Research directions for MIKADO

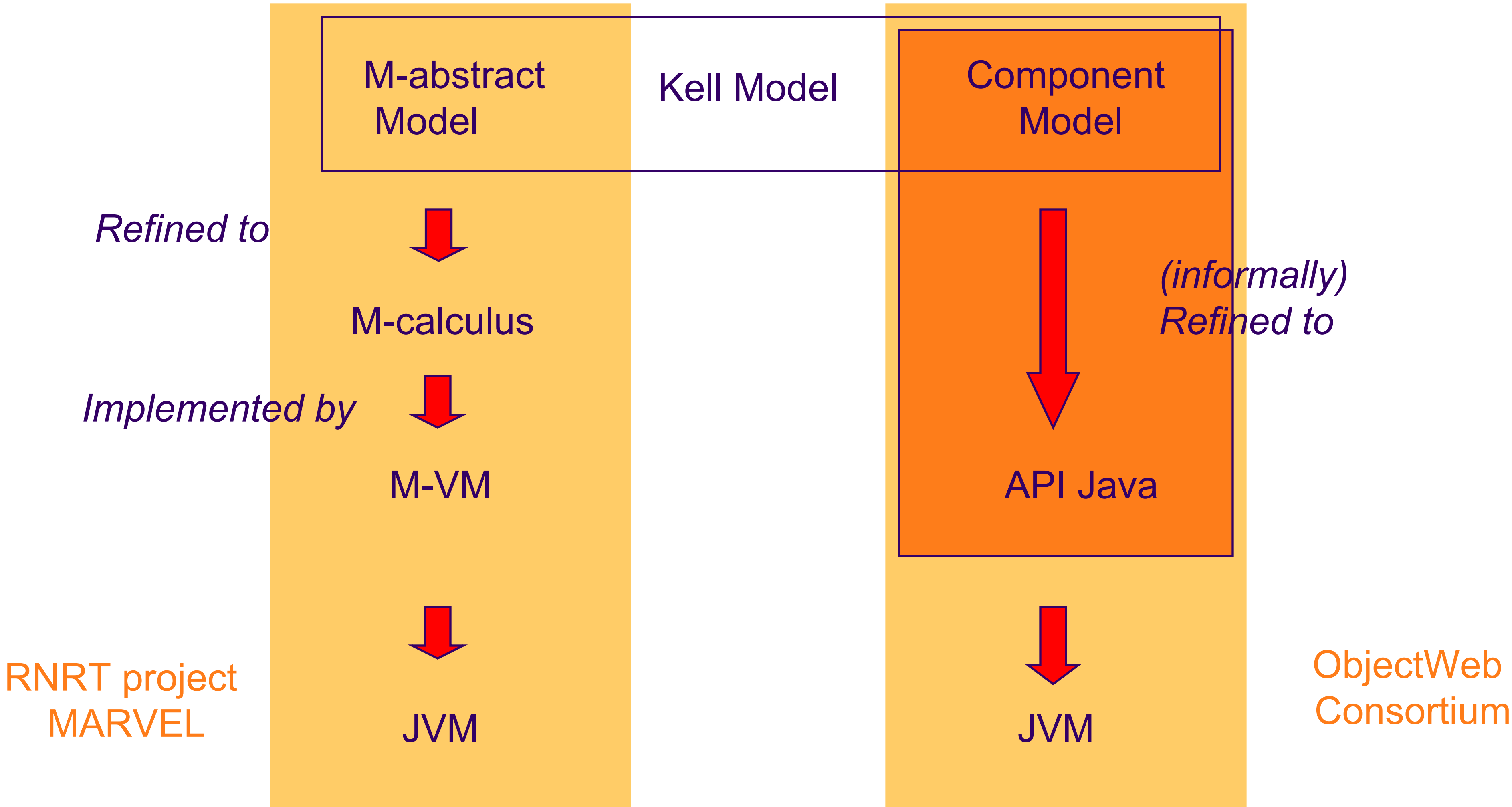
Florence Germain

France Telecom R & D, DTL/ASR

Kick-off MIKADO

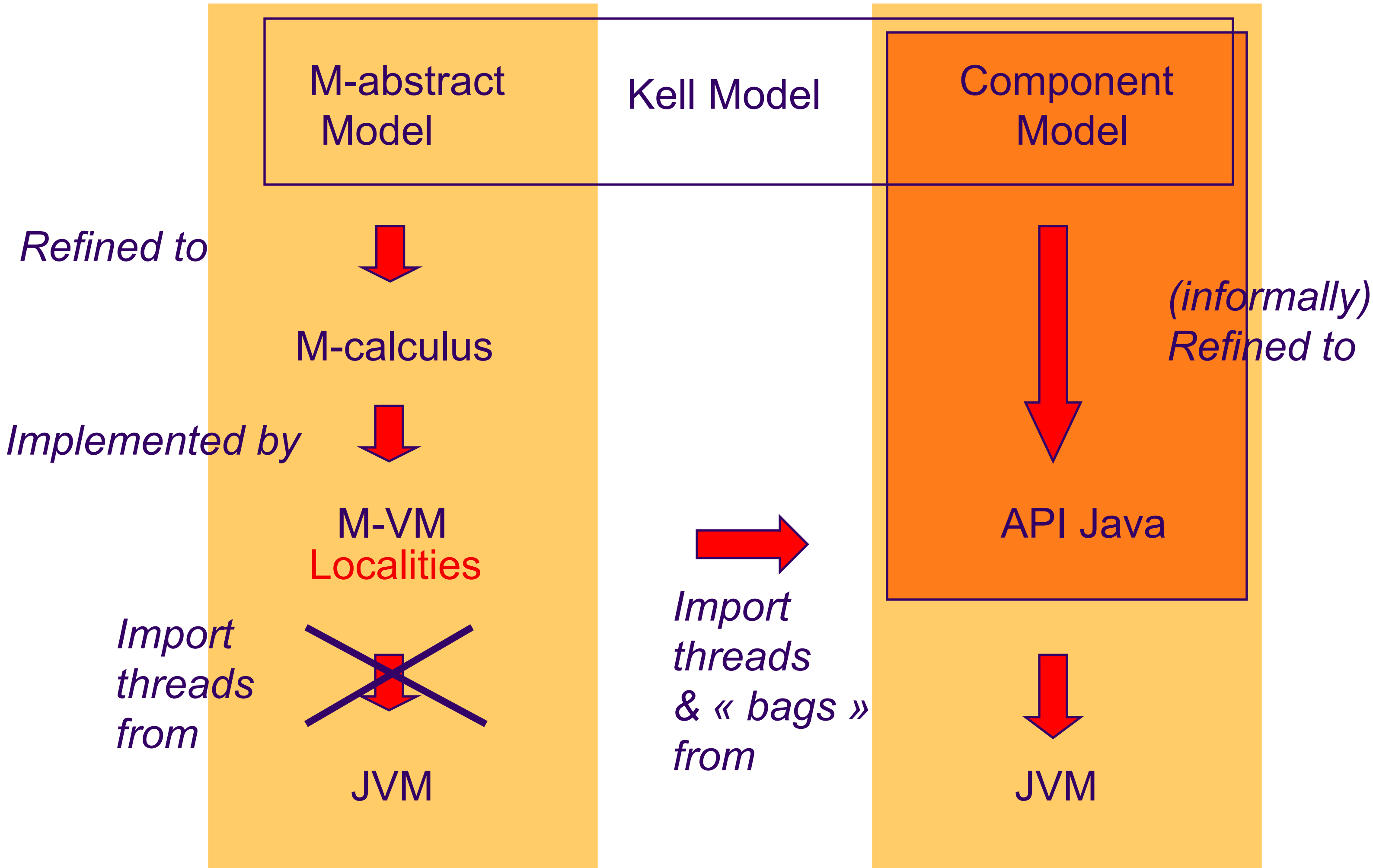
19-20 Février 2002

Historical context



France Télécom R&D

In terms of abstractions…



Plan

- Historical context
- Component Framework (CF)
  - the general model
  - the concrete model
- Perspectives
  - CF-based implementation of M-calculus
  - M-calculus as a formal basis for CF

CF: a support for component oriented programming

- Introduces the concept of “component” as a *runtime* structure (e.g. manifest during system execution)
- Assumes components to be *units* of *dynamic* system *configuration*

⬇

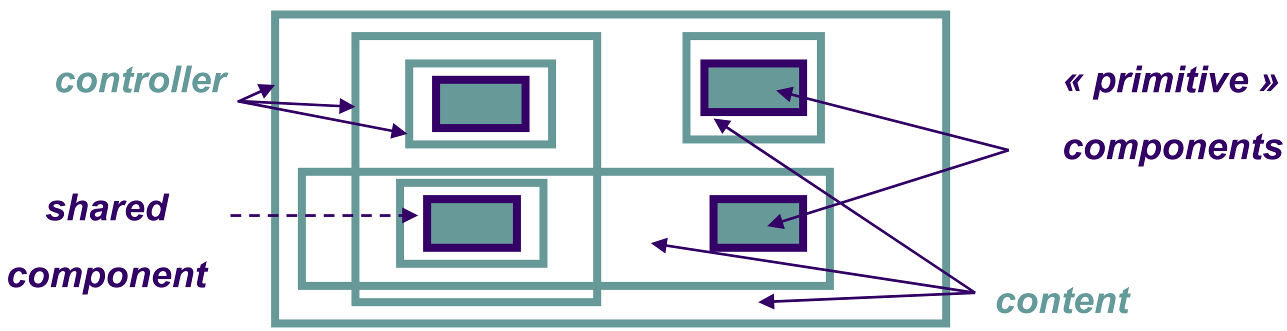
- Components allow programmers to dynamically access and manipulate their structure and behaviour
  - Structure = containment relationships + bindings
  - Behaviour = lifecycle
- Components exhibit introspection and intercession capabilities

CF is born from limitations...

- Architecture description languages (ADLs)
  - Allow system architecture description in terms of components, configuration, connectors, ports, roles …
  - Tools for validatidon, for application generation
  - Limitations: cannot deal with dynamic reconfiguration (changes in the implementation not reflected in the architecture)
- Industrial composition frameworks (EJB's, COM, CCM)
  - Introduces containers, supporting technical services such as persistance, transactions, security, ...
  - Limitations: composition of components is not a component, semantic of components is given by technology (not accessible by programs)

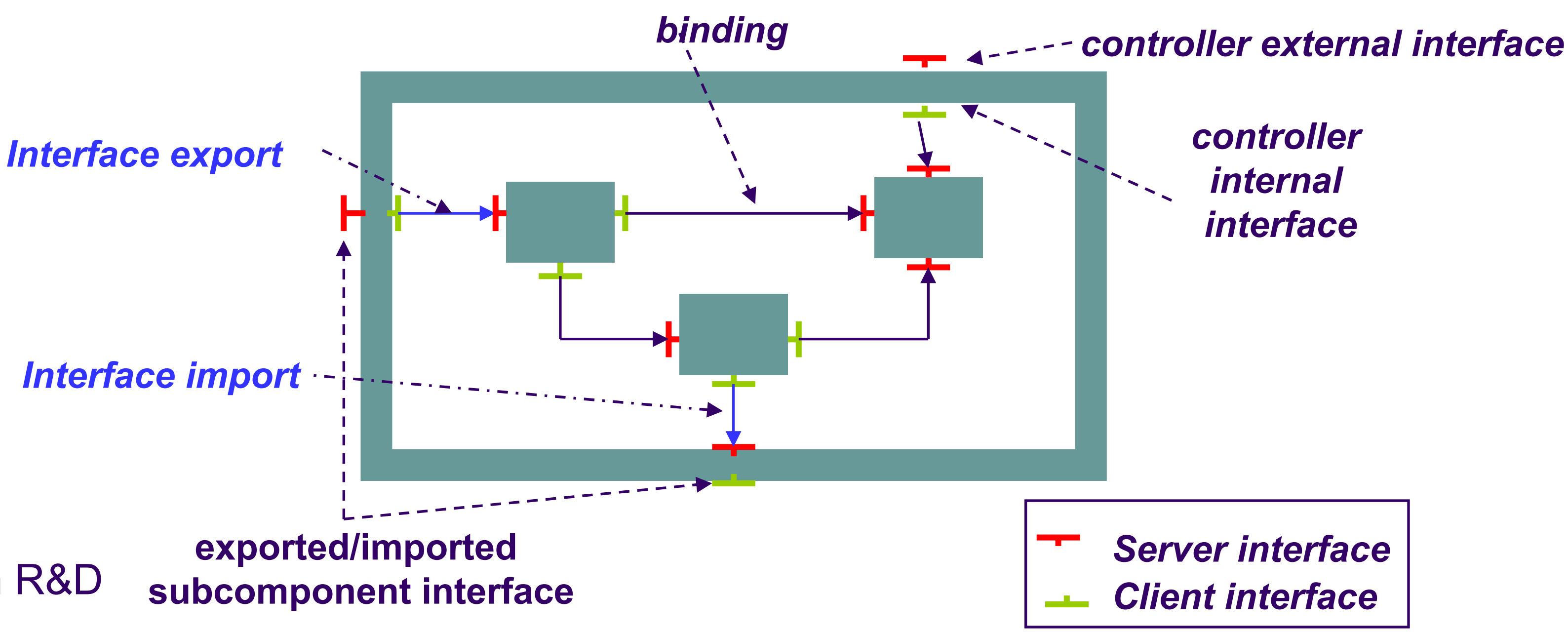
CF General Model: structure of kells

- Three mains concept: names, interfaces and kells
- A kell = a membrane + a plasm
- A plasm encloses a finite number of kells (recursivity)
- The membrane embodies the control behaviour of the kell:
  - intercepts incoming or outcoming signals
  - provides an explicit representation of the kells in the plasm
  - manages the activity of the kells in the plasm
- Different kells may have overlapping plasm
  - semantic of overlapping given by the enclosing membrane



CF General Model: interfaces

- Kells interact with environment through signals at identified access points called interfaces, and referenced by names
- Arguments of signal may be names or values, or kells (kells goes through membranes)
- Membranes determines visibility for interfaces of the plasm
- Membranes may have internal interfaces, visible from the plasm only



CF General model: kells behaviour

- Autorized behaviour of kells = {transitions}
- A transition specifies
  - the original kell
  - a finite set of oncoming signals
  - a finite set of outgoing signals
  - a finite set of resulting kells
- Example of a kell factory

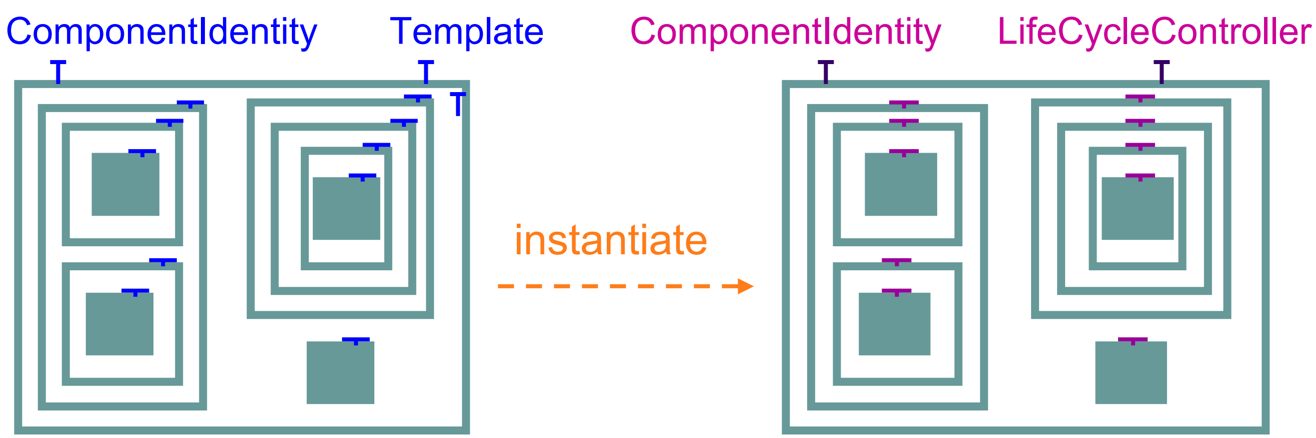
- Signals, interfaces and kells are typed in the general model (statically verifiable predicate constraining nature and behaviour)

CF Concrete model: API core

- Package Root:
  - ComponentIdentity, InterfaceReference
- Package Type:
  - ComponentType, InterfaceType, TypeIdentity
- Package Factory:
  - Template, TemplateFactory
- Package Control:
  - ContentController, BindingController, LocalBindingController, AttributeController, LifeCycleController

CF Concrete model: core API

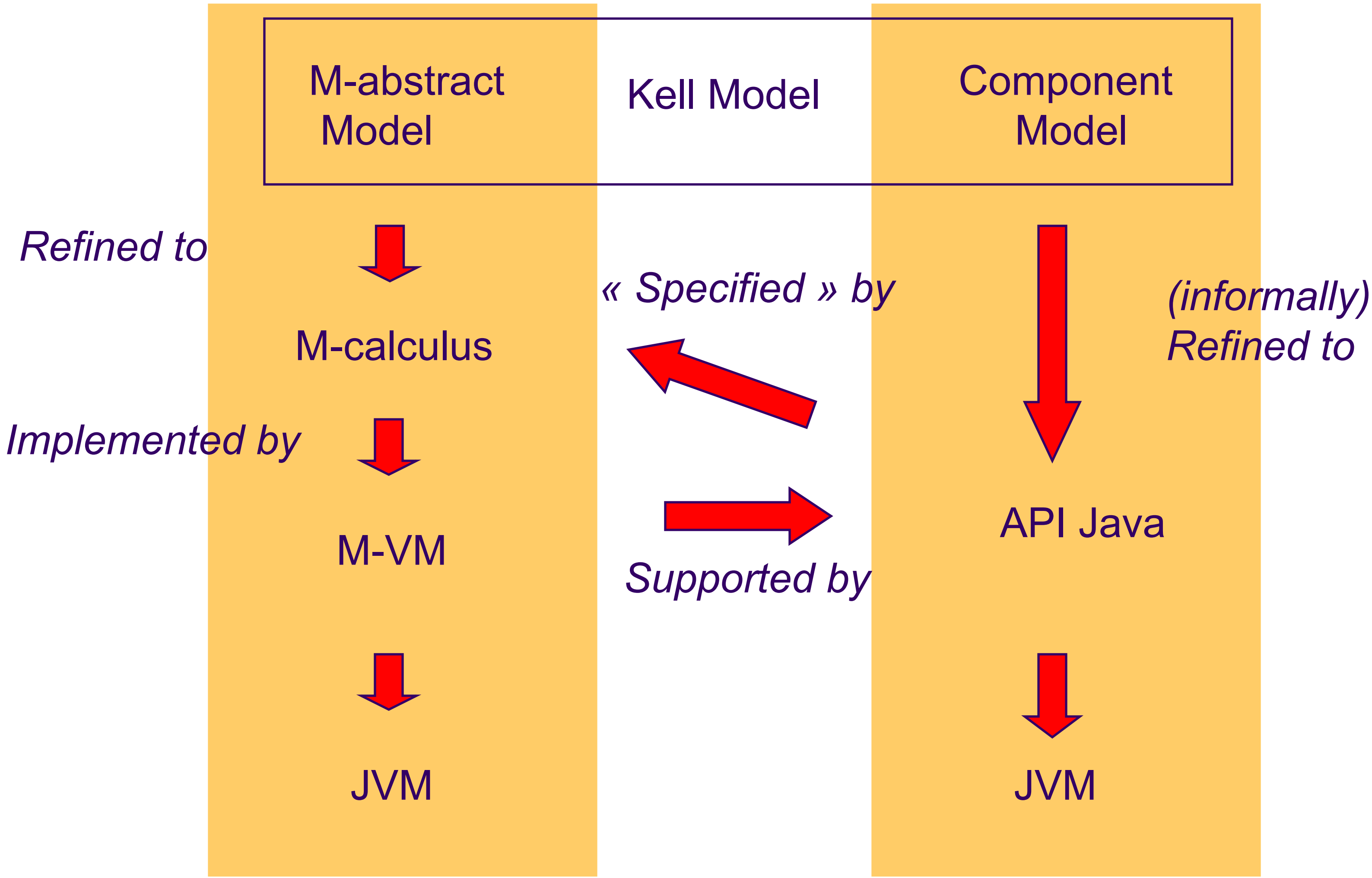
- Examples
  - Template: instantiate()
  - TemplateFactory: createTemplate(ComponentType type)
  - ContentController: add(remove)SubComponent(), getSubComponents()
  - BindingController: lookup(), bind(), unbind()
  - LifeCycleController: getState(), start(), stop()



CF Concrete model: increments

- Increments for programming support
  - Containers "A la EJB" (AttributeController & LifeCycleController)
  - Domain Supports (BindingController & ContentController & LifeCycleController)
  - Template Components (Template)
  - Factories (TemplateFactory)
- Increments for composition support
  - Components for contractual composition, for behavioural composition,...
- Increments for administration support
  - Components for observation, for supervision,...
- Increments for configuration support
  - Components for persistence, for security, ...

Perspectives



From the M-calculus point of view ...

- CF as a support for the implementation of M-calculus
  - explicitation of the concept of bag, at run-time level
  - leads to a new CF increment
- CF as a support for the implementation other (every?) langages/calculus conforming to M-calcul
  - exercice of transposition, at abstract level only
  - may prove the validity of « *runtime patterns* » for domains-oriented programming

From CF point of view, ...

- M-calculus, for specification of the behaviour of components
  - domain-oriented reasoning will be easier in M-calculus than in Java…
- M-calculus + logical tools, for specification (and proof?) of invariants for components
  - theorems will be easier to express (and to prove!) in M-calcul than in Java …
- Automatic code generation from M-calculus to CF …