $t_3 = t_2 + \Delta T$

Running the model



1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time t_3 and $t_3 = t_2 + \Delta T$
3. The machine delivers the coffee
   - 3rd snapshot at t_3
   - the user receives the served event at time $t_3 = t_2 + \Delta T$

Running the model



1. The user inserts the coin
   - 1rst snapshot at t_1
   - the user produces the `coin` event with time t_1
2. The user pushes the button
   - 2nd snapshot at t_2
   - the user produces the `coffee` event with time t_2
   - the machine produces the `served` event with time t_3 and $t_3 = t_2 + \Delta T$
3. The machine delivers the coffee
   - 3rd snapshot at t_3
   - the user receives the served event at time $t_3 = t_2 + \Delta T$

Agenda

1. Context, existing approaches & motivations
2. ModHel'X: underlying concepts
3. The coffee machine example
4. Discussion & conclusion

Discussion (1/2)

- Intended workflow and required effort to use ModHel'X

once!

1. An expert of a modeling language describes:
   - The structural and semantic elements of the language
     - *Specialized meta-model*
     - *Imperative semantics of the execution operations*
   - Transformations from the original meta-model of the language to the ModHel'X meta-model
2. Experts define interaction patterns (= "classical glues") for each pair of model of computation that may interact
3. Designers use ModHel'X

- Supported MoCs
  - Continuous behaviors: numerical solving (approximation)
  - Cyclic dependencies: fixed point semantics (monotonicity…)
  - Non-determinism: "controlled" non-determinism (pseudo-random functions allowed)

Discussion (2/2)

- ModHel'X & PtolemyII
  - PtolemyII is our main source of inspiration
  - Contributions:

	ModHel'X	**PtolemyII**
Abstract syntax	UML meta-model	Proprietary abstract syntax
Specification of the semantics of MoCs	Generic execution algorithm + imperative semantics	Structured Java code
Specification of the interactions between MoCs	InterfaceBlock operations + imperative semantics	Predefined combinations
Execution paradigm	Observation of blocks	Firing of actors

Discussion (2/2)

- ModHel'X & PtolemyII
  - PtolemyII is our main source of inspiration
  - Contributions:

	ModHel'X	PtolemyII
Abstract syntax	UML meta-model	Proprietary abstract syntax
Specification of the semantics of MoCs	Generic execution algorithm + imperative semantics	Structured Java code
Specification of the interactions between MoCs	InterfaceBlock operations + imperative semantics	Predefined combinations
Execution paradigm	Observation of blocks	Firing of actors

Conclusion

- ModHel'X = an approach to multi-formalism modeling with
  - A generic meta-model for representing heterogeneous models
    - A specific structure for the explicit and flexible specification of the interactions between MoCs
  - A generic algorithm for executing heterogeneous models
    - A fixed frame for expressing MoCs
- Work in progress
  - Prototype based on the Eclipse Modeling Framework (EMF)
    - Several implemented MoCs
    - Working on the Synchronous DataFlow and UML StateCharts MoCs
  - Concrete syntax of our language (OMG ImperativeOCL – QVT)
    - Verbosity
    - Formal semantics
- Perspectives
  - Facets (non-functional properties for embedded systems)

References

[Kermeta] Muller, P.-A., F. Fleurey and J.-M. J´ez´equel, Weaving executability into object-oriented meta-languages, in: Proceedings of the 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS/UML 2005), 2005, pp. 264–278.

[ATOM^3] de Lara, J. and H. Vangheluwe, ATOM3: A tool for multi-formalism modelling and meta-modelling, in: 5th Fundamental Approaches to Software Engineering International Conference (FASE 2002), 2002, pp. 595–603.

[PtolemyII] Eker, J., J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs and Y. Xiong, Taming heterogeneity – the Ptolemy approach, Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software 91 (2003), pp. 127–144.

ÉCOLE SUPÉRIEURE D'ÉLECTRICITÉ

Supélec

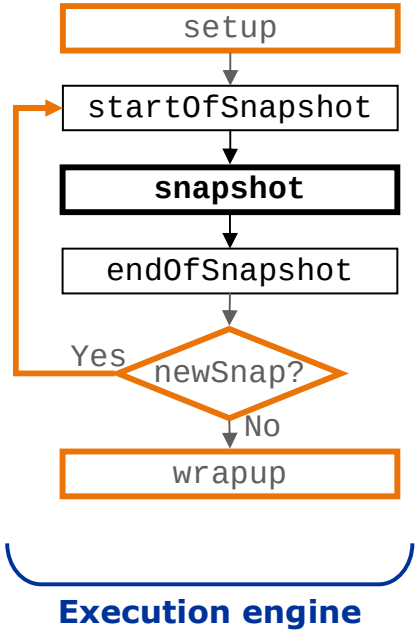
Appendix

Time

- Only notion of Time = succession of snapshots
  - Each MoC can define its own notion of time upon it
  - Each MoC can express constraints on its "date" at the next snapshot

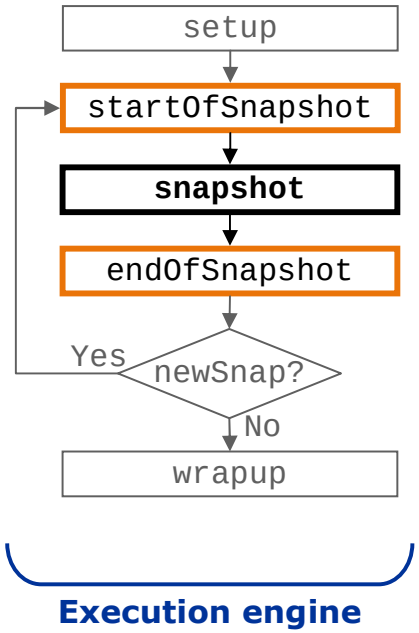
  - Time can be synchronized at the boundary between MoCs
  - Constraints are propagated through the hierarchy to the top level

Structure of the algorithm



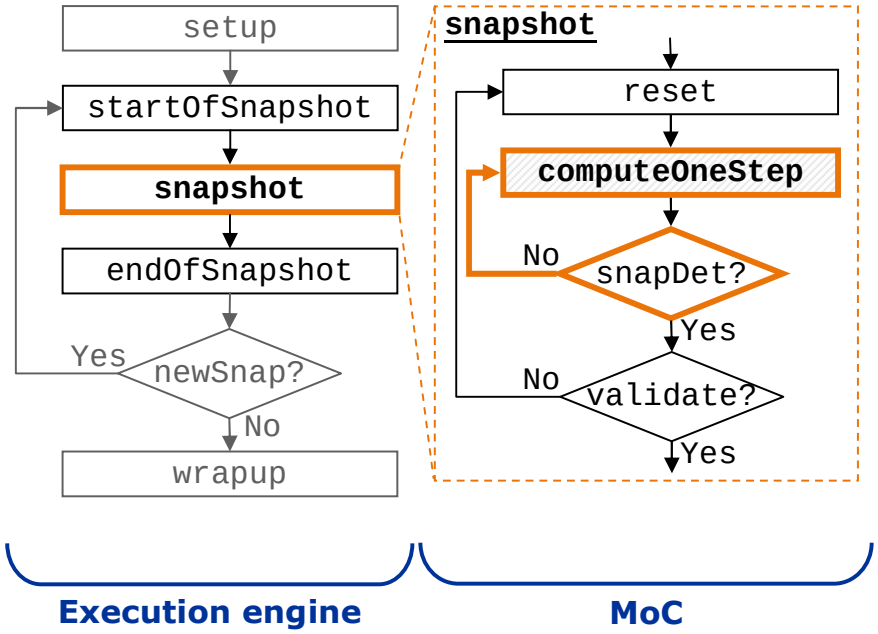
- Loop for triggering the snapshots
  - Handle environment changes and emitted constraints

Structure of the algorithm



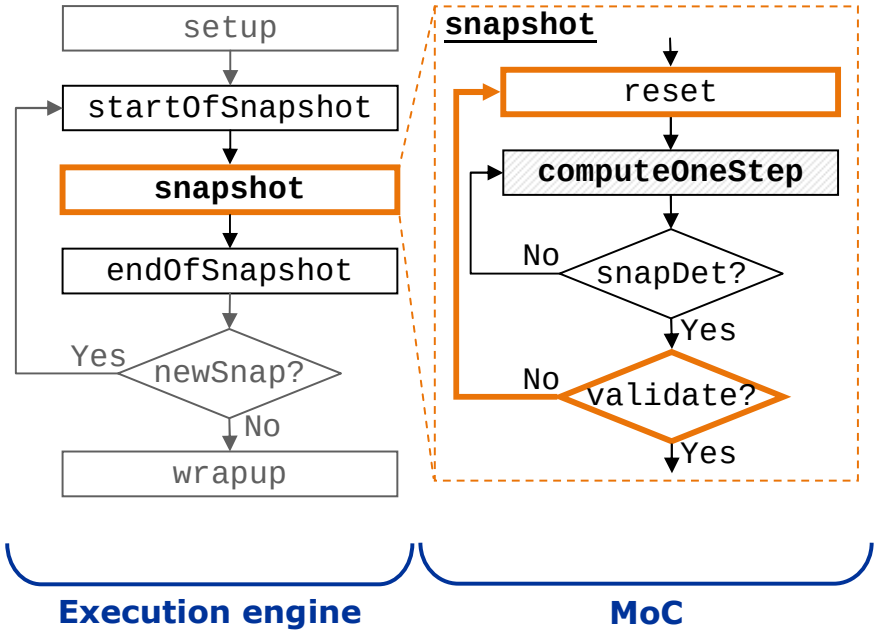
- Loop for triggering the snapshots
  - Handle environment changes and emitted constraints
- Preparation/end of a snapshot
  - Provide inputs/Propagate outputs and model state changes

Structure of the algorithm



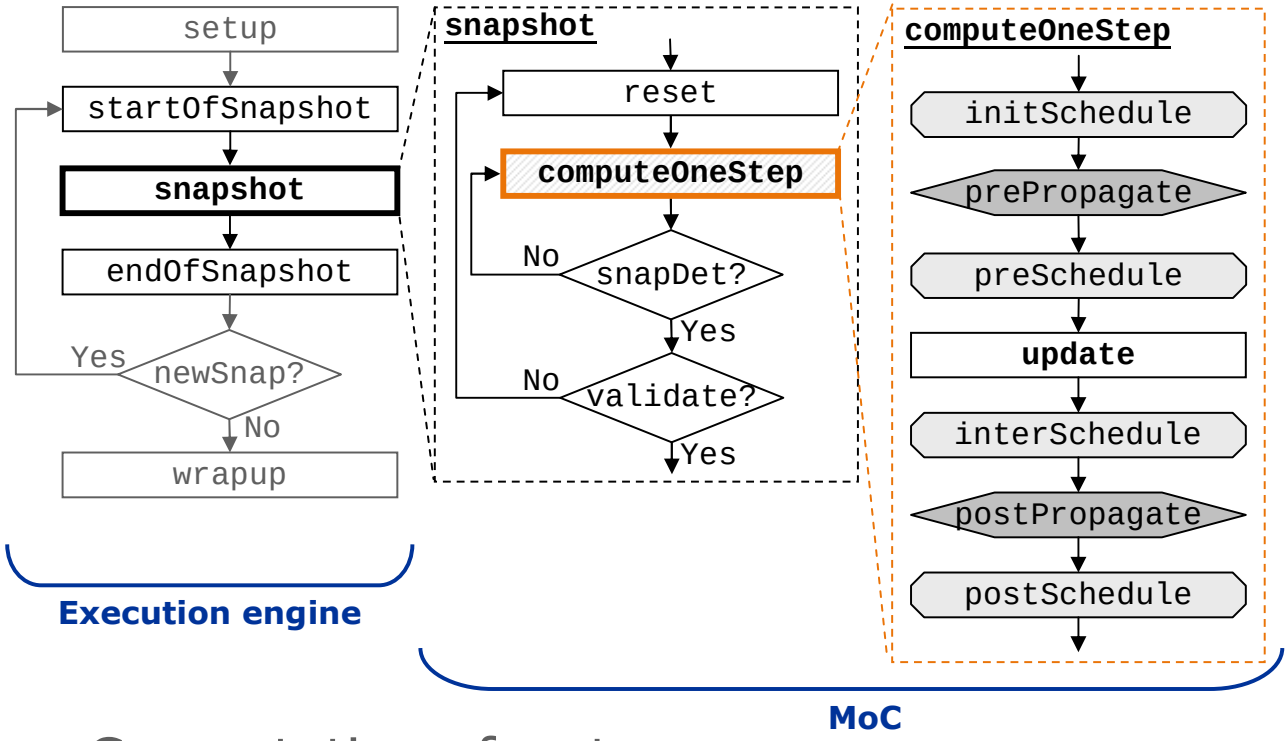
- Successive steps for computing
  - The current outputs
  - The new state of the model

Structure of the algorithm



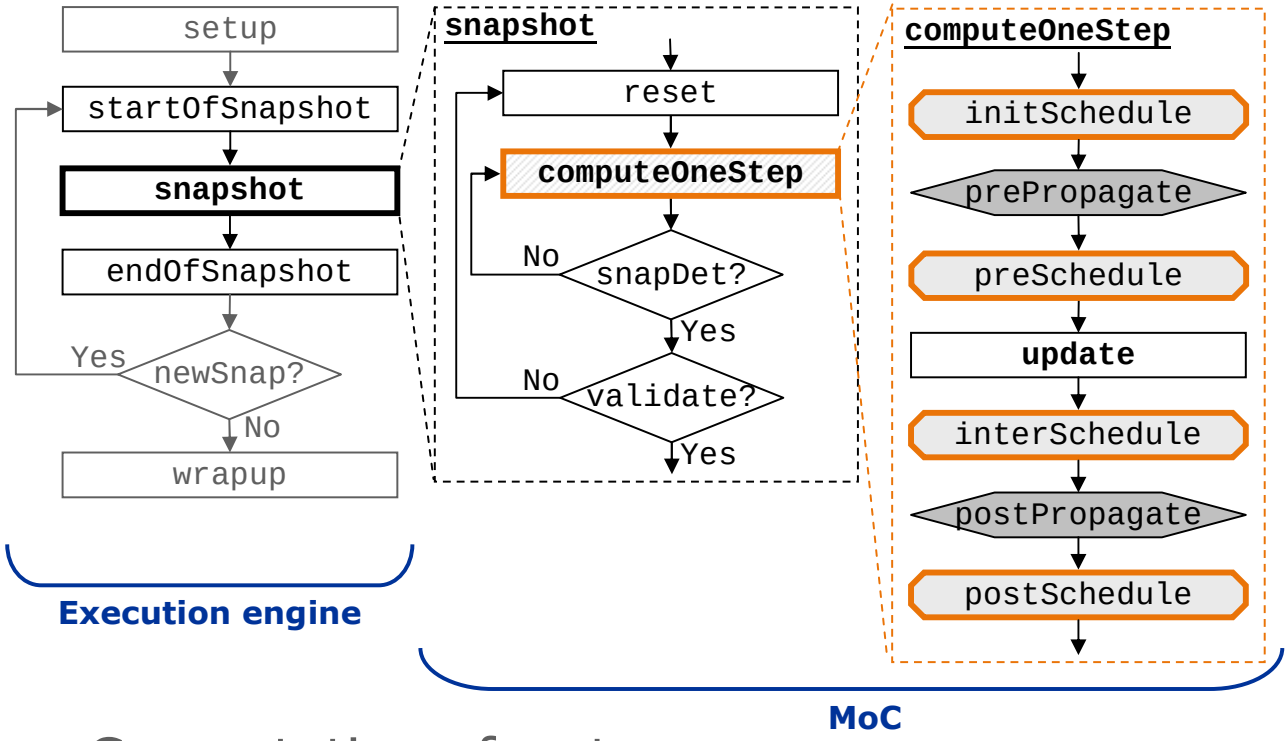
- Successive steps for computing
  - The current outputs
  - The new state of the model
- Validation of the computed snapshot

Structure of the algorithm



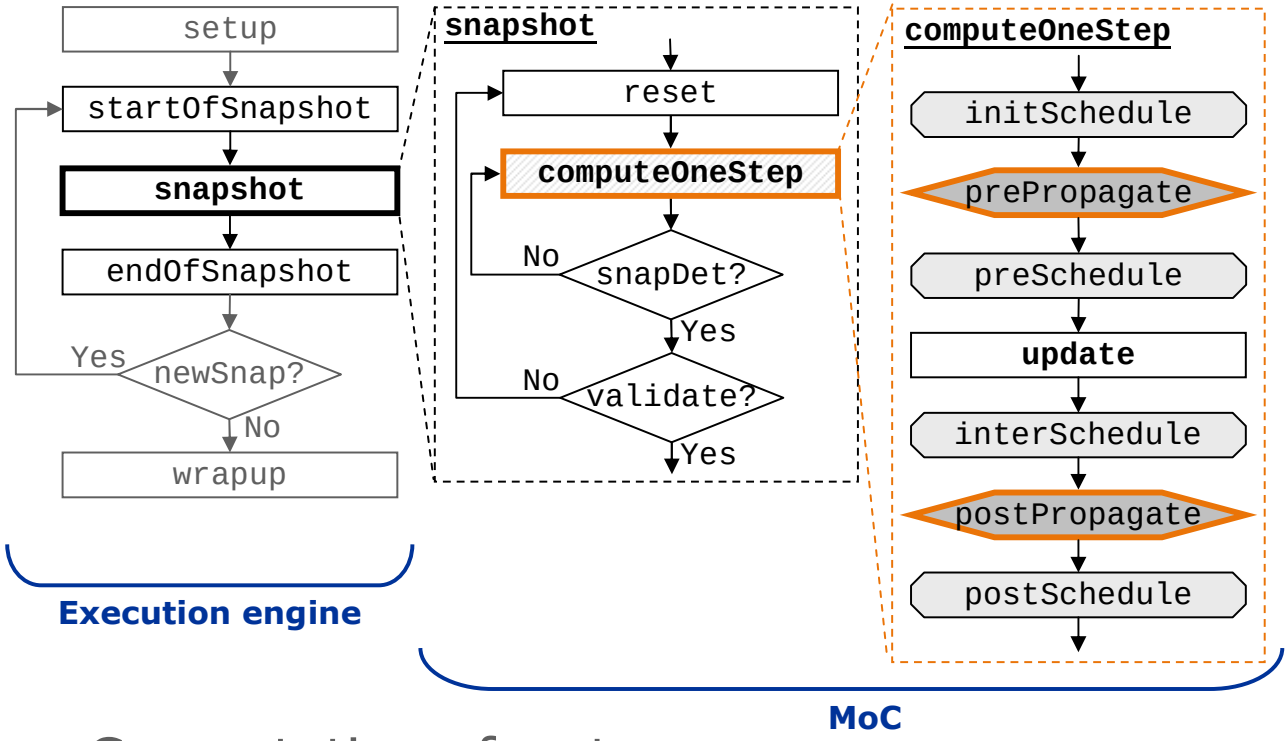
- Computation of a step

Structure of the algorithm



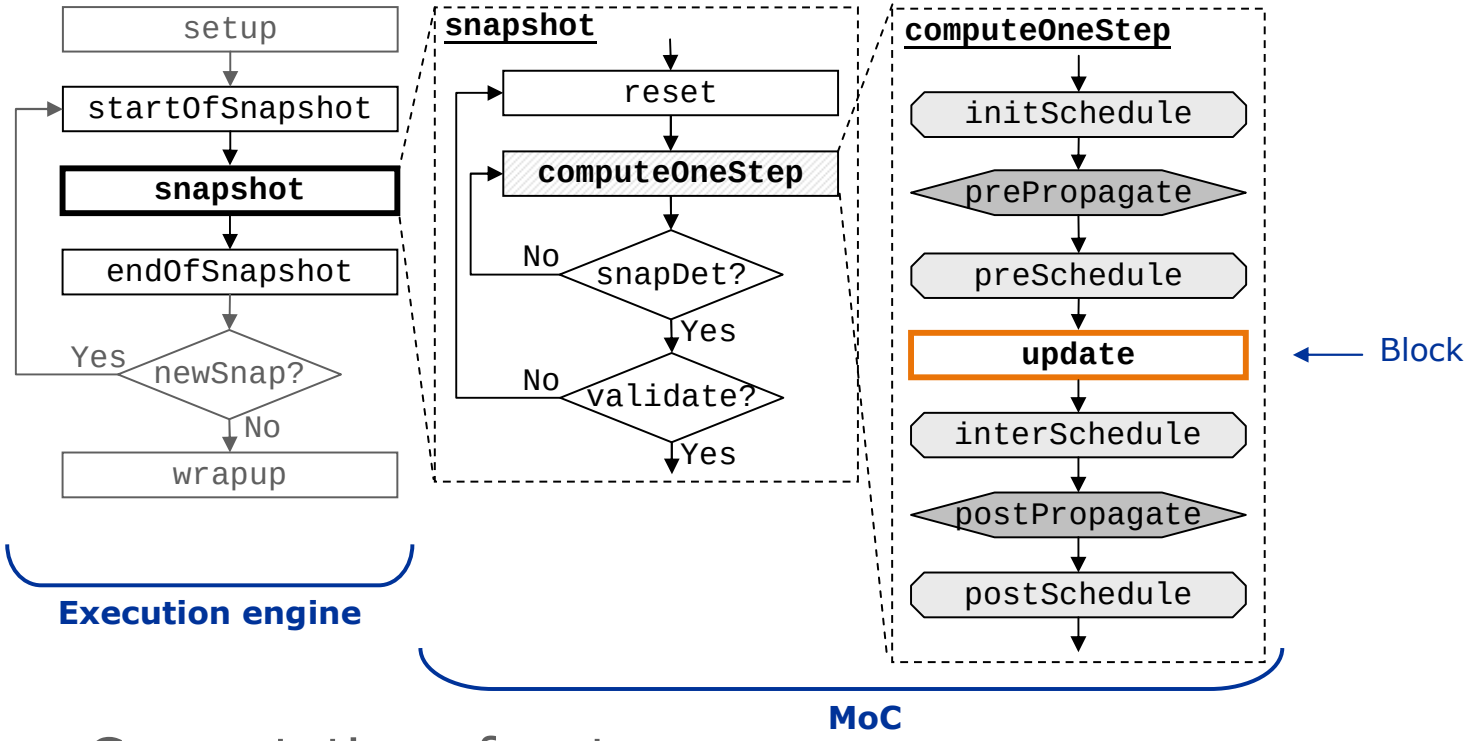
- Computation of a step
  - Scheduling "policies"

Structure of the algorithm



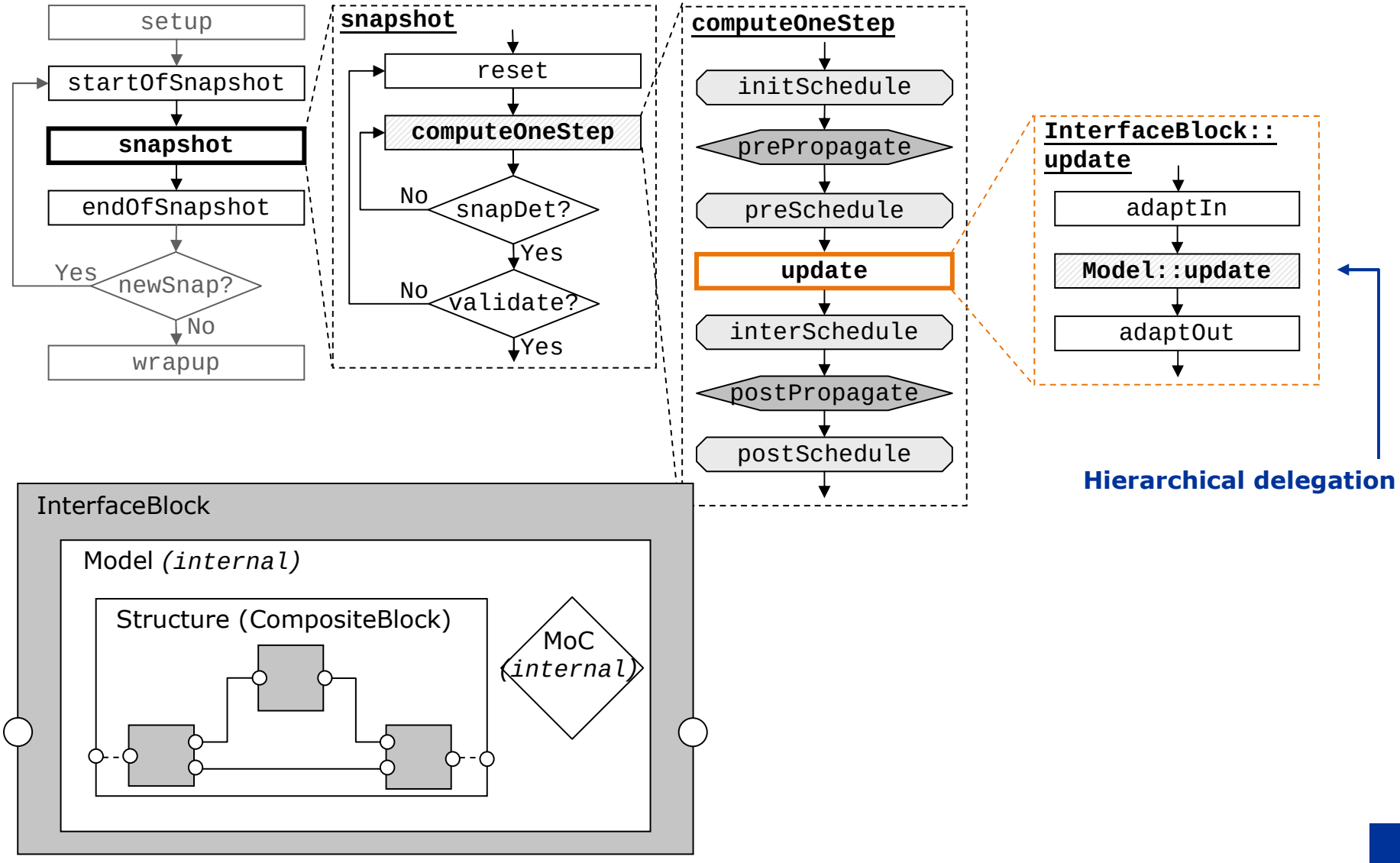
- Computation of a step
  - Scheduling “policies”
  - Propagation “policies”

Structure of the algorithm

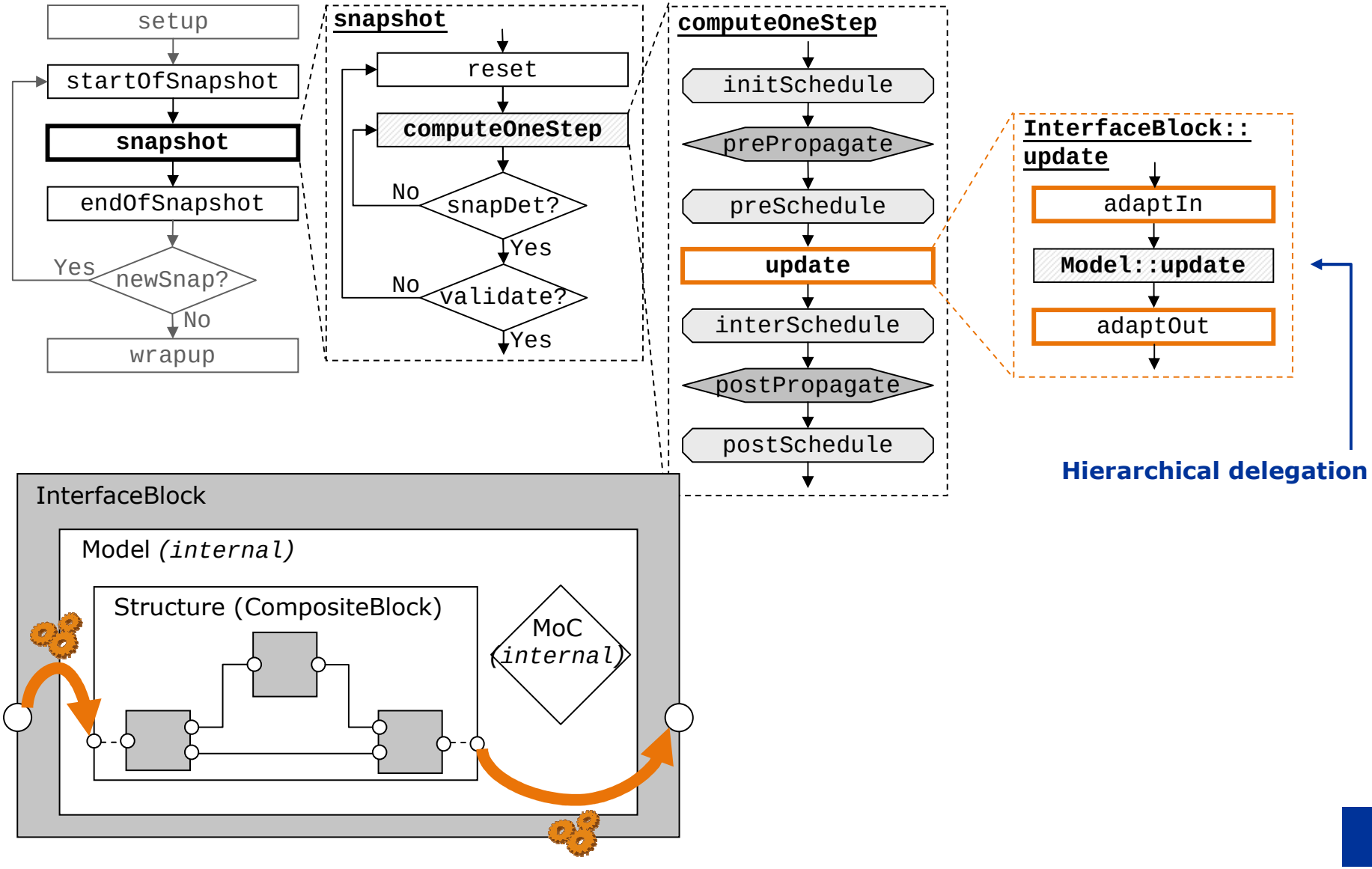


- Computation of a step
  - Scheduling "policies"
  - Propagation "policies"
  - Update

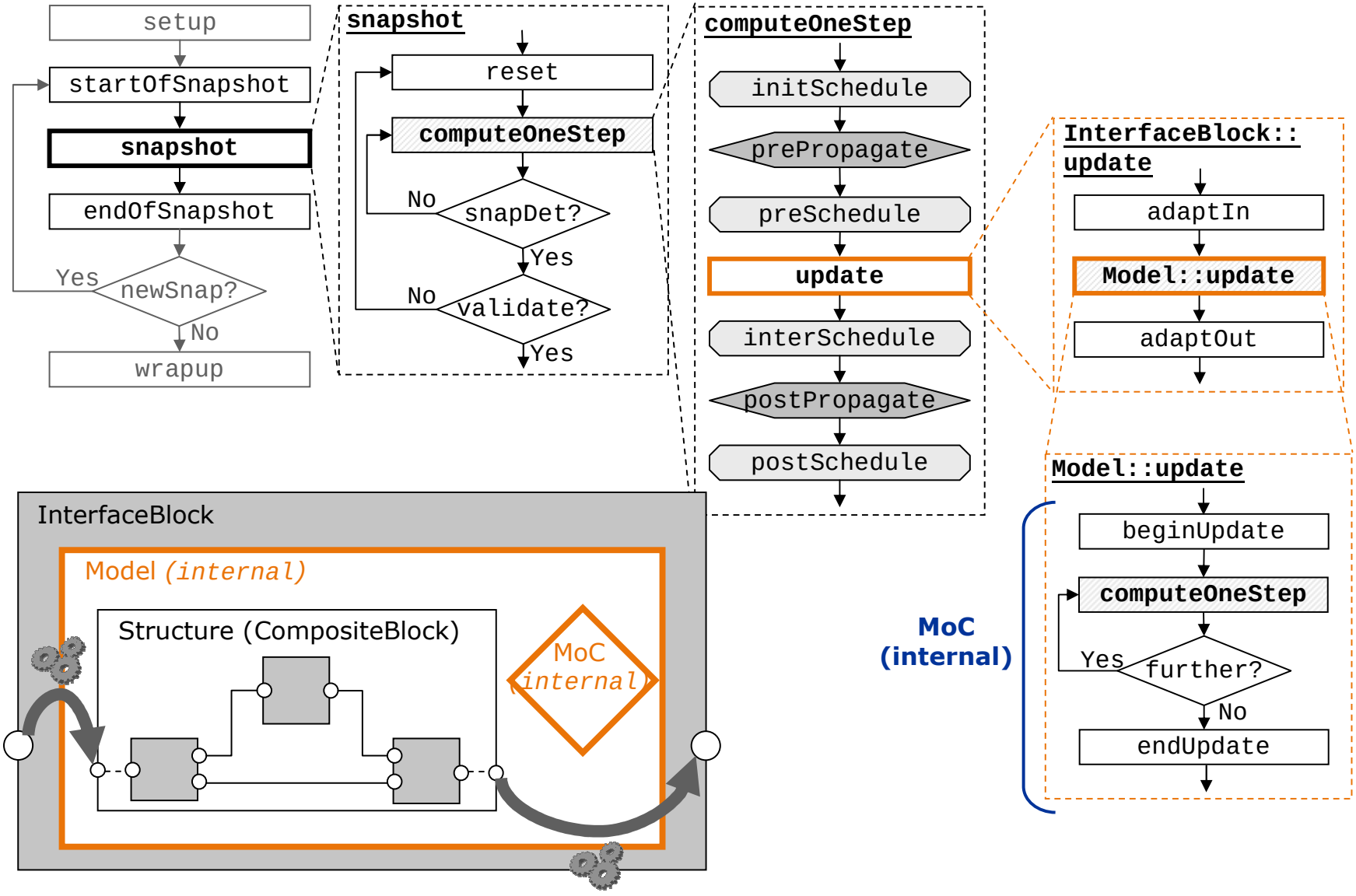
Structure of the algorithm



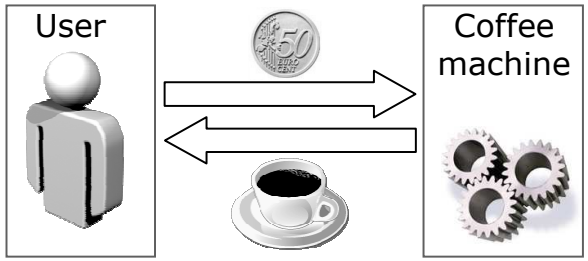
Structure of the algorithm



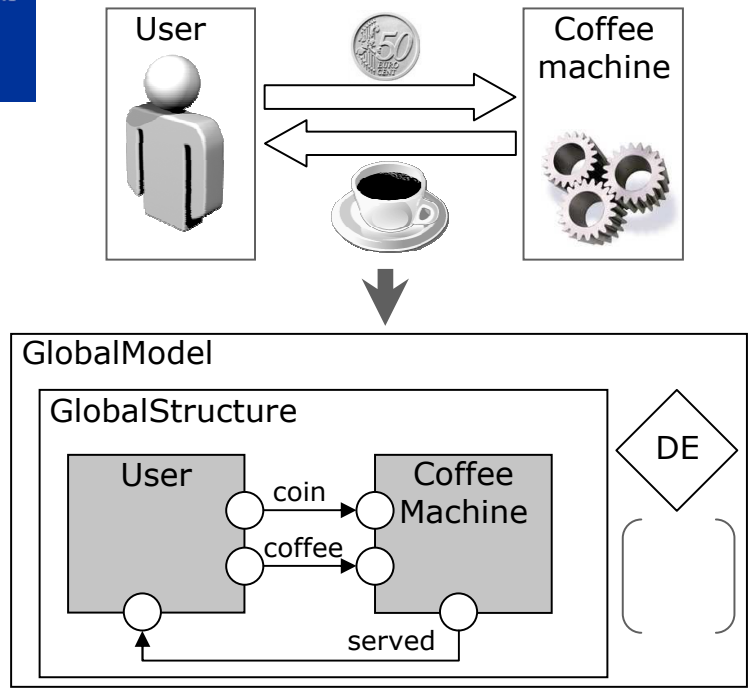
Structure of the algorithm



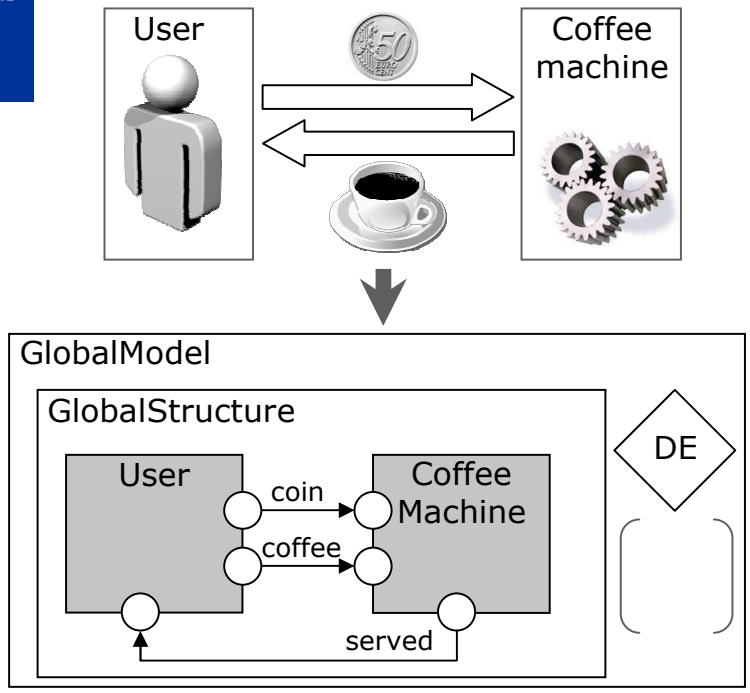
The coffee machine example



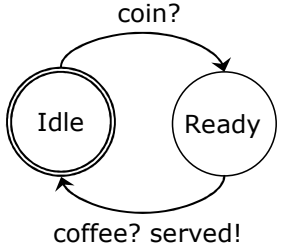
The coffee machine example



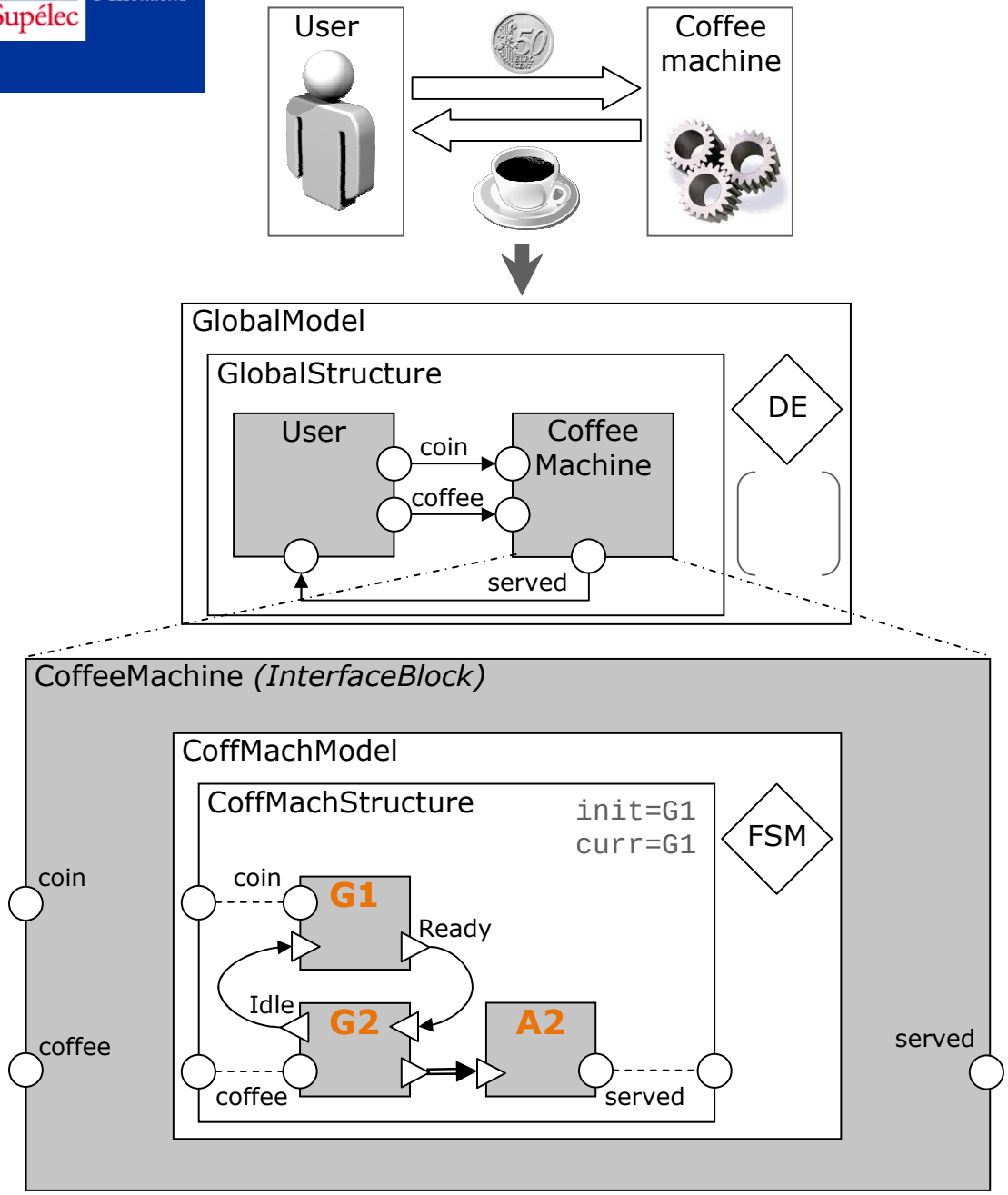
The coffee machine example



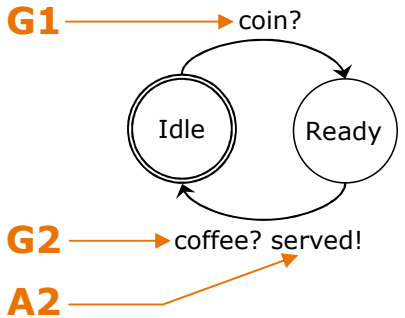
- Coffee machine automaton



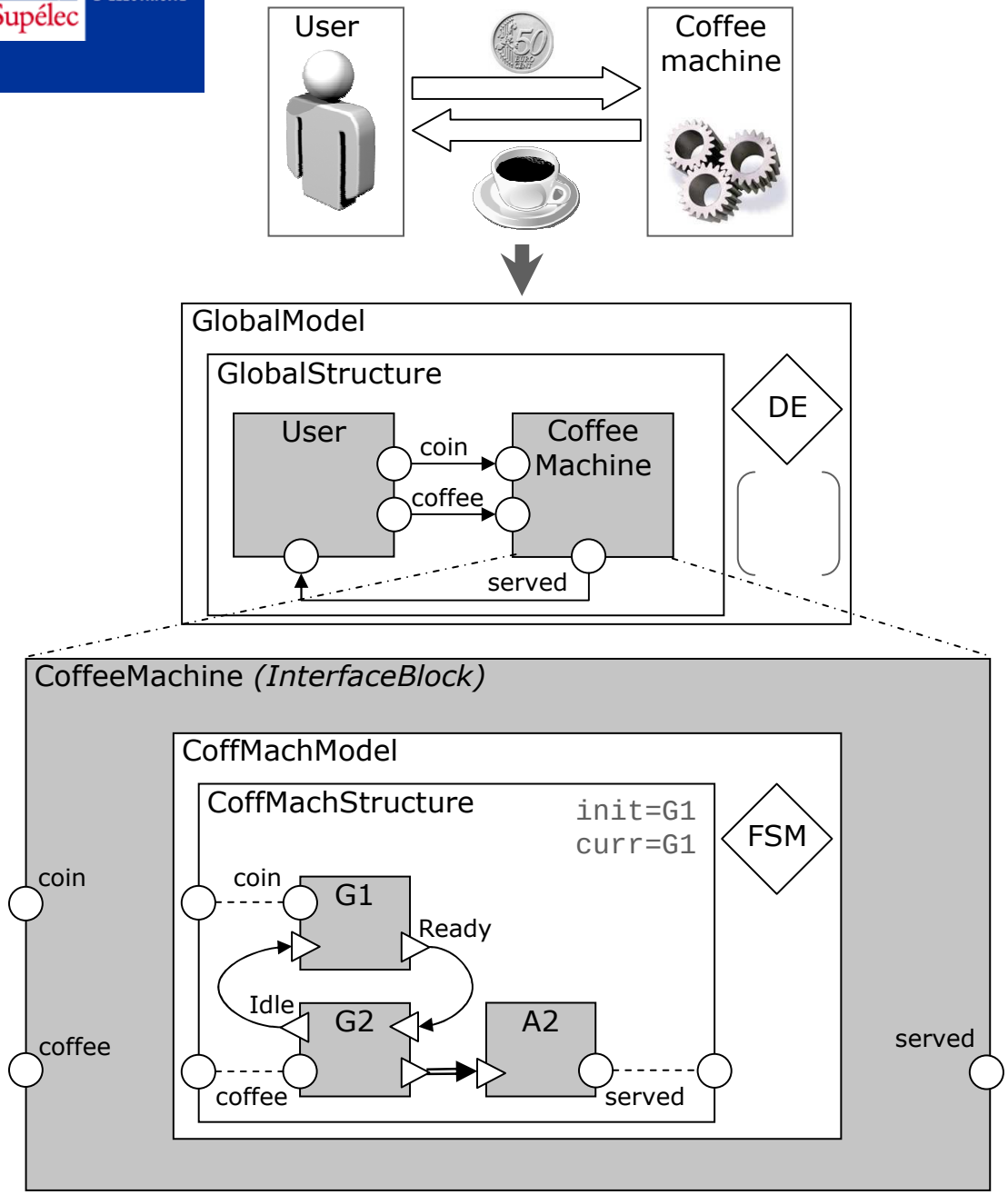
The coffee machine example



- Coffee machine automaton

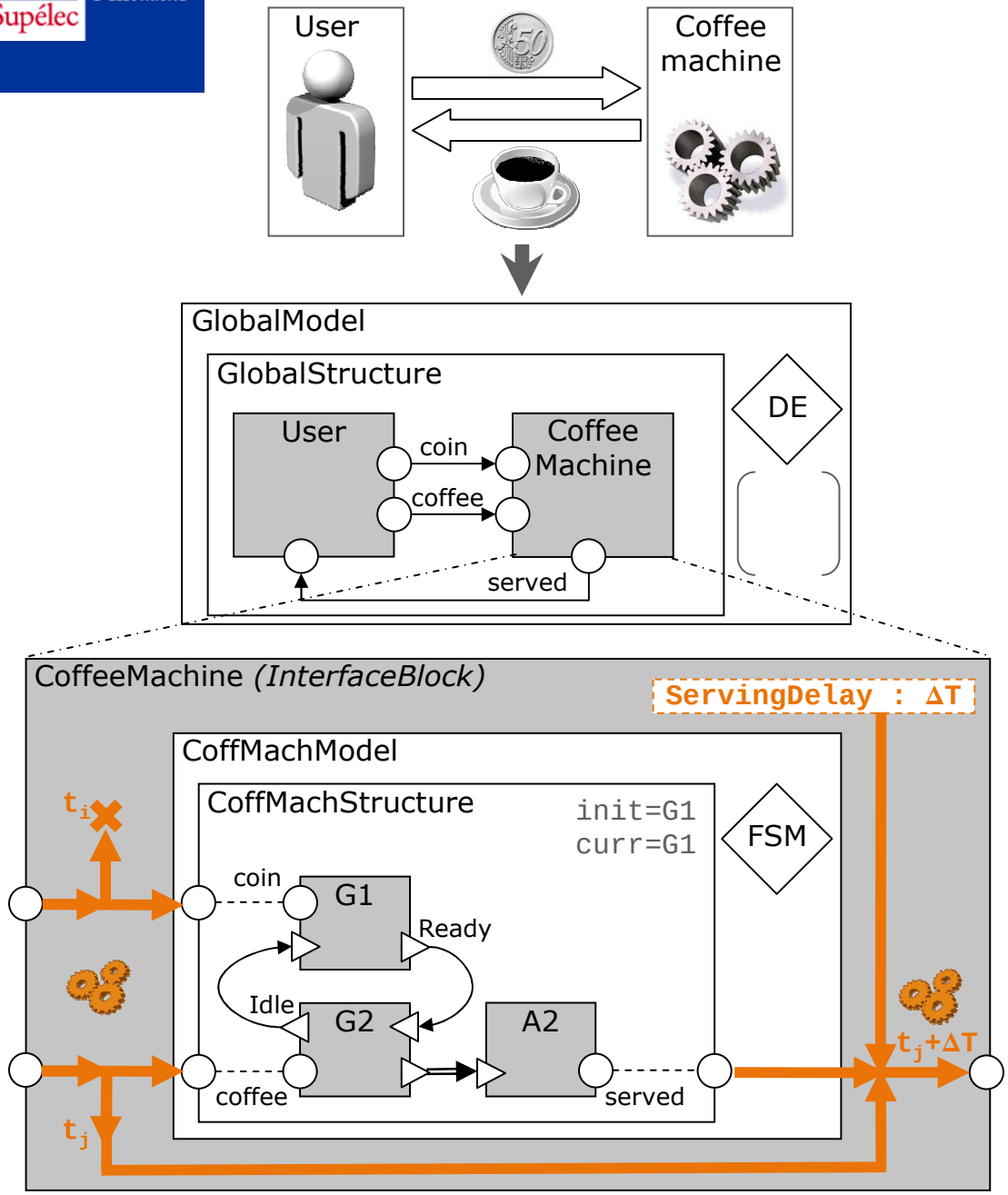


The coffee machine example



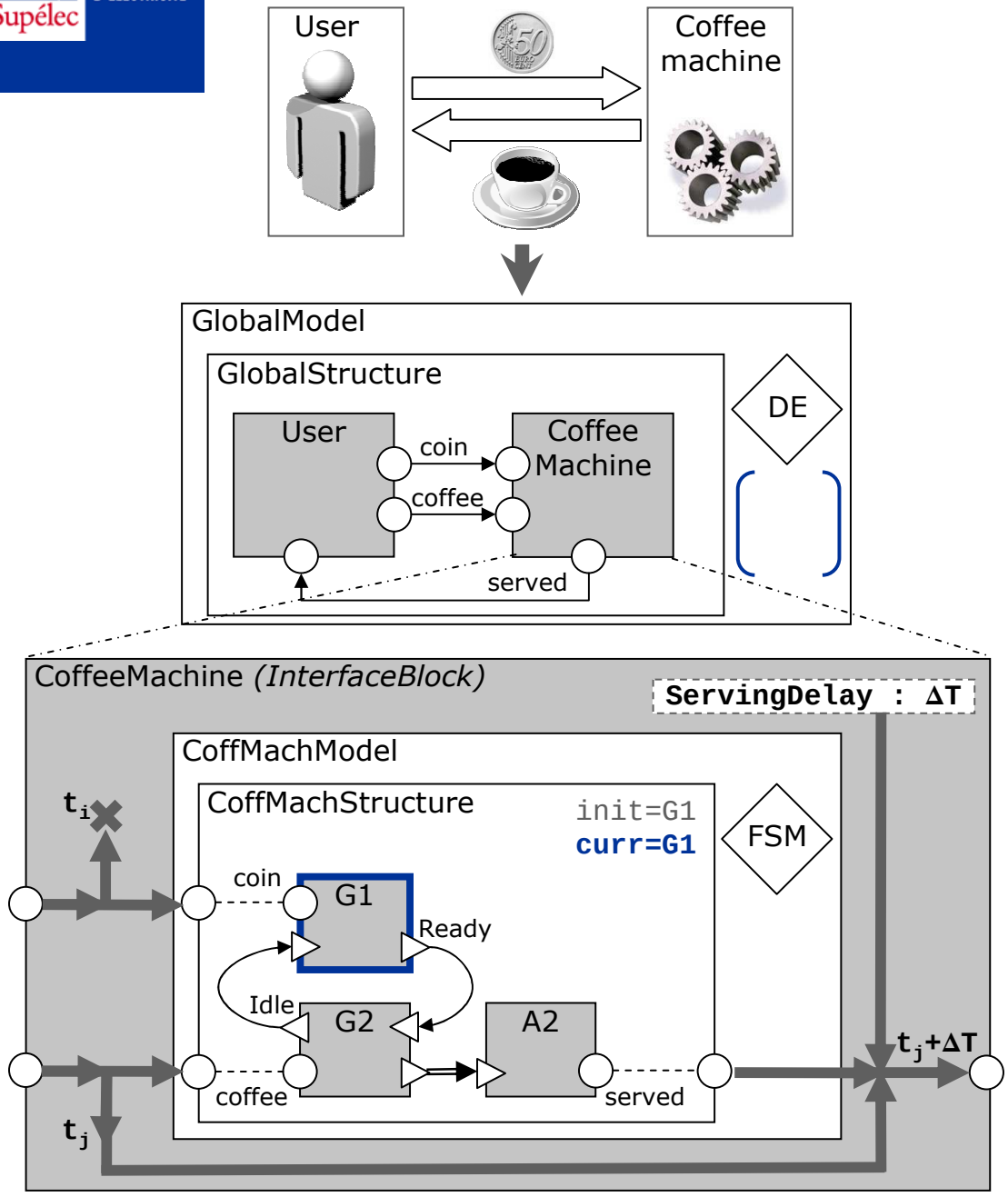
- Time "gluing" ?
  Serving delay ?

The coffee machine example

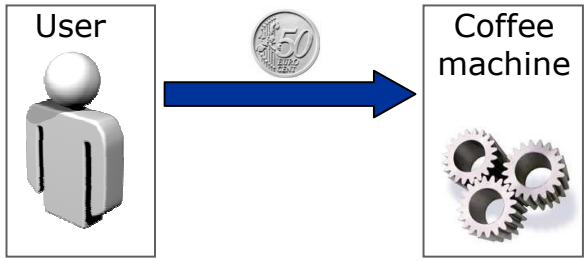


- Time "gluing" ? Serving delay ?
  - adaptIn
  - adaptOut

Running the model

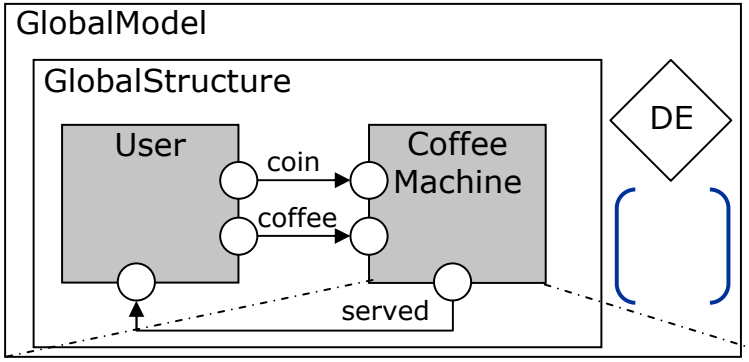


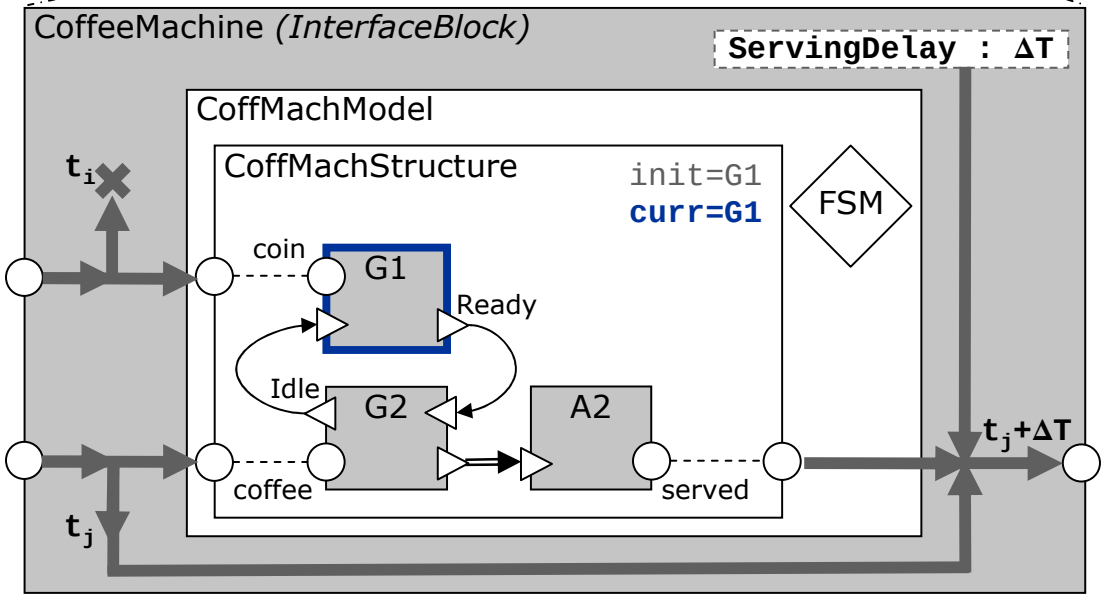
Running the model



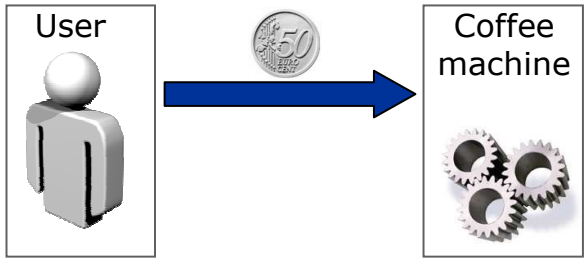
- Constraint on the user

constraints=
[user,0]



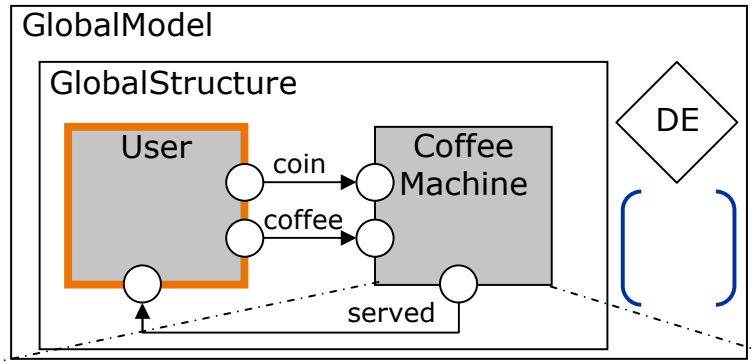


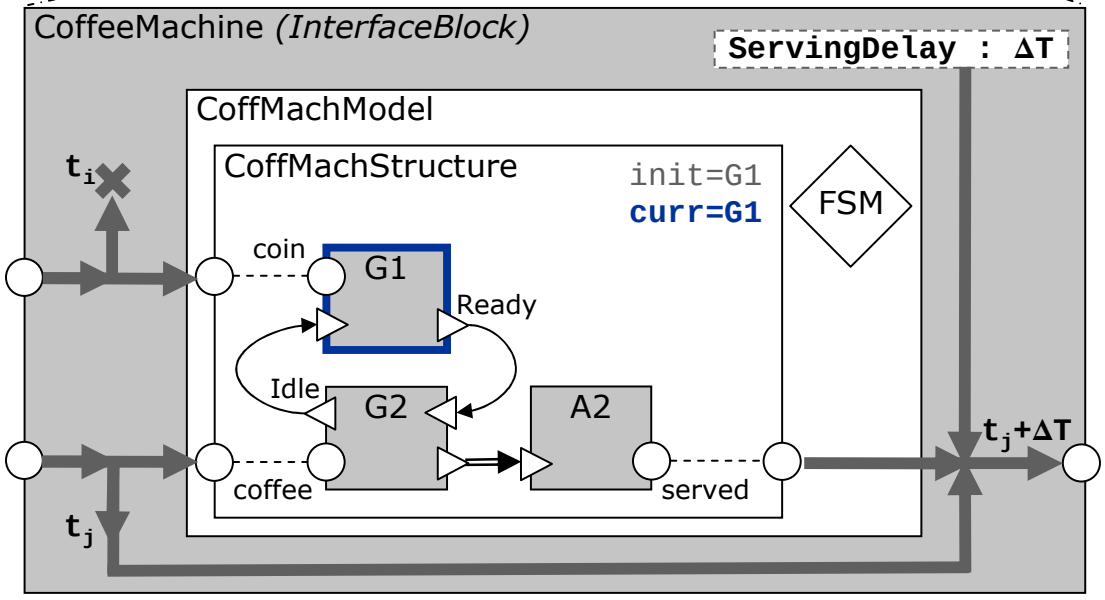
Running the model



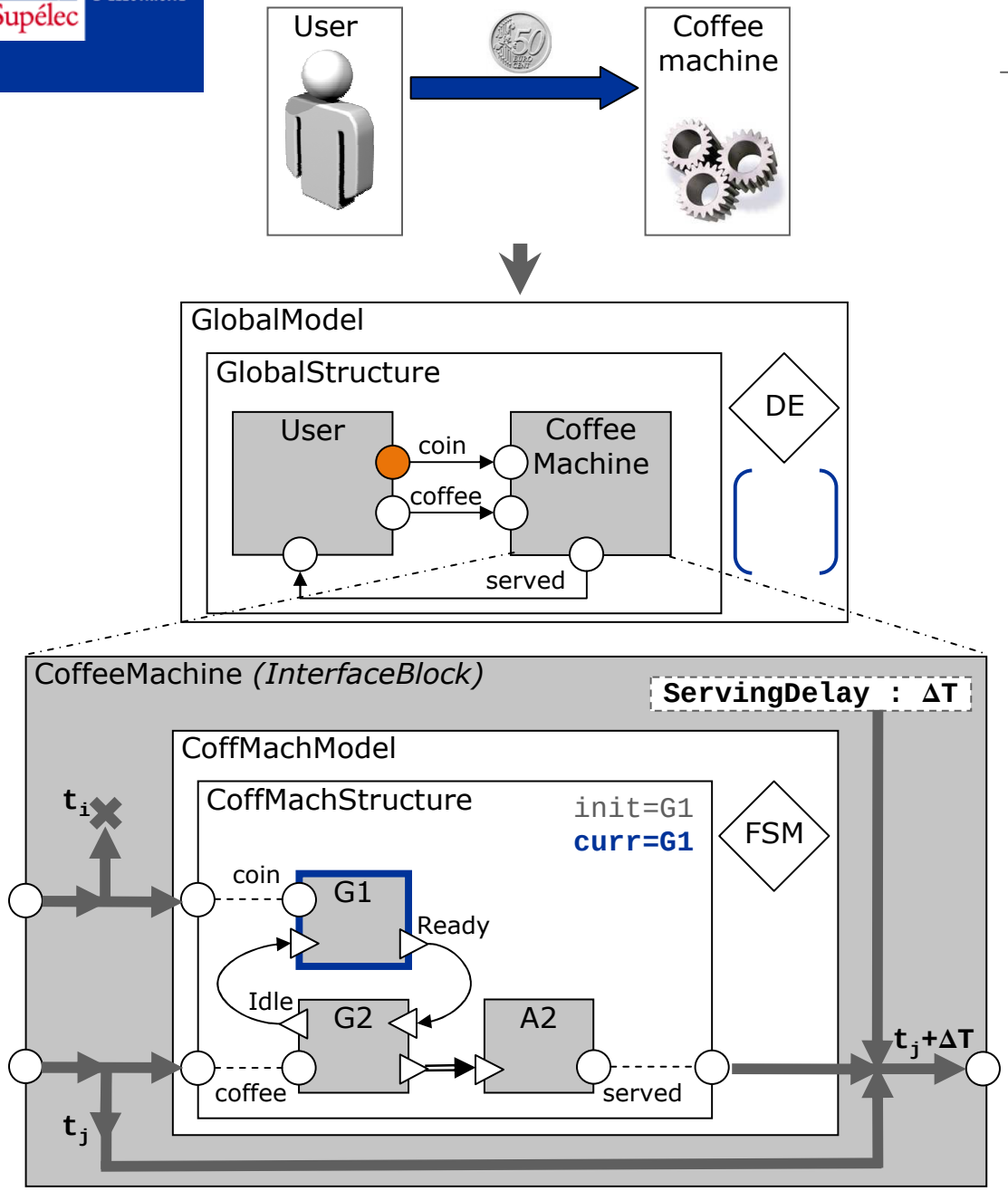
- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

constraints= [user,0]



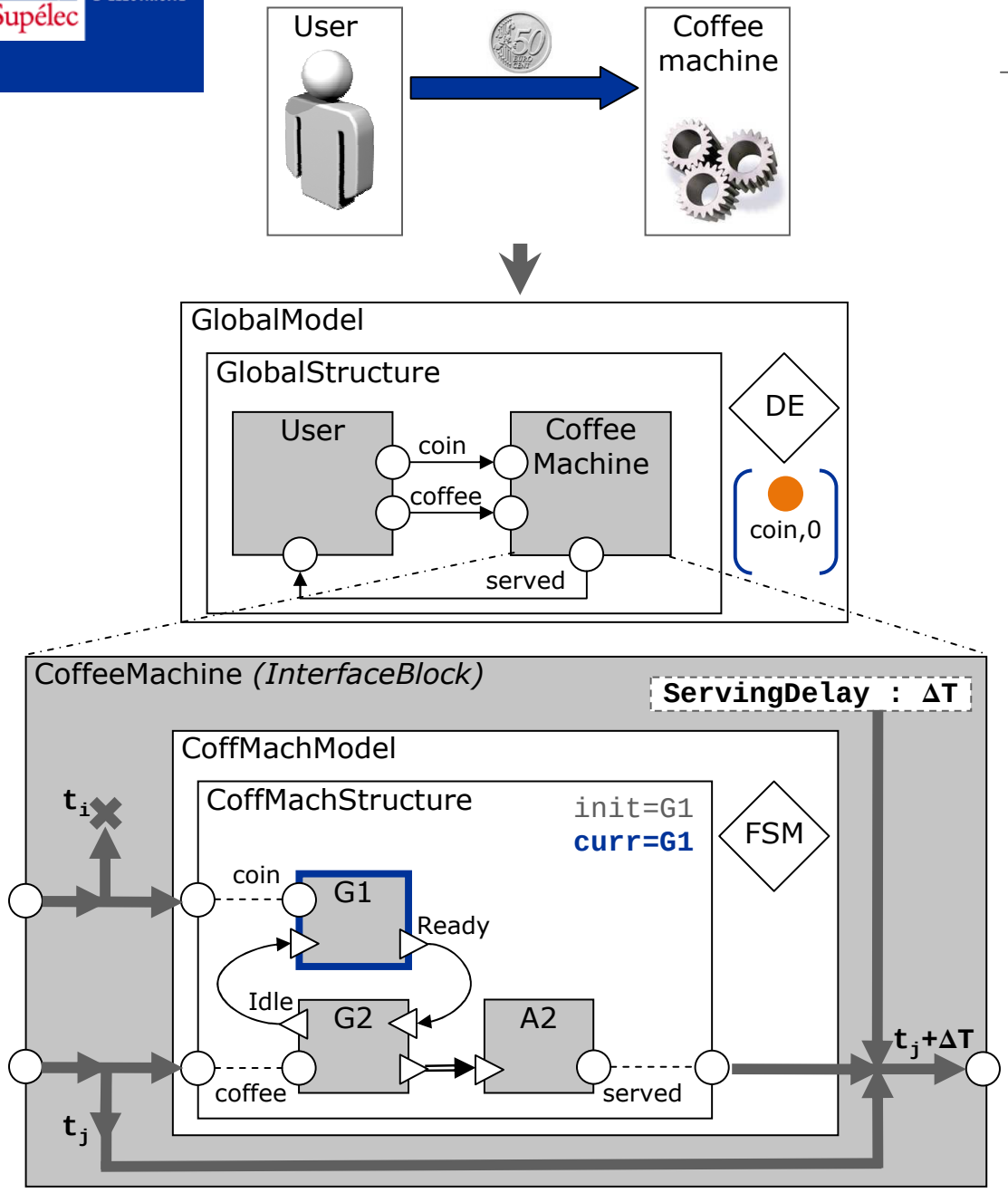


Running the model



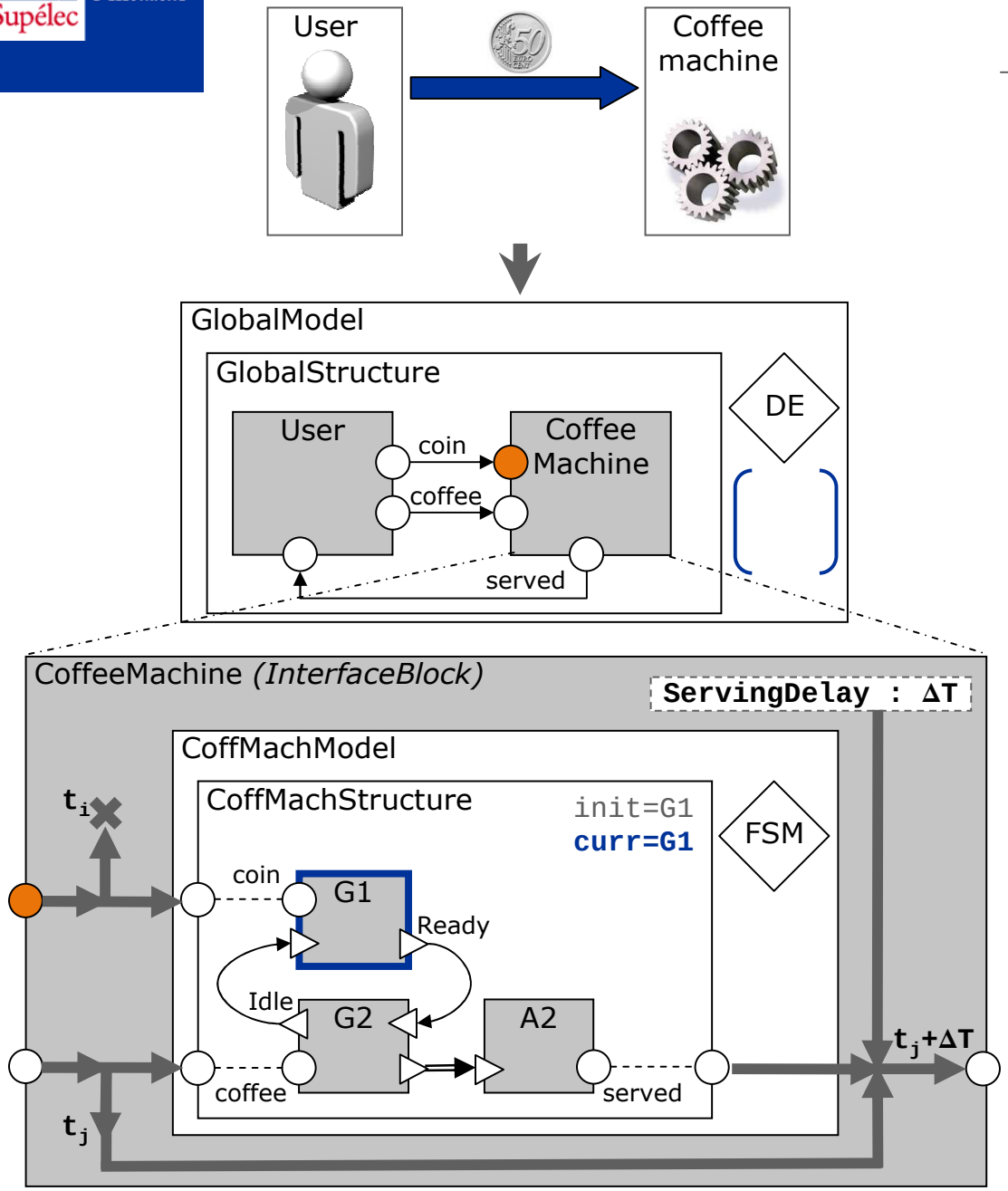
- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

Running the model

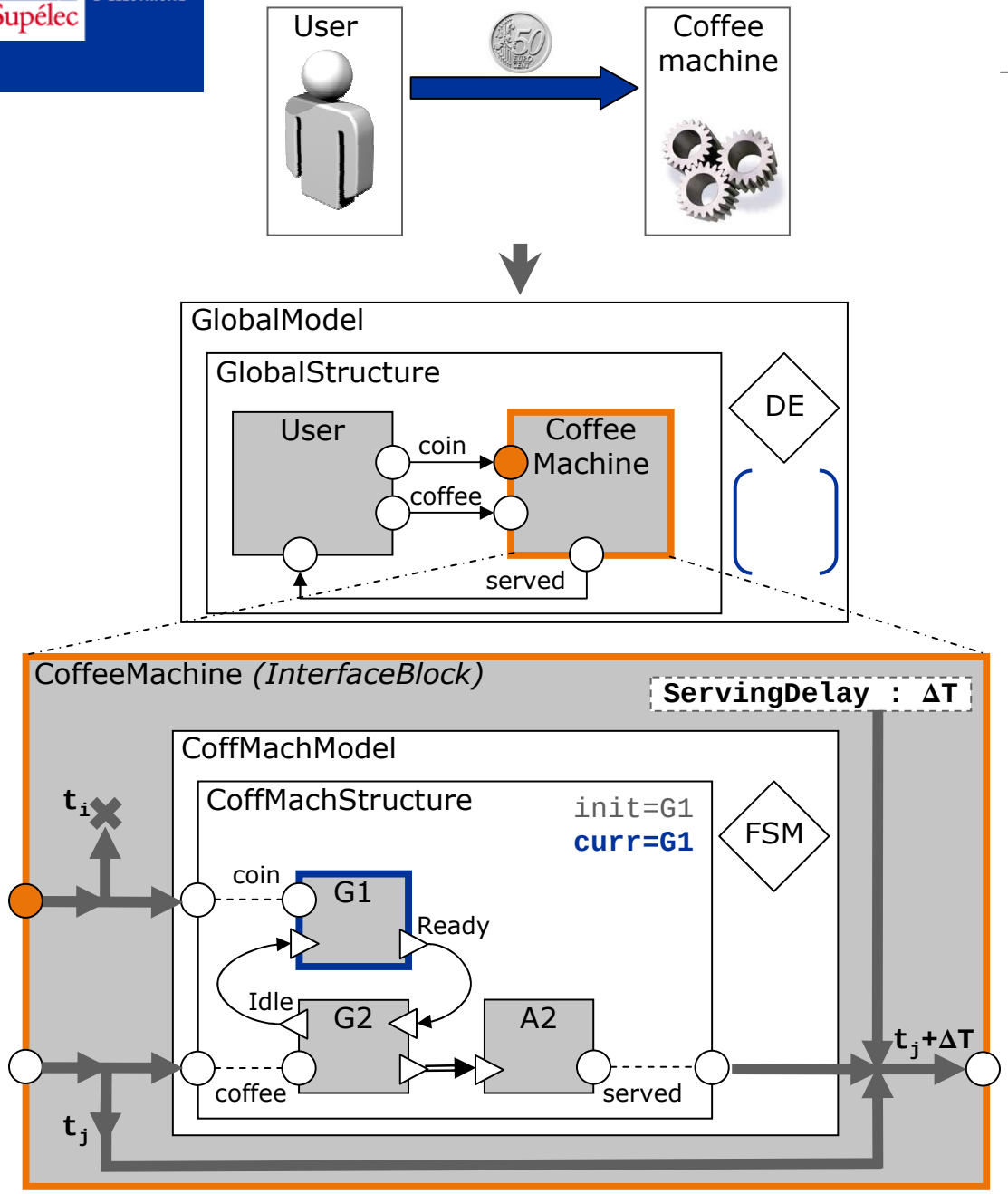


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

Running the model

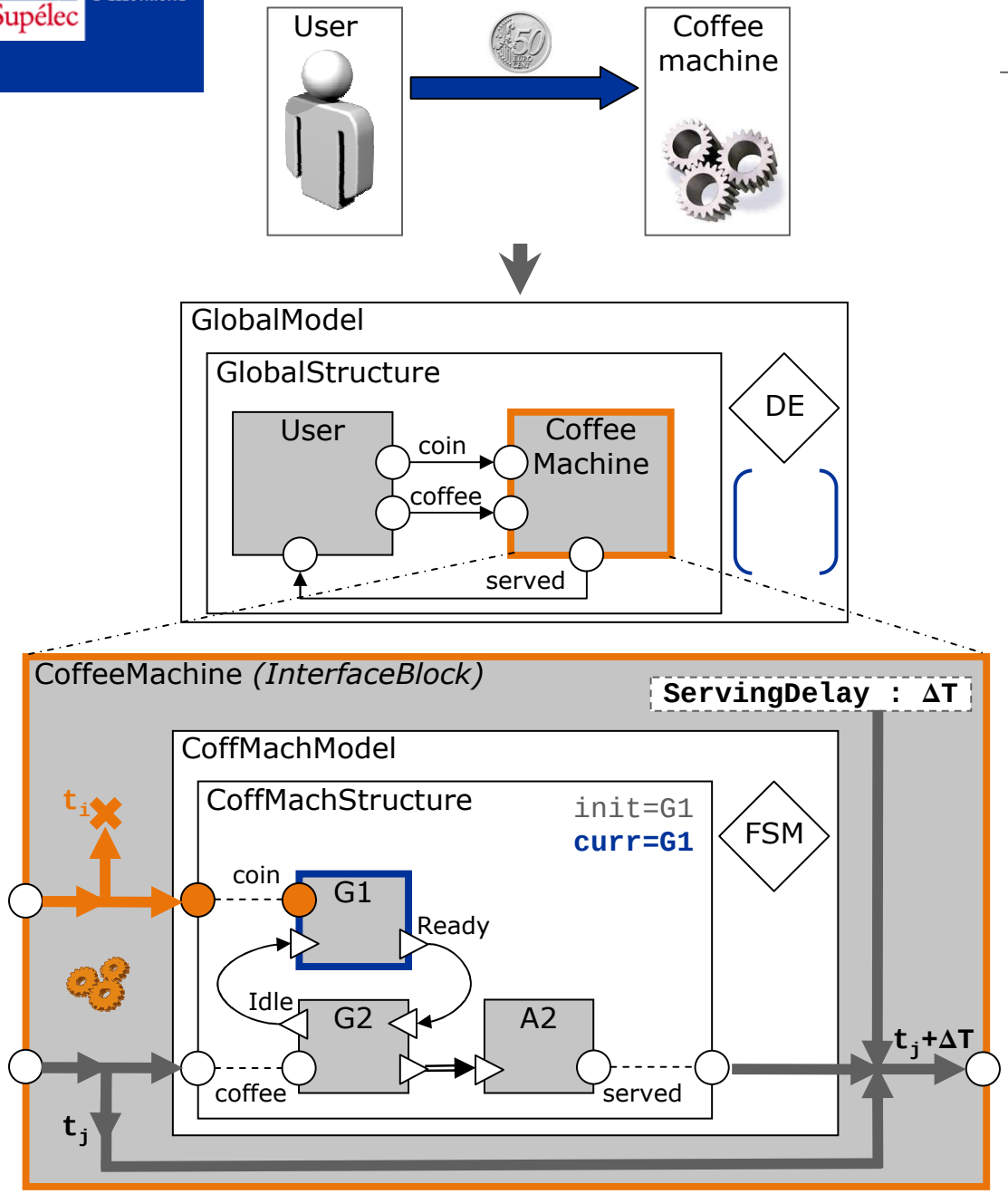


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

Running the model

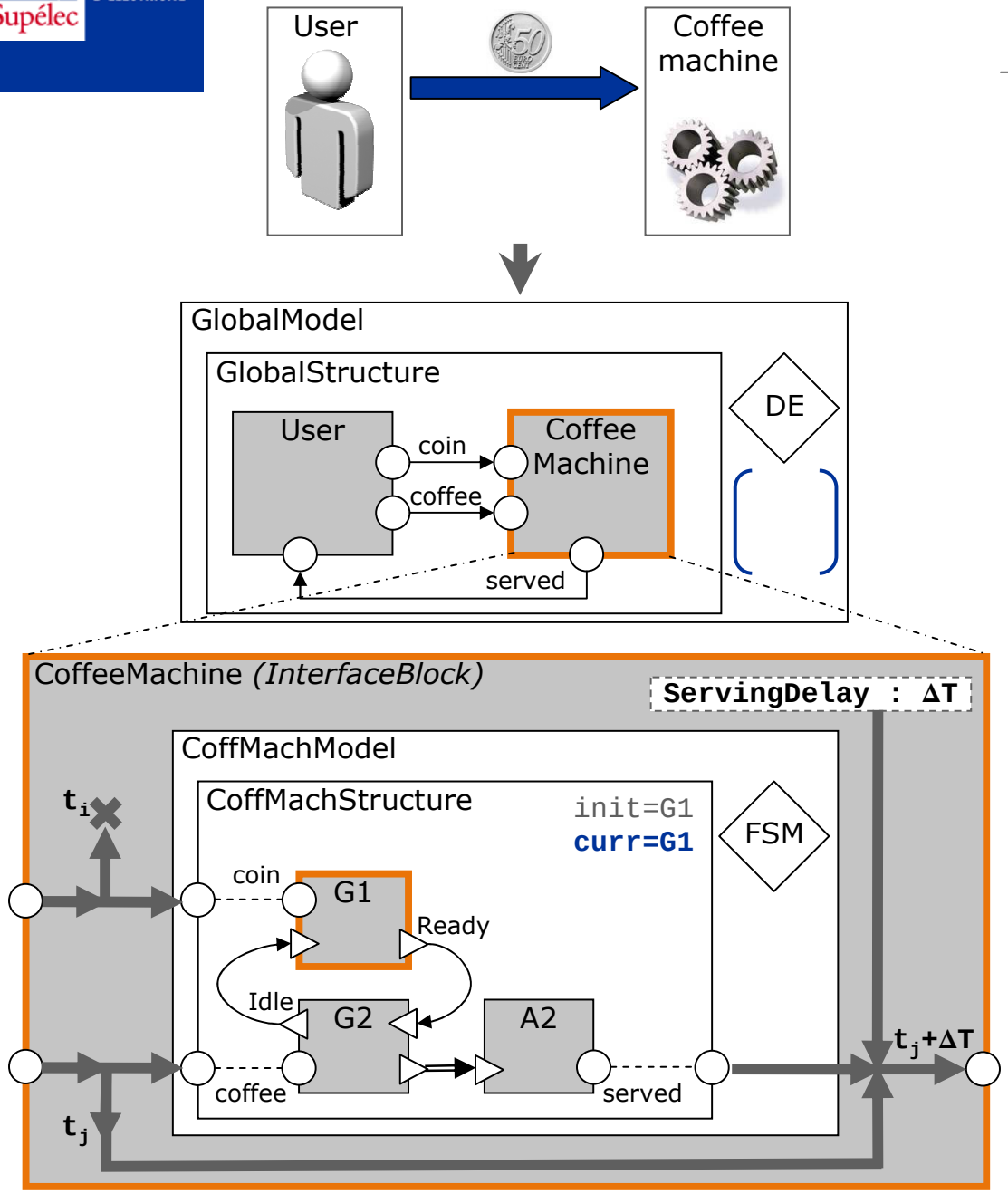


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

Running the model

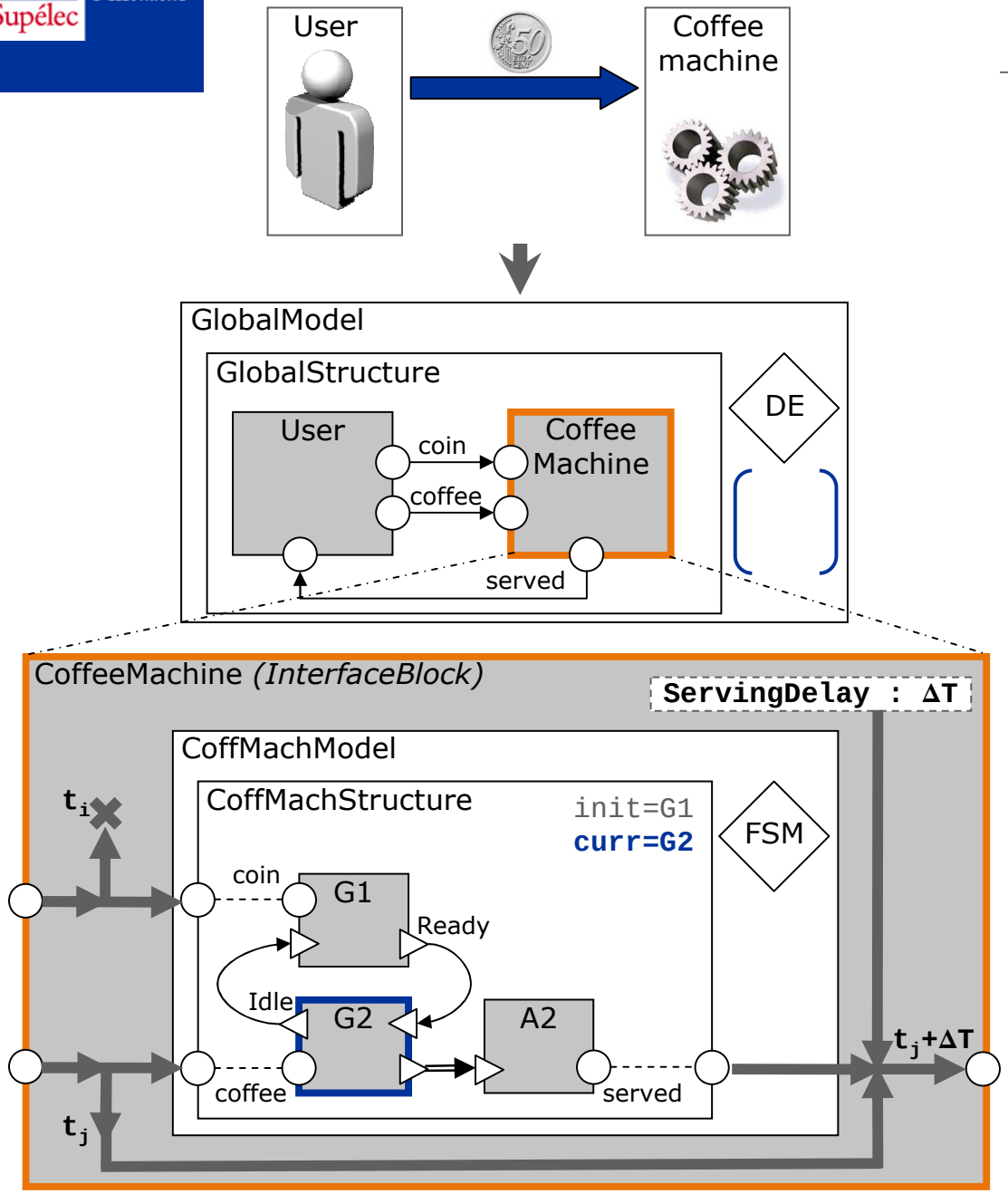


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

Running the model

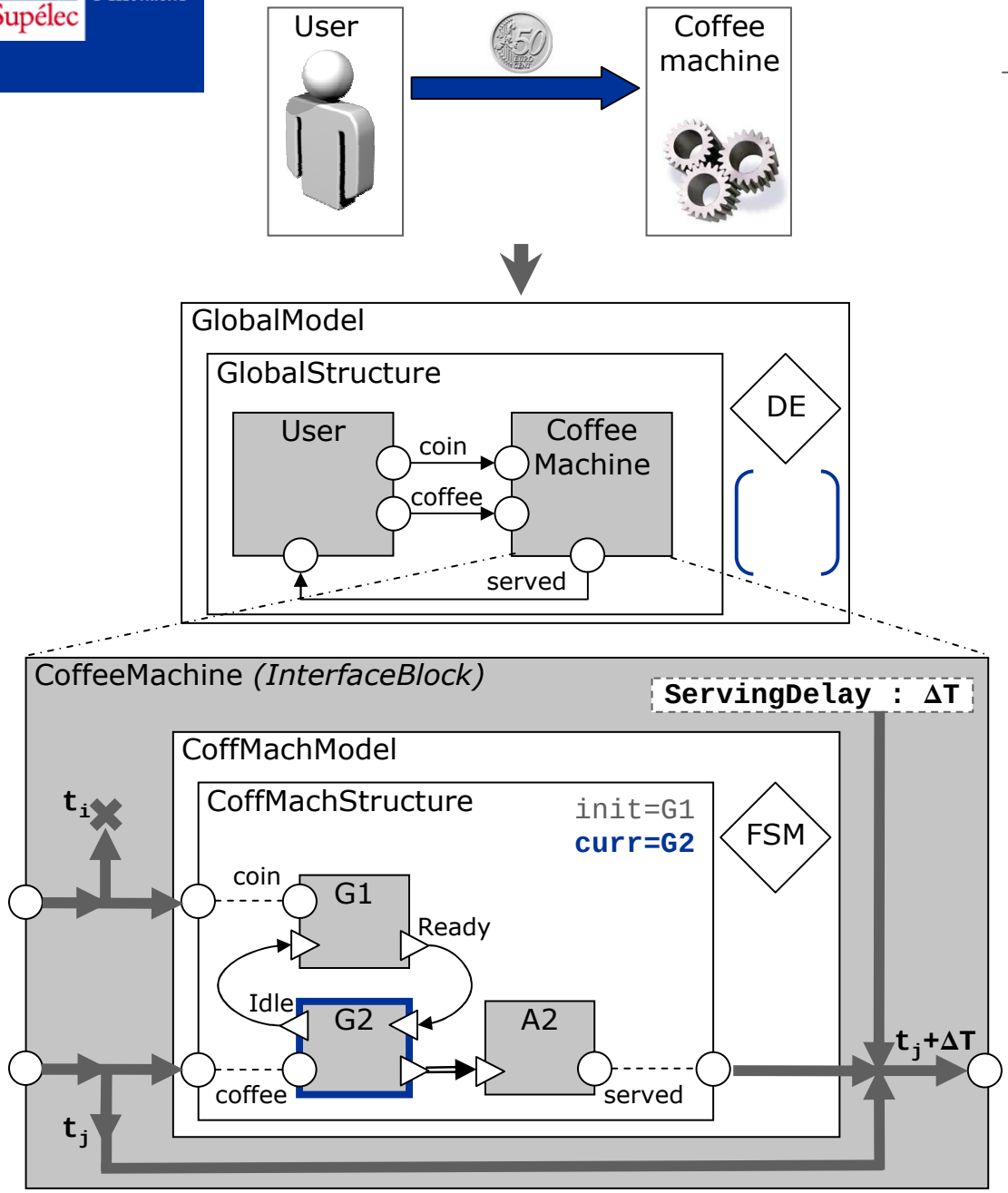


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

Running the model

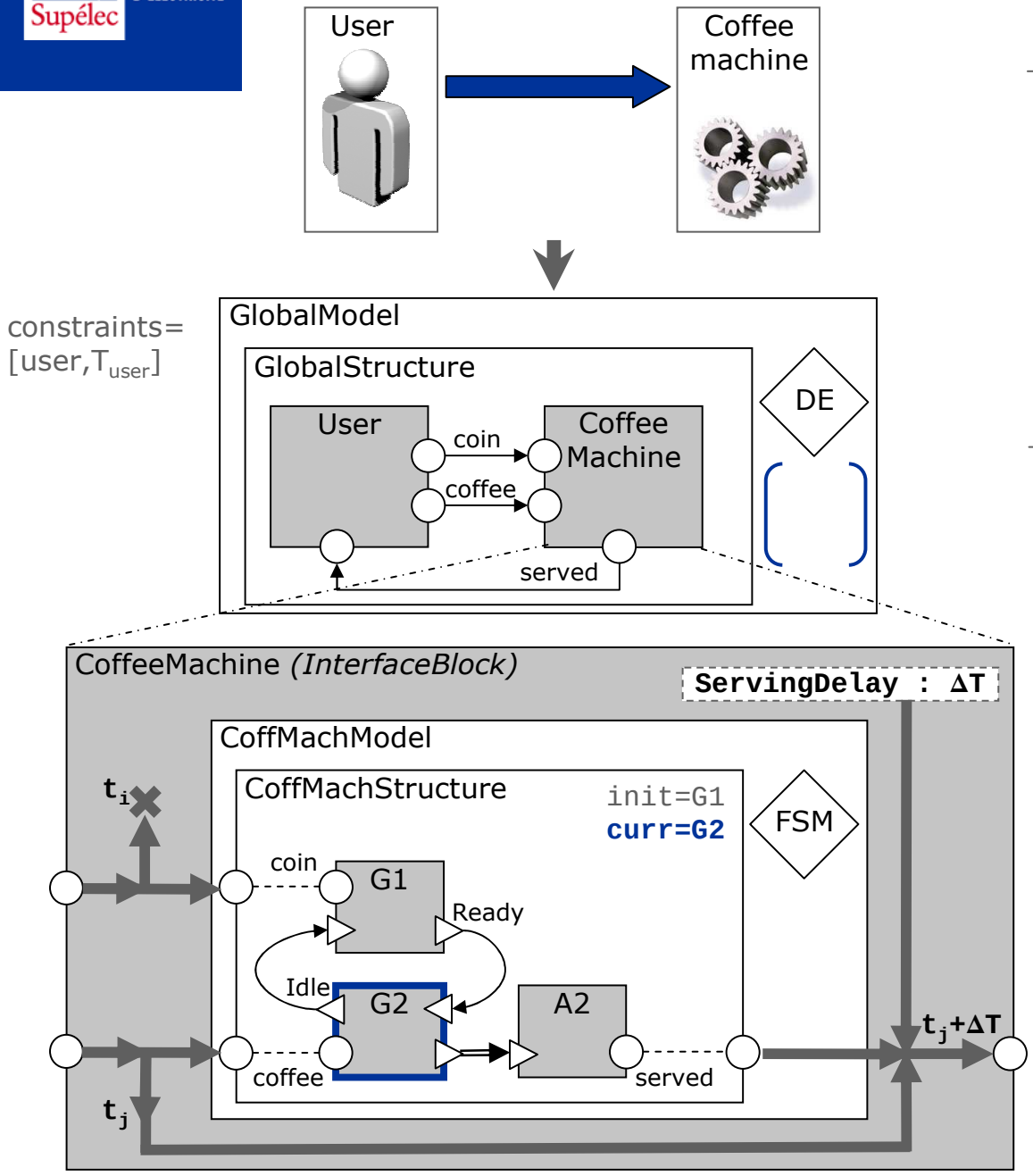


- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

"the user inserts the coin"

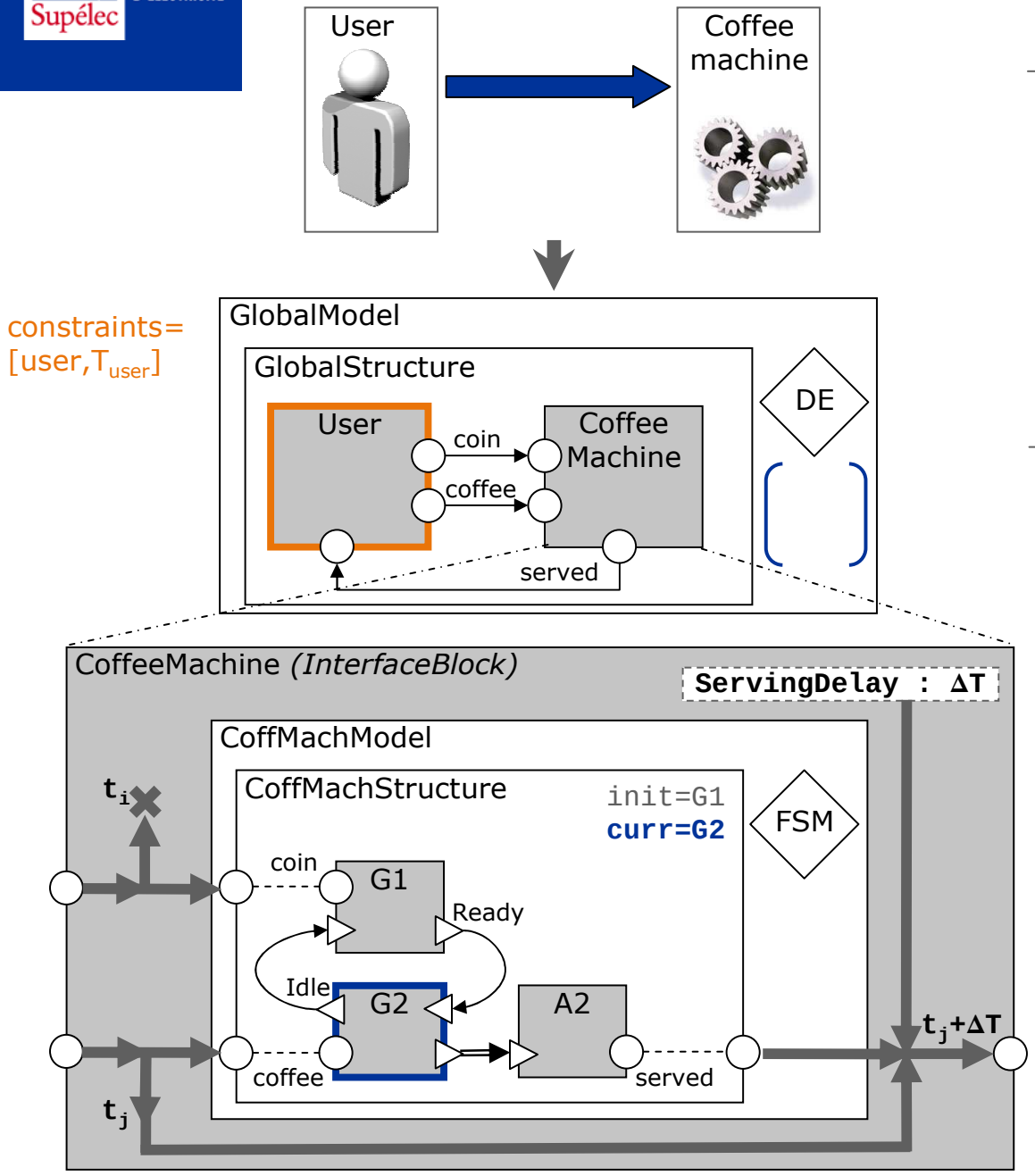
Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

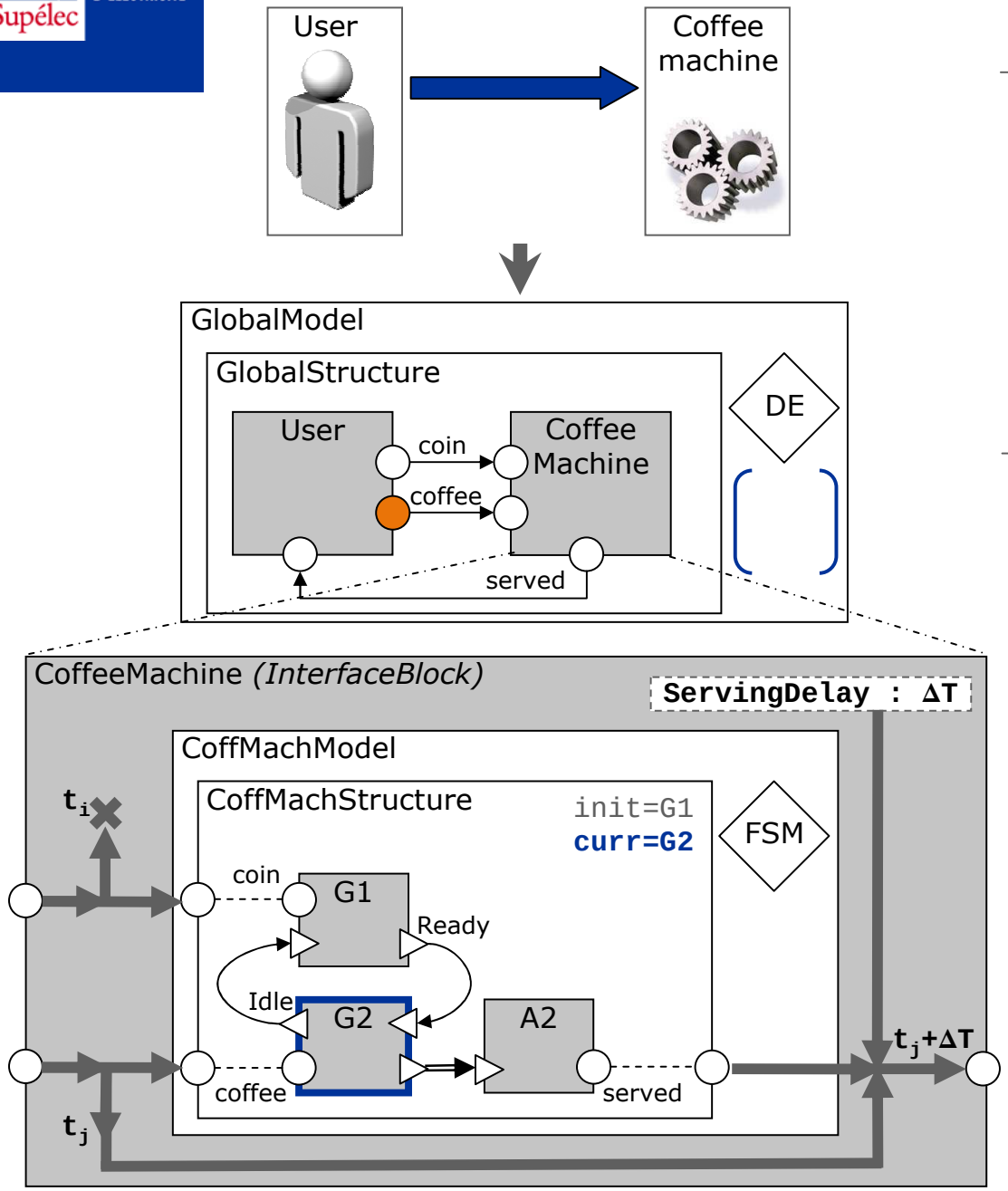
Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

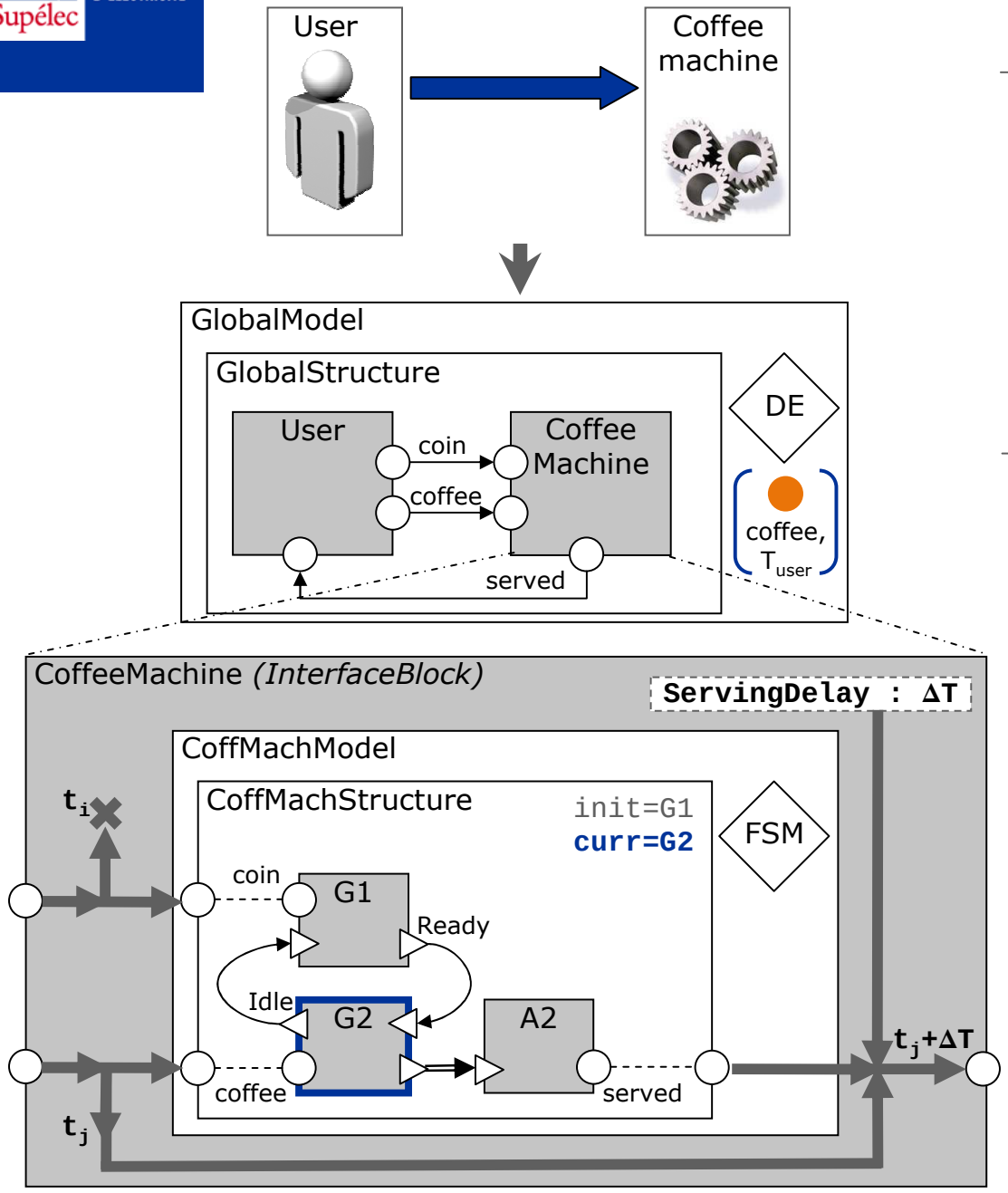
Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

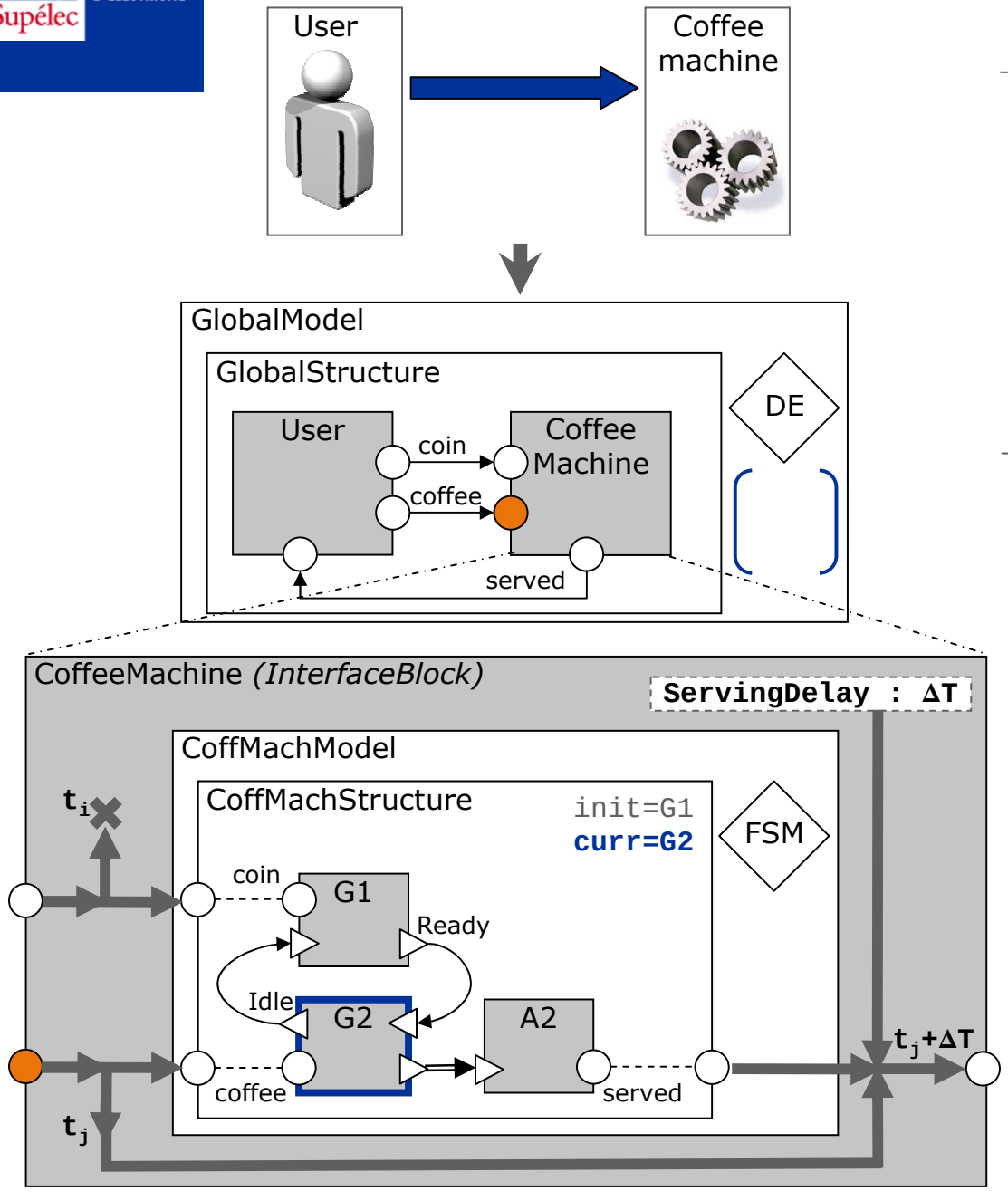
Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

Running the model



- Constraint on the user

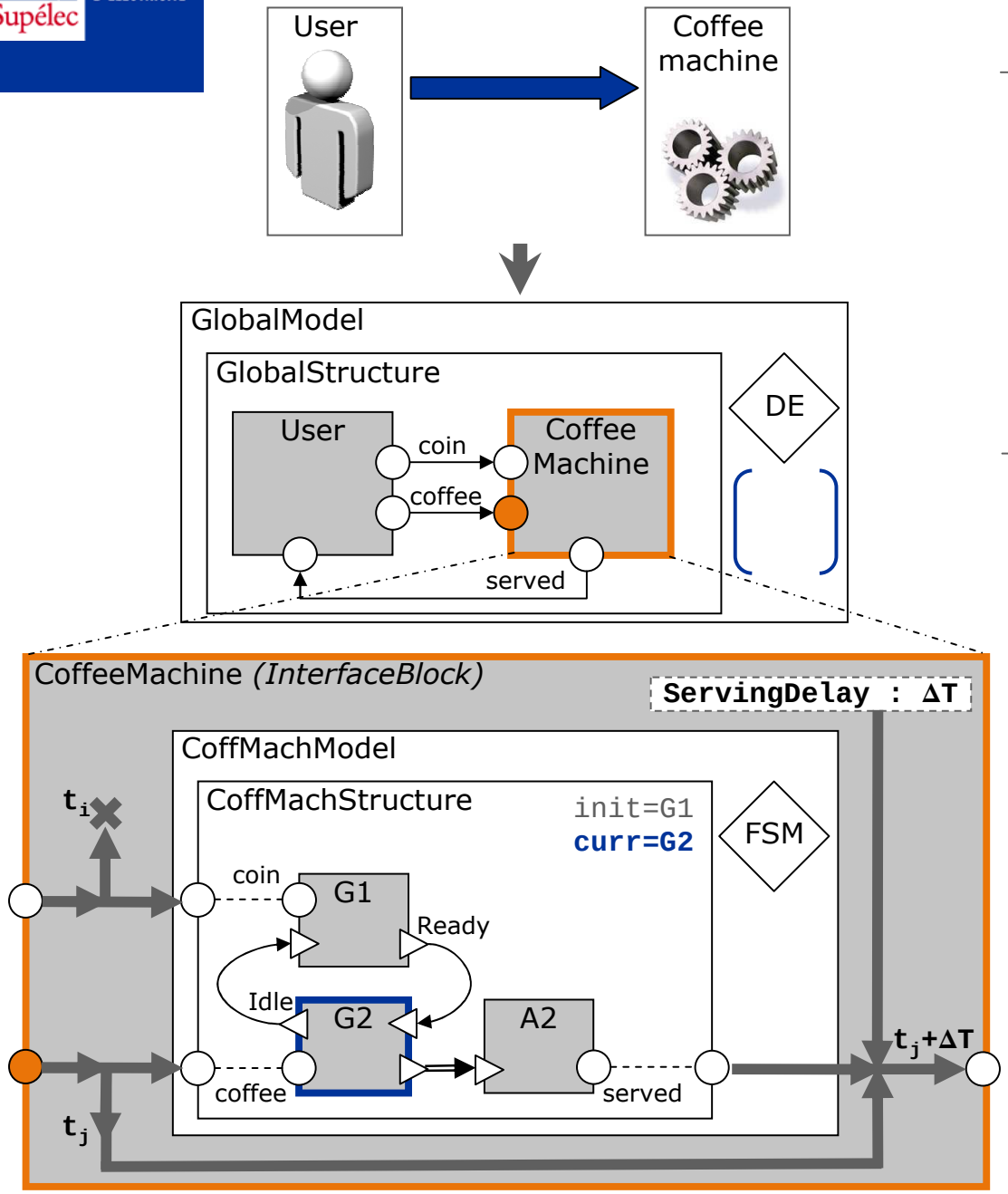
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

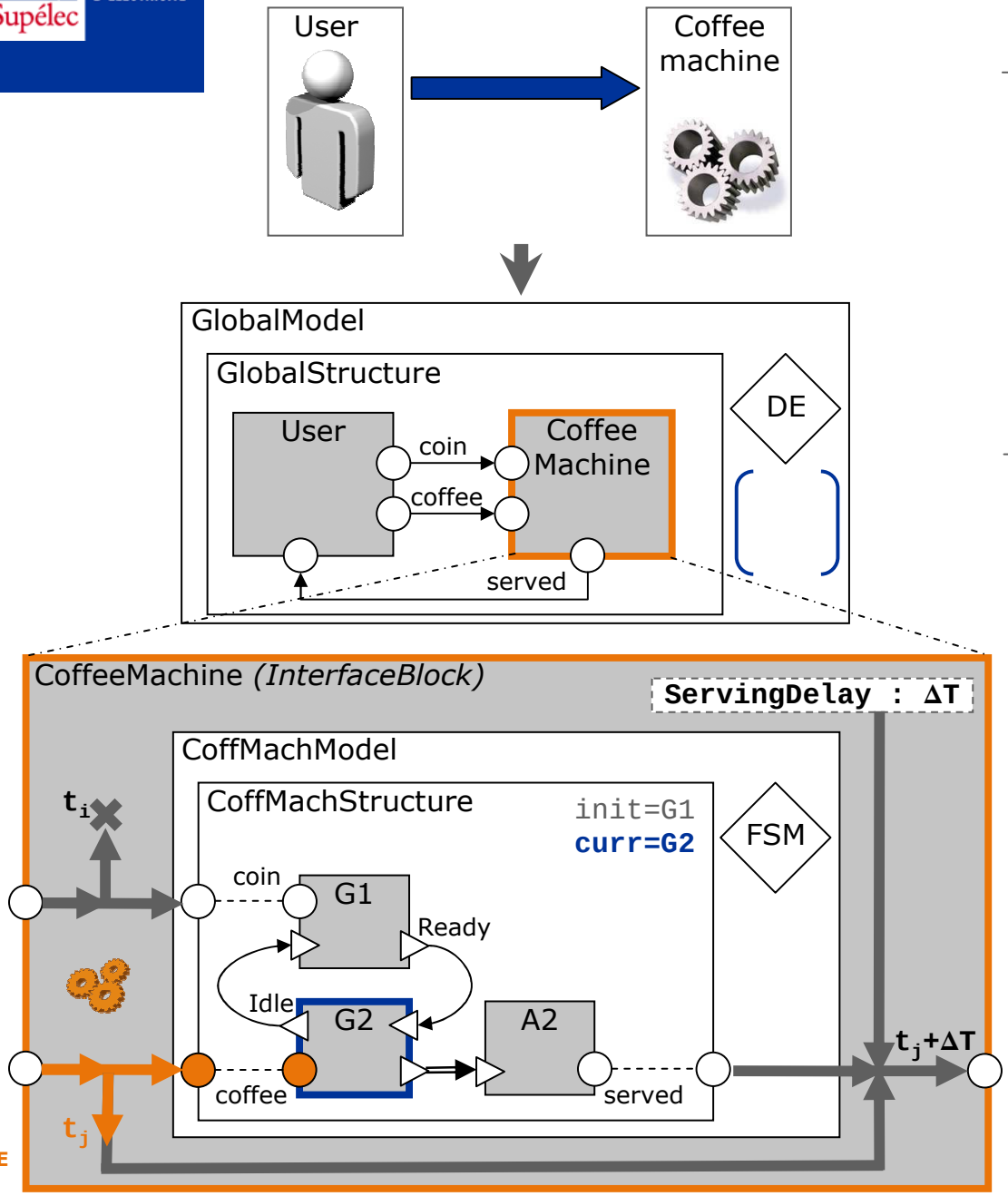
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

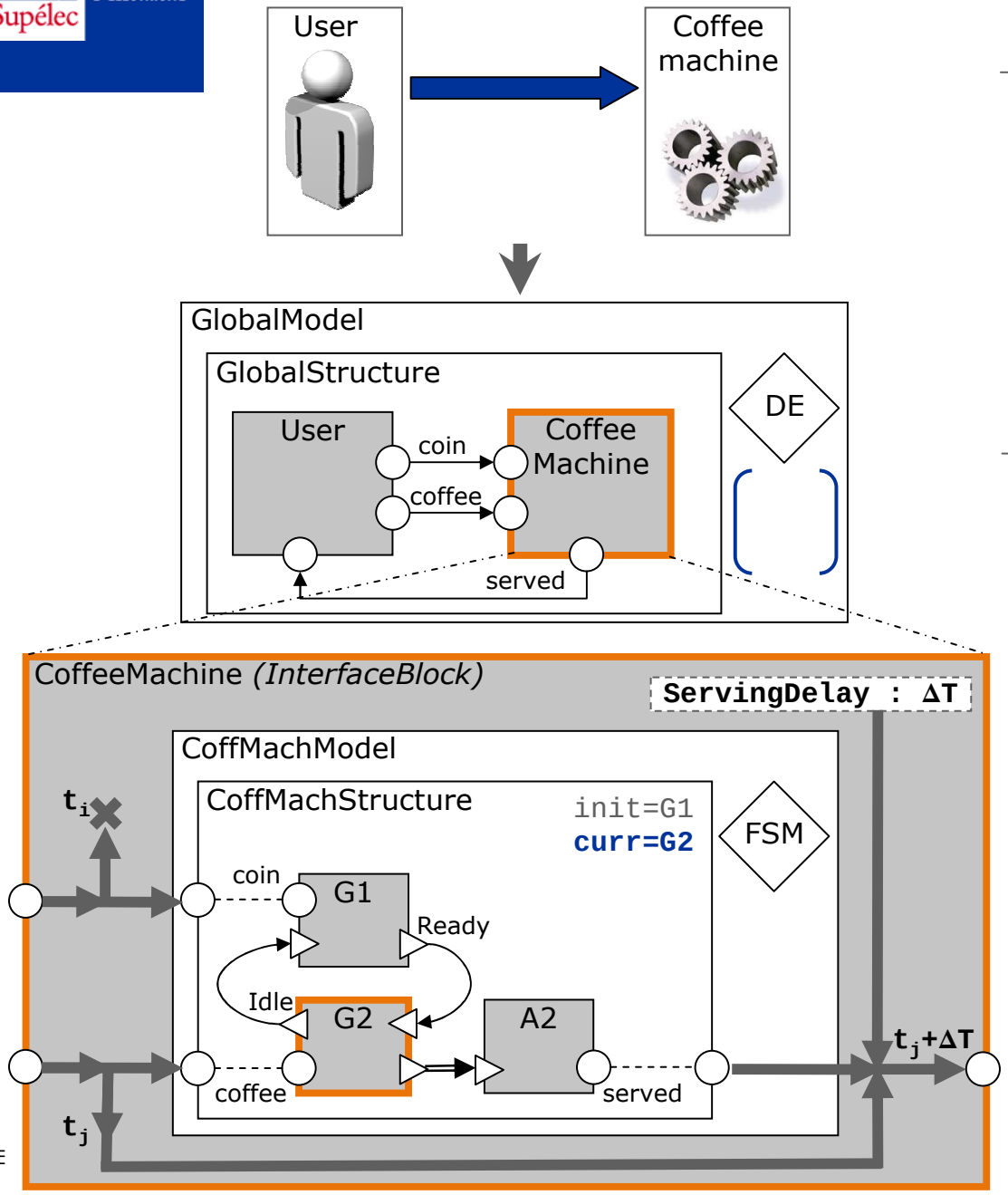
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

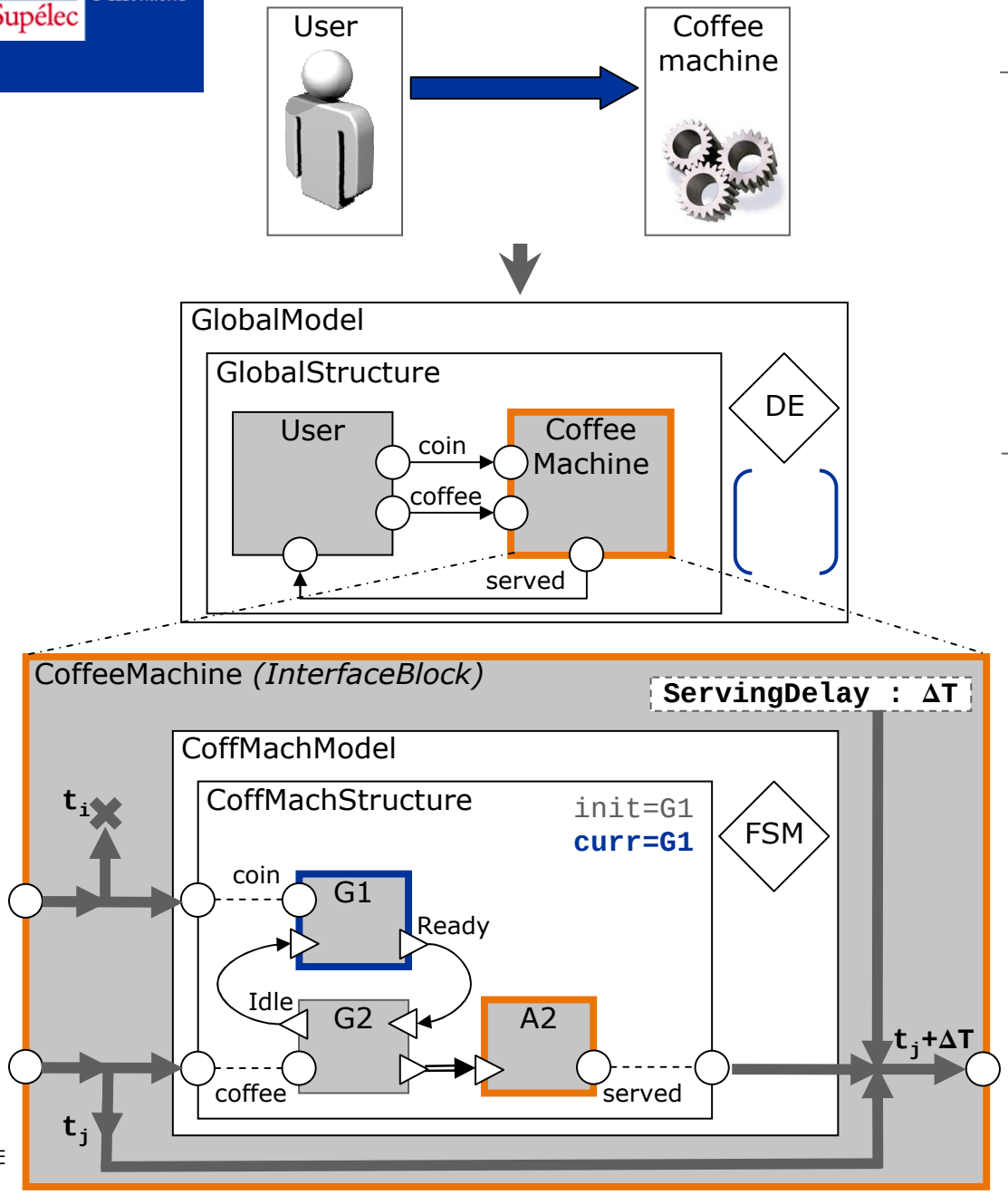
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

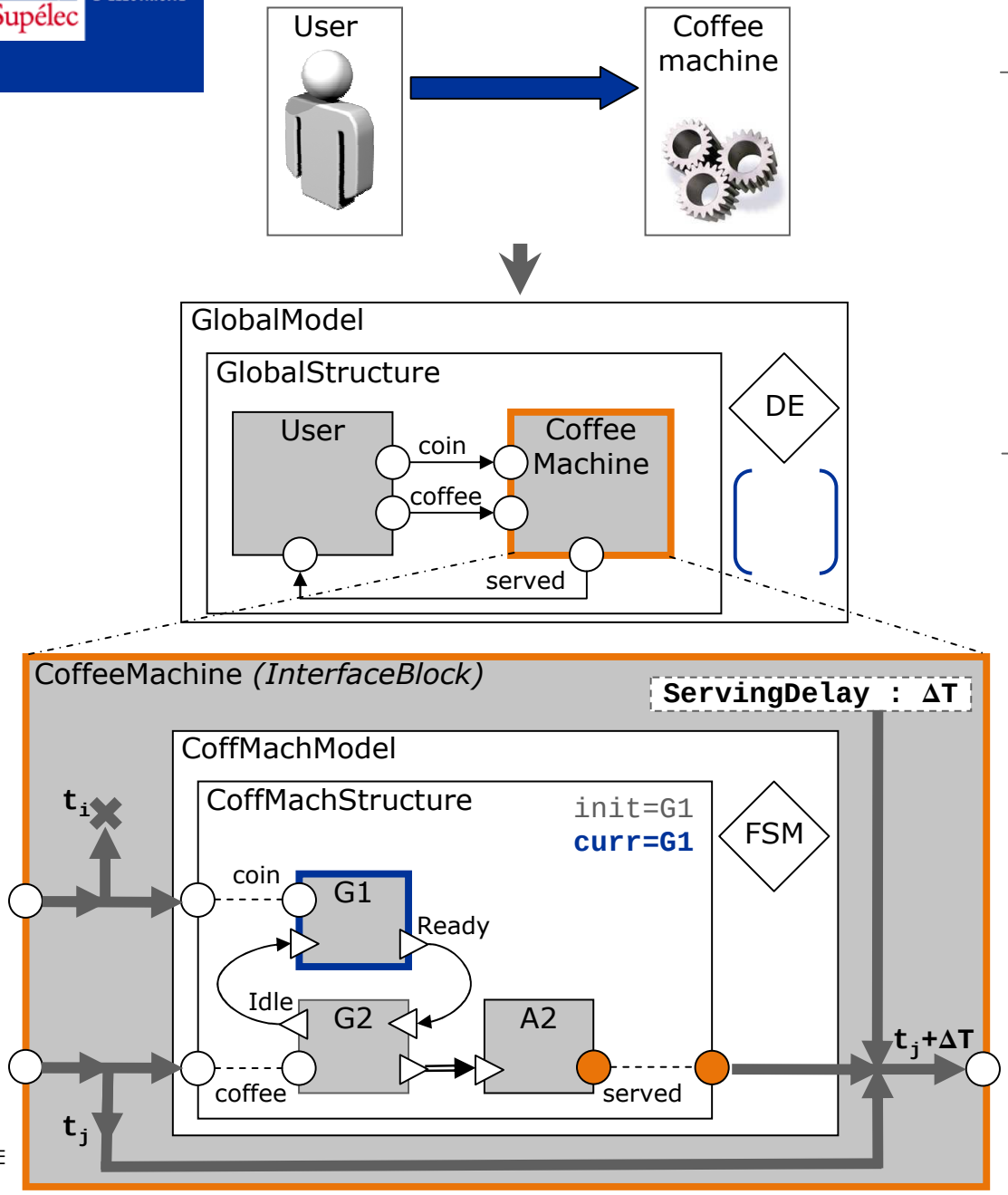
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

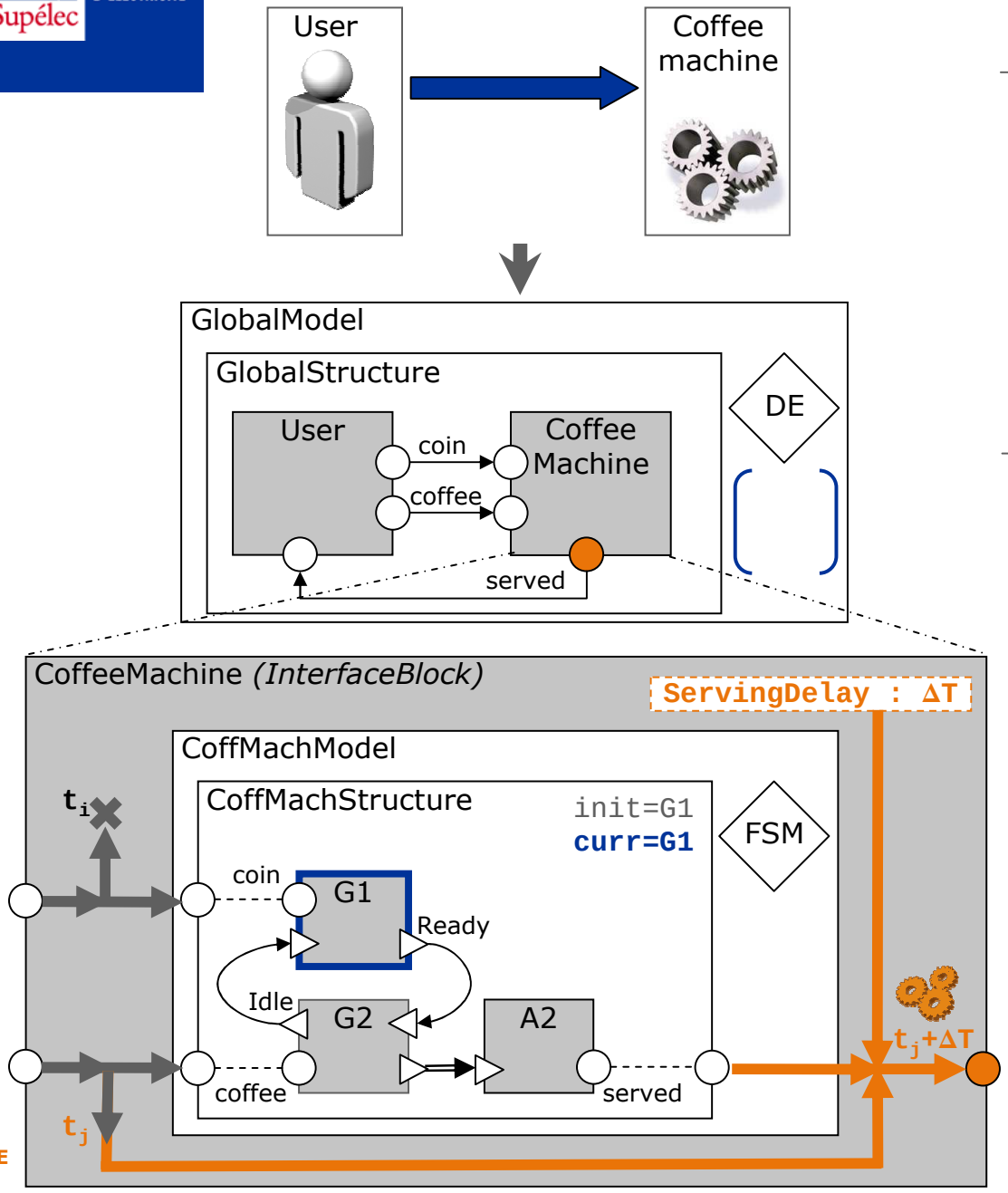
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model



- Constraint on the user

- First snapshot
  - $t_{DE} = 0$

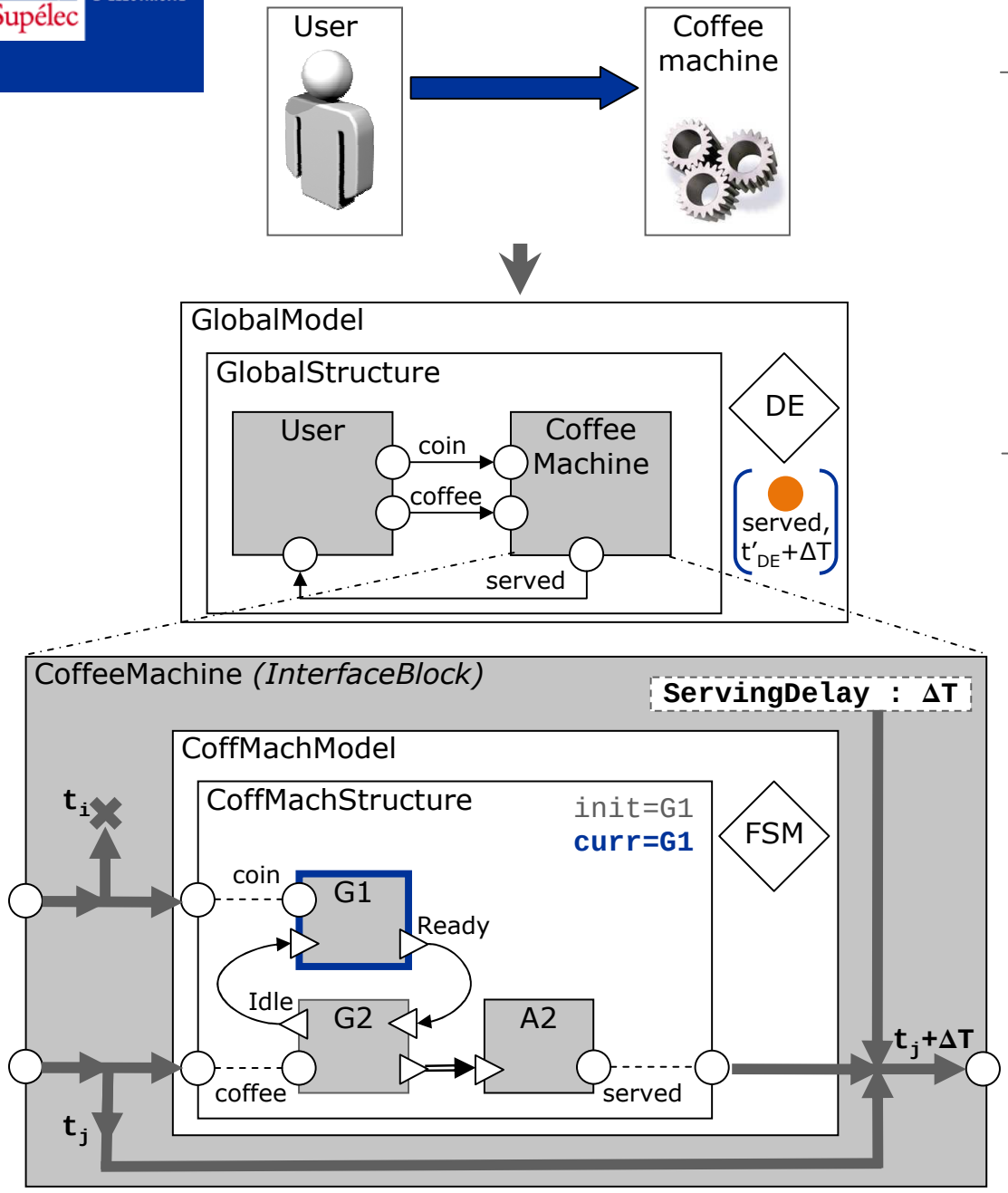
  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

$t_j + \Delta T = t'_{DE} + \Delta T$

Running the model



- Constraint on the user

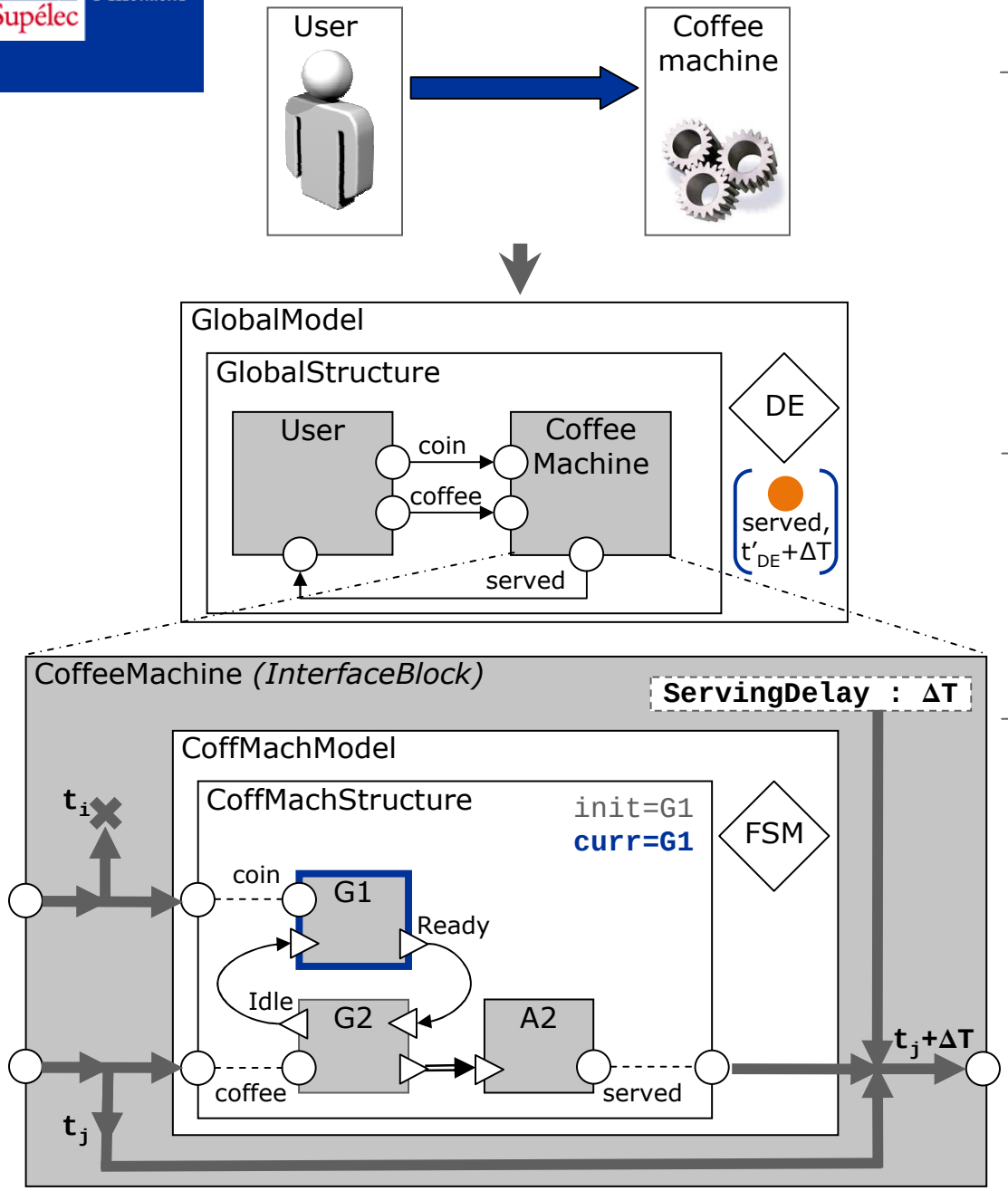
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user

- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model

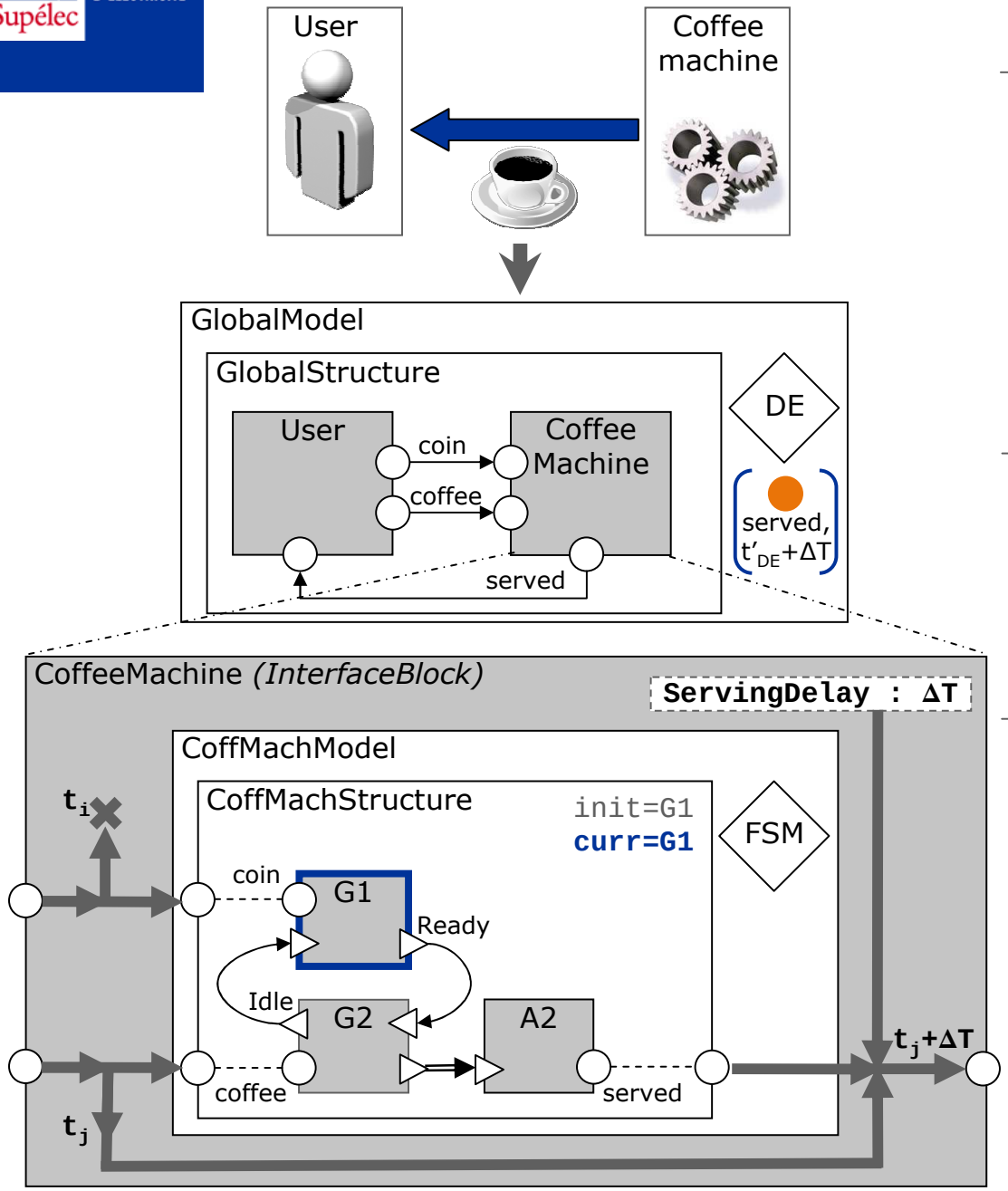


- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"

Running the model

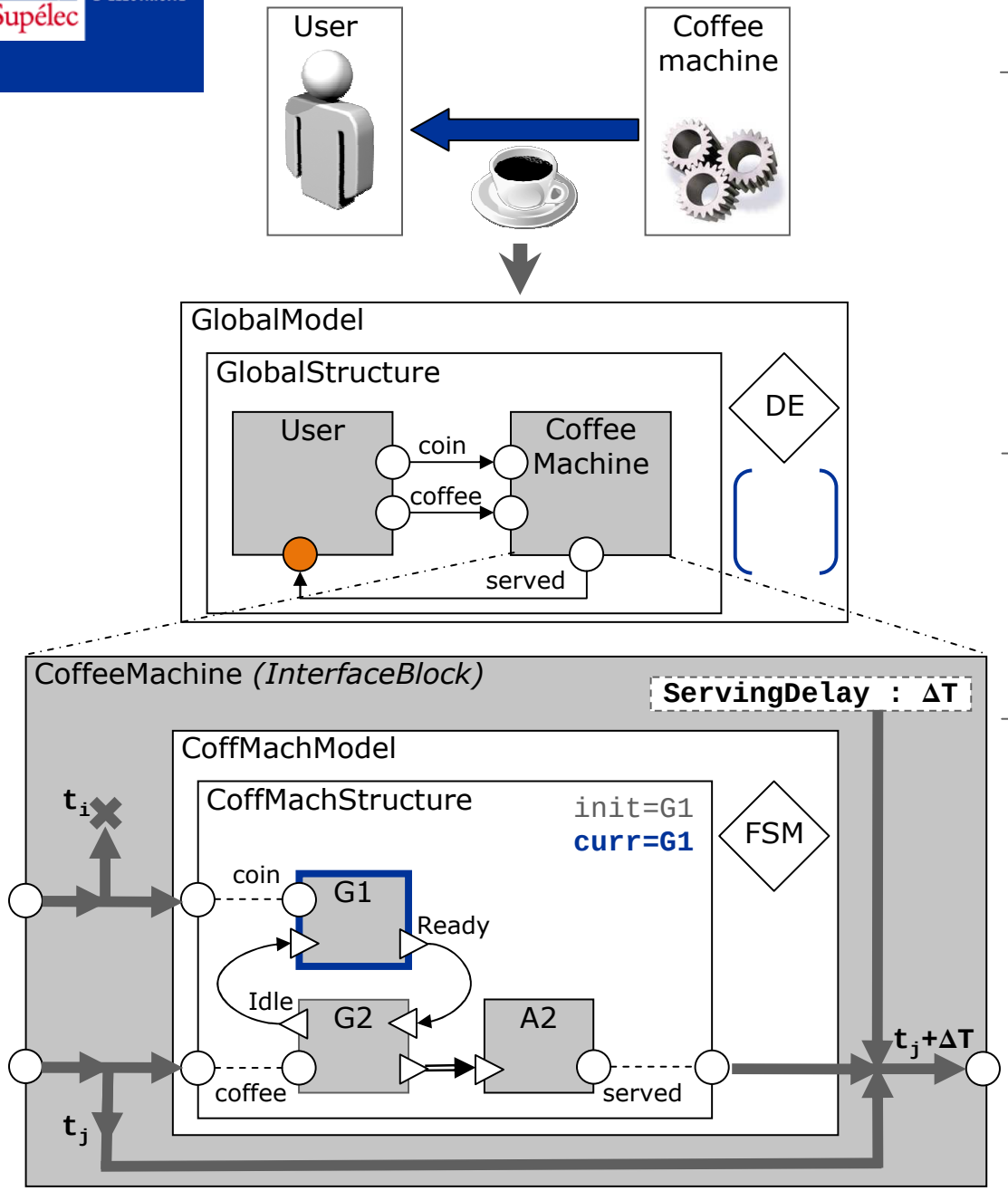


- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"
- Third snapshot
  - $t''_{DE} = t'_{DE} + \Delta T$

Running the model



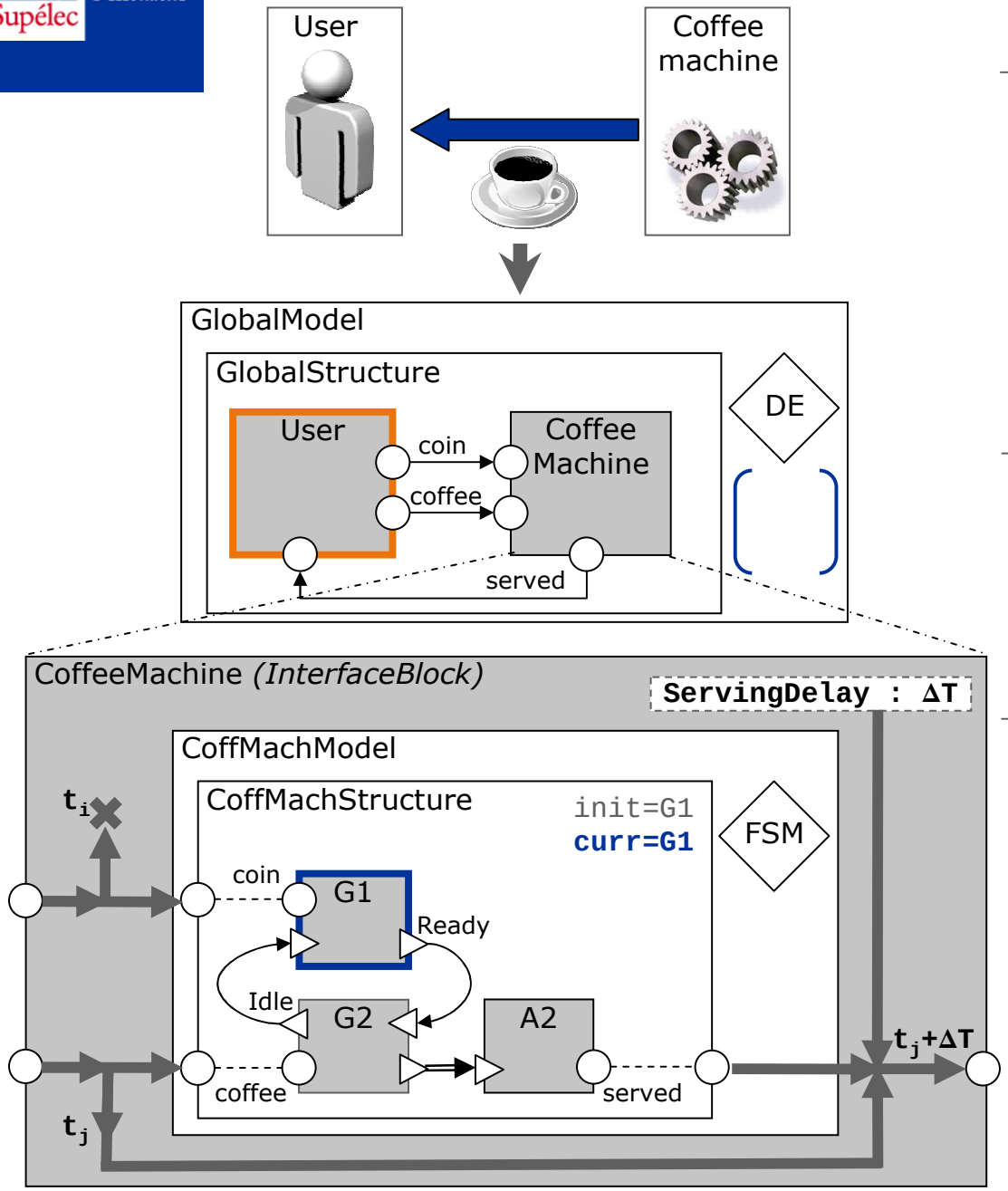
- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"
- Third snapshot
  - $t''_{DE} = t'_{DE} + \Delta T$

  "the machine delivers the coffee"

Running the model



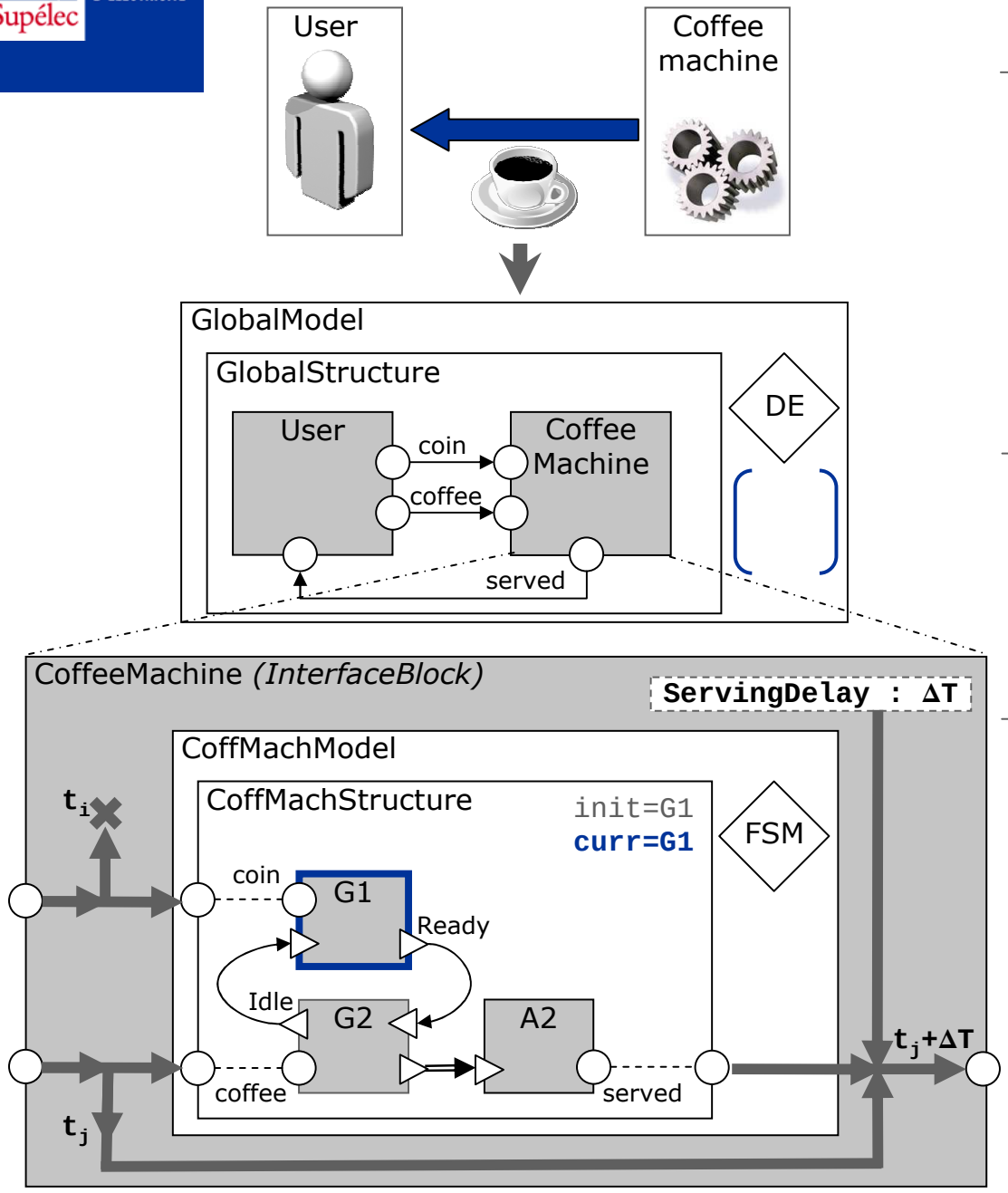
- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"
- Third snapshot
  - $t''_{DE} = t'_{DE} + \Delta T$

  "the machine delivers the coffee"

Running the model



- Constraint on the user
- First snapshot
  - $t_{DE} = 0$

  "the user inserts the coin"
  - Constraint on the user
- Second snapshot
  - $t'_{DE} = T_{user}$

  "the user pushes the button"
- Third snapshot
  - $t''_{DE} = t'_{DE} + \Delta T$

  "the machine delivers the coffee"