

*comments inside..
See also JOOP-Sept. 99 for
details*

Traceability Management with UML through Goal-Oriented Objects (v2.0)

A proposal for the UML's Activity Diagram

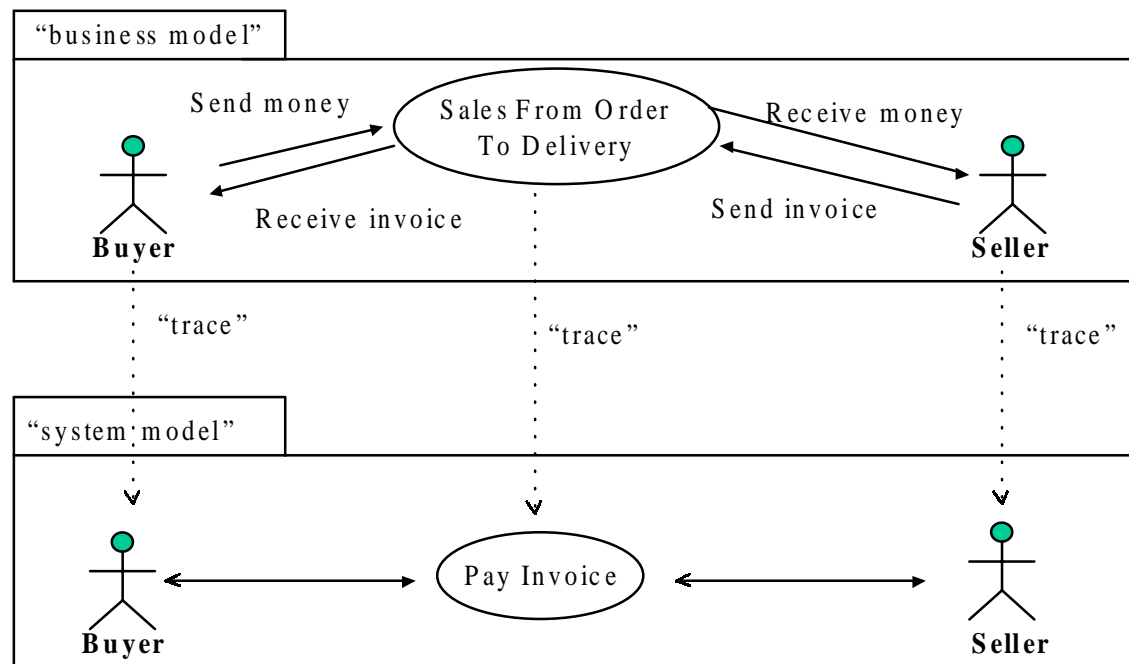
OMG- AD Task Force Meeting
November 17th, 1999

Birol Berkem

e-mail : berkem@cnam.fr
36, Av. du Hazay
95800 Paris- Cergy / France
Tel : +33.1.34.32.10.84

Gathering Requirements and Traceability Management with UML

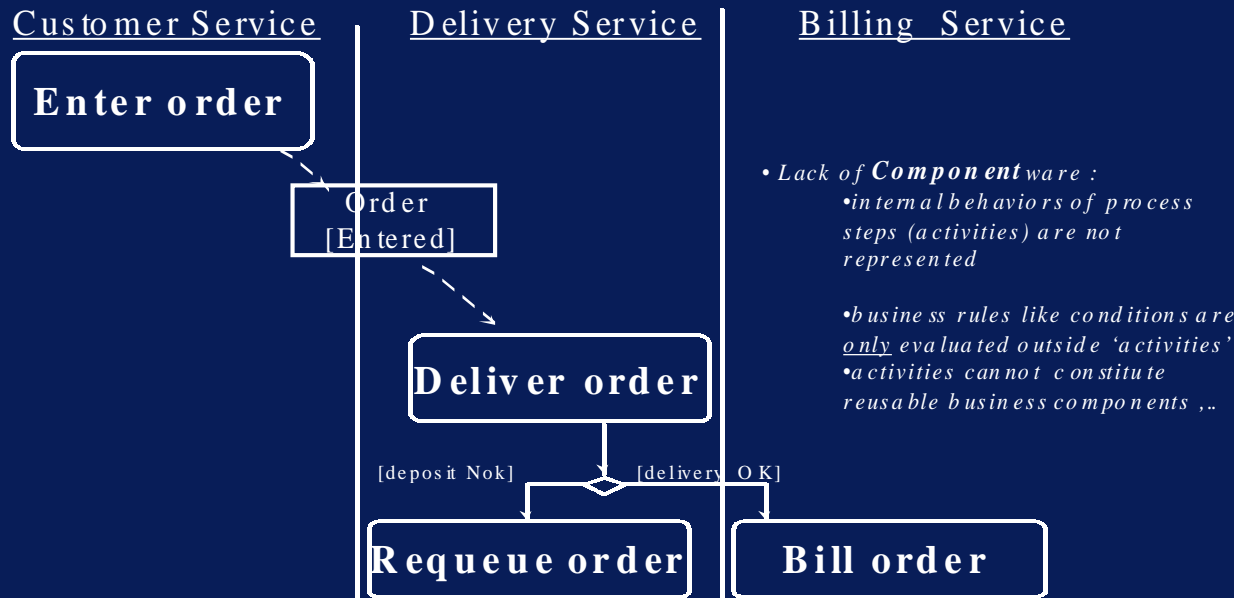
- Requirements (in Enterprise and System levels) do need to be placed into “formal structures” in order to be gathered coherently and traced toward software components



Specifying requirements with the UML 's Activity Diagram : issues

- Emerging goals inside an activity (sub-activities) are not represented
 - No response to ‘ how an activity is performed ’
- Evolutivity for Requirements : N.A. via activity states
- Bridging towards Use Cases / Software : No traceability !

Modeling the Business using the UML's Activity Diagram



- Process steps (Activities) are not ‘Goal Oriented’ !

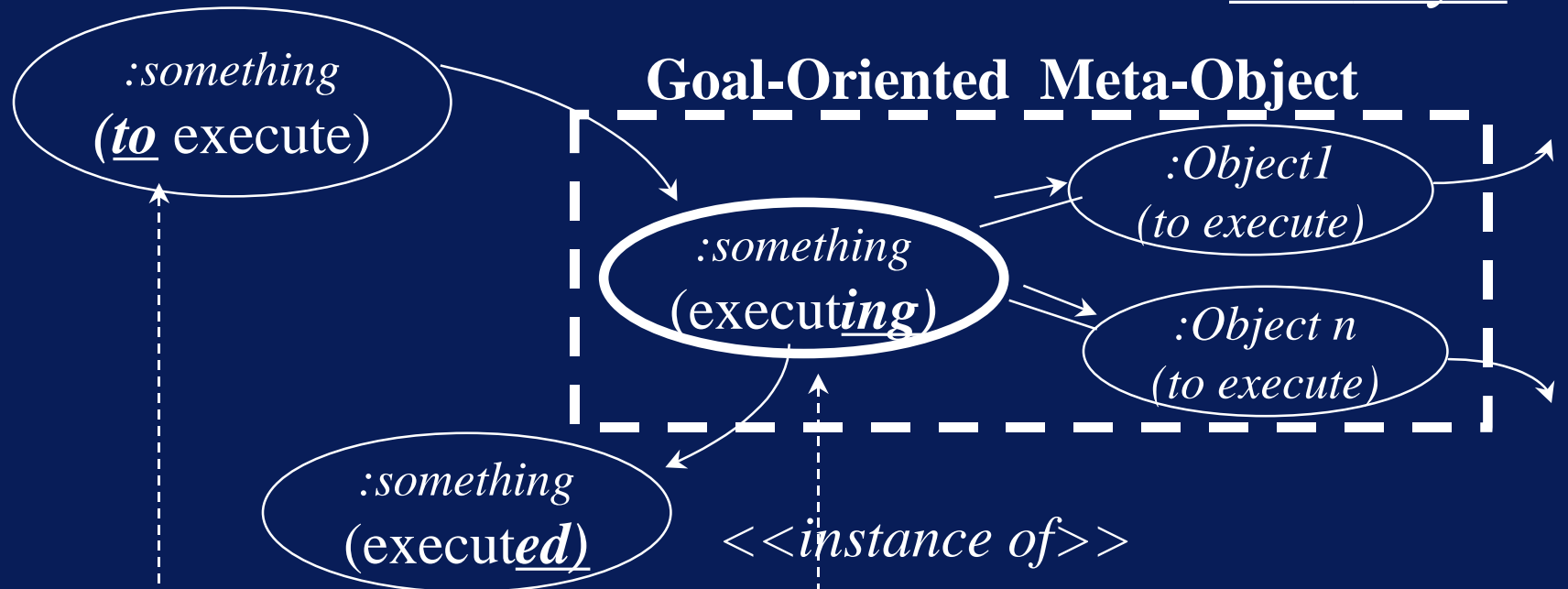
Need for “Goal Orientation” to Gather & Trace Requirements

- Activity Diagrams do not provide boundaries for ‘requirement components’ that encapsulate goals and related sequences inside a requirement
 - They do not provide a “formal structure” for gathering requirements
 - Tracing & testing requirements as components become difficult even with well-structured use cases
- Use cases and Interaction diagrams are better structured when they are aligned on these ‘requirement boundaries’.
- Activity/ Action States need to represent subsequent goals to reach for every ‘ requirement component ’

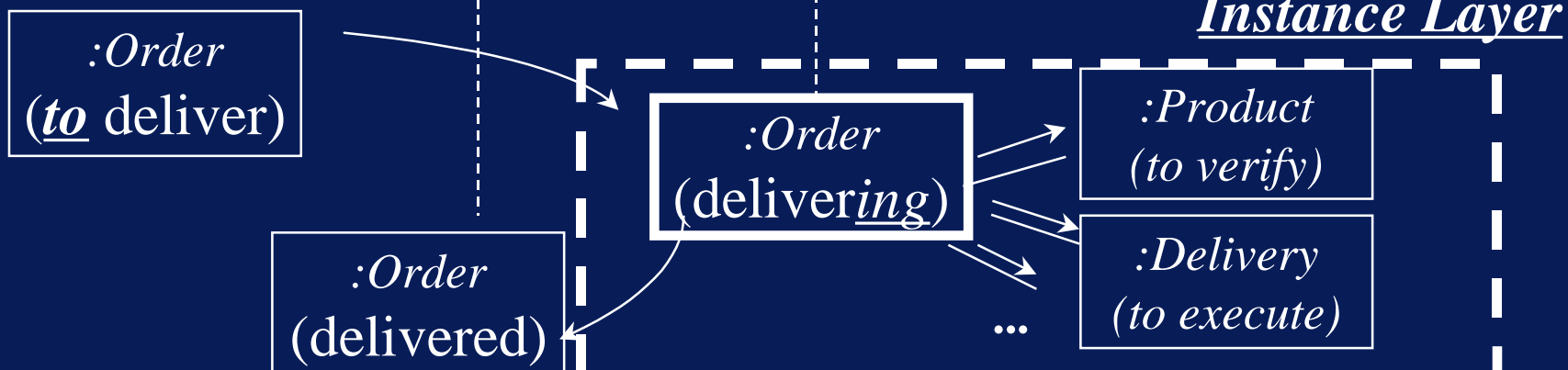
A Goal-Oriented Framework to specify 'Requirements'

A requirement is 'Something to execute' that needs to specify how functionally its goal may be reached

Meta Layer

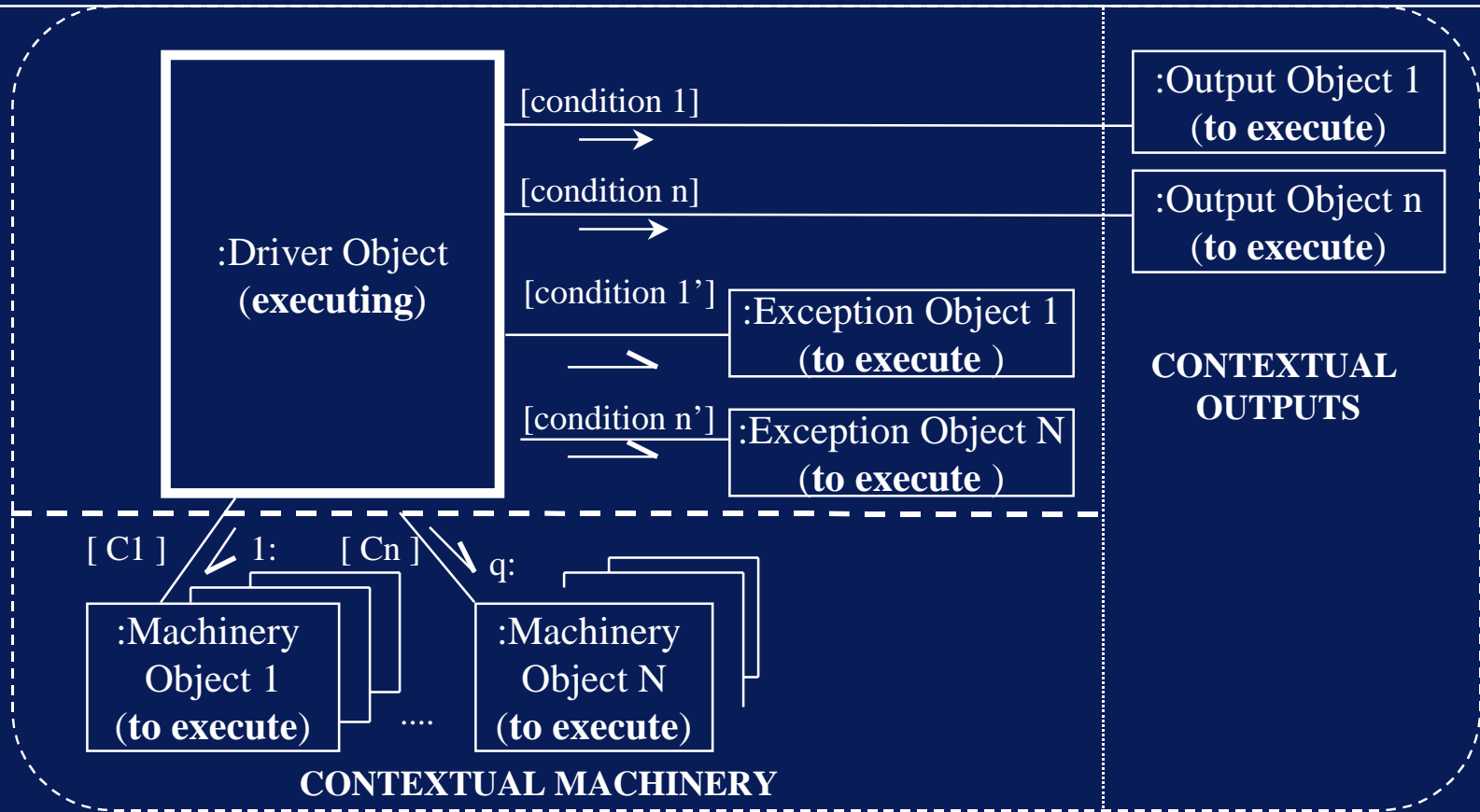


Instance Layer



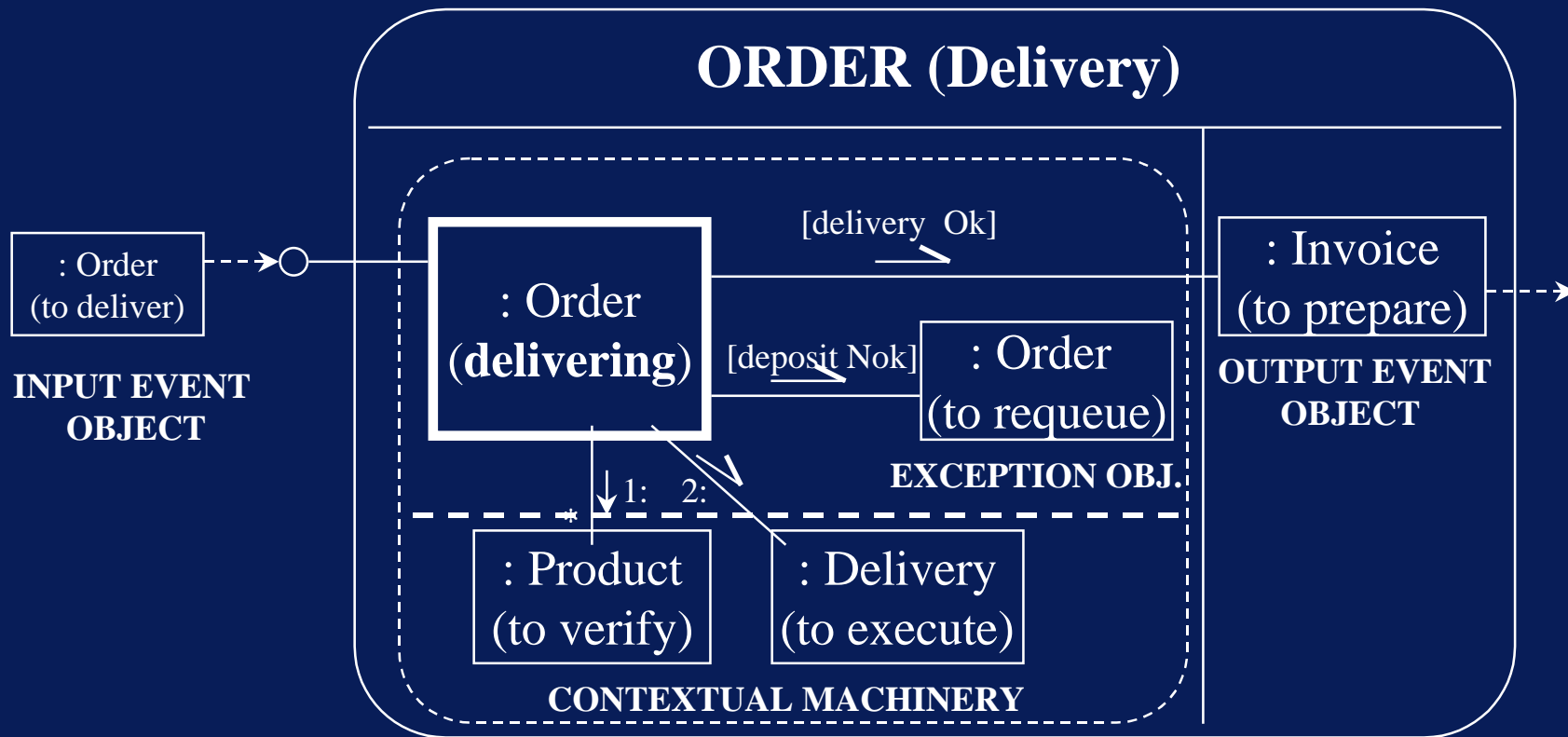
Goal Oriented Object (GOO) : Task oriented set of objects that express their contextual behavior using ‘behavioral states’

GOAL ORIENTED OBJECT (Execution)



- Machinery, Exception and Output Objects are ‘event typed’ objects.
- Driver object is a function typed object

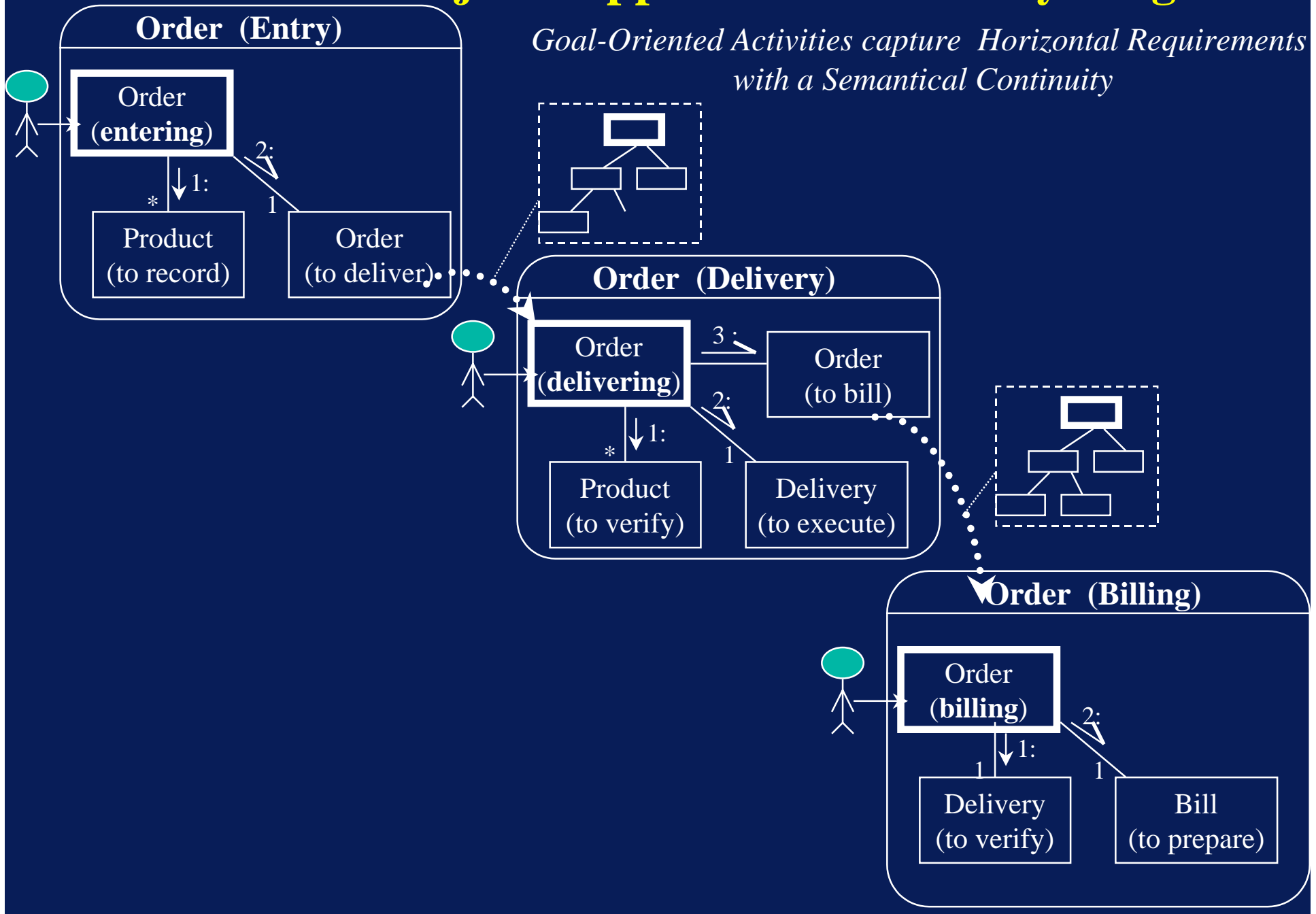
Order (Delivery) : A Goal-Oriented Object (GOO)



- Order (delivering) owns the necessary know-how to realize its goal.
- It controls the activity by sending messages that are executed by its contextual machinery.
- Objects of the Contextual Machinery carry their **responsibility** to realize the goal of the driver.

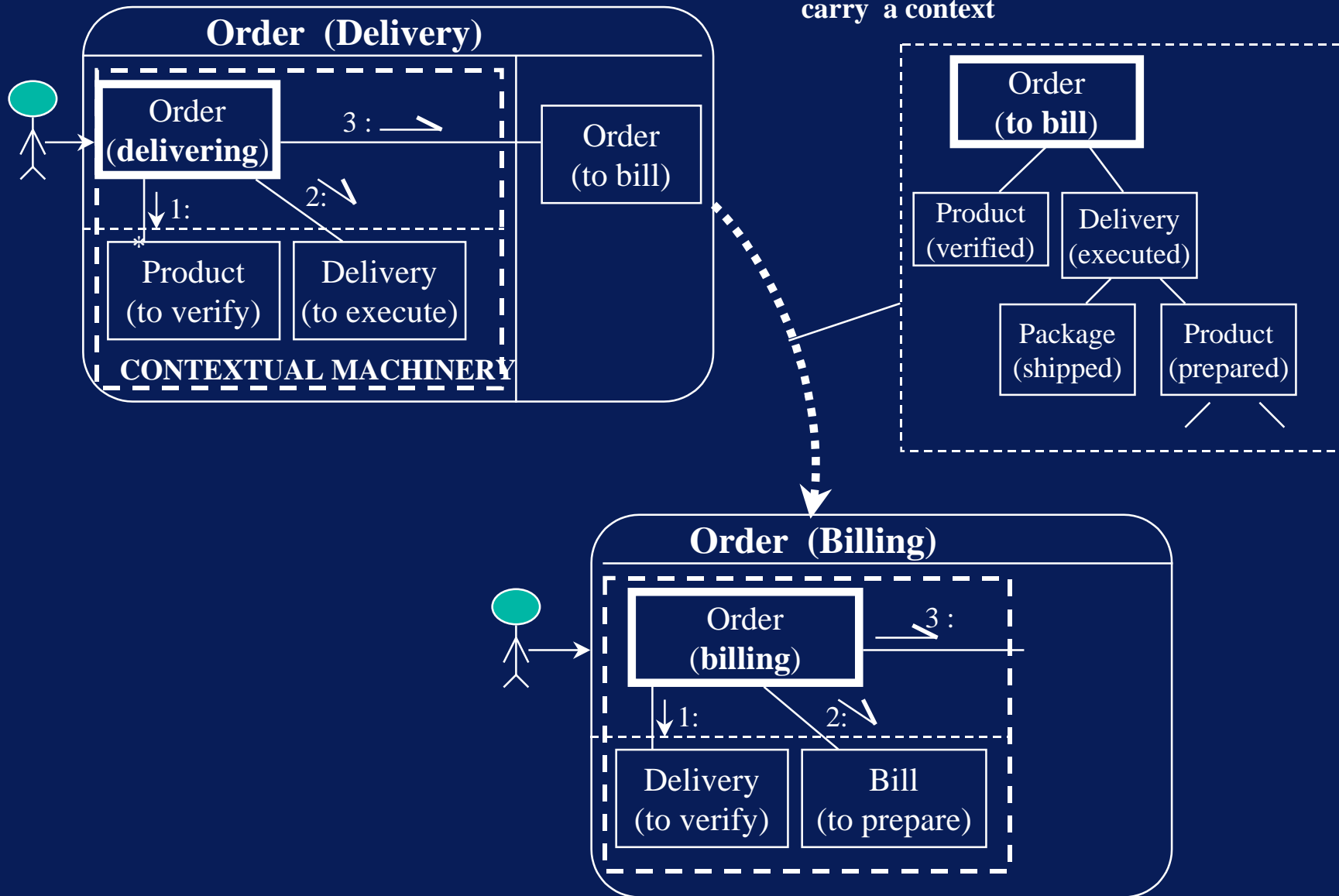
Goal-Oriented Objects applied to the Activity Diagram

Goal-Oriented Activities capture Horizontal Requirements with a Semantical Continuity

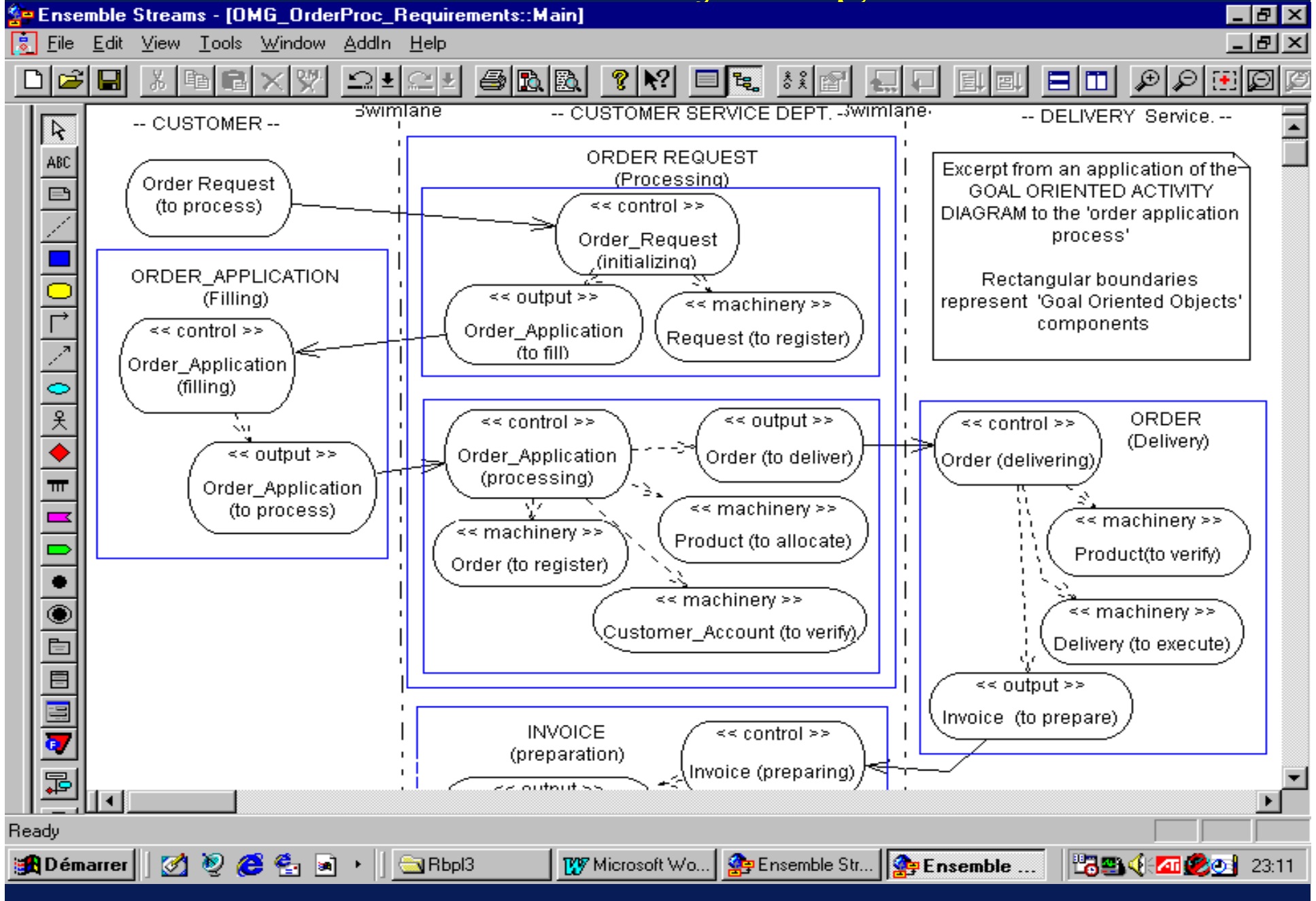


Flows carry Context between GOOs

Flows between two Independent or Nested GOOs carry a context



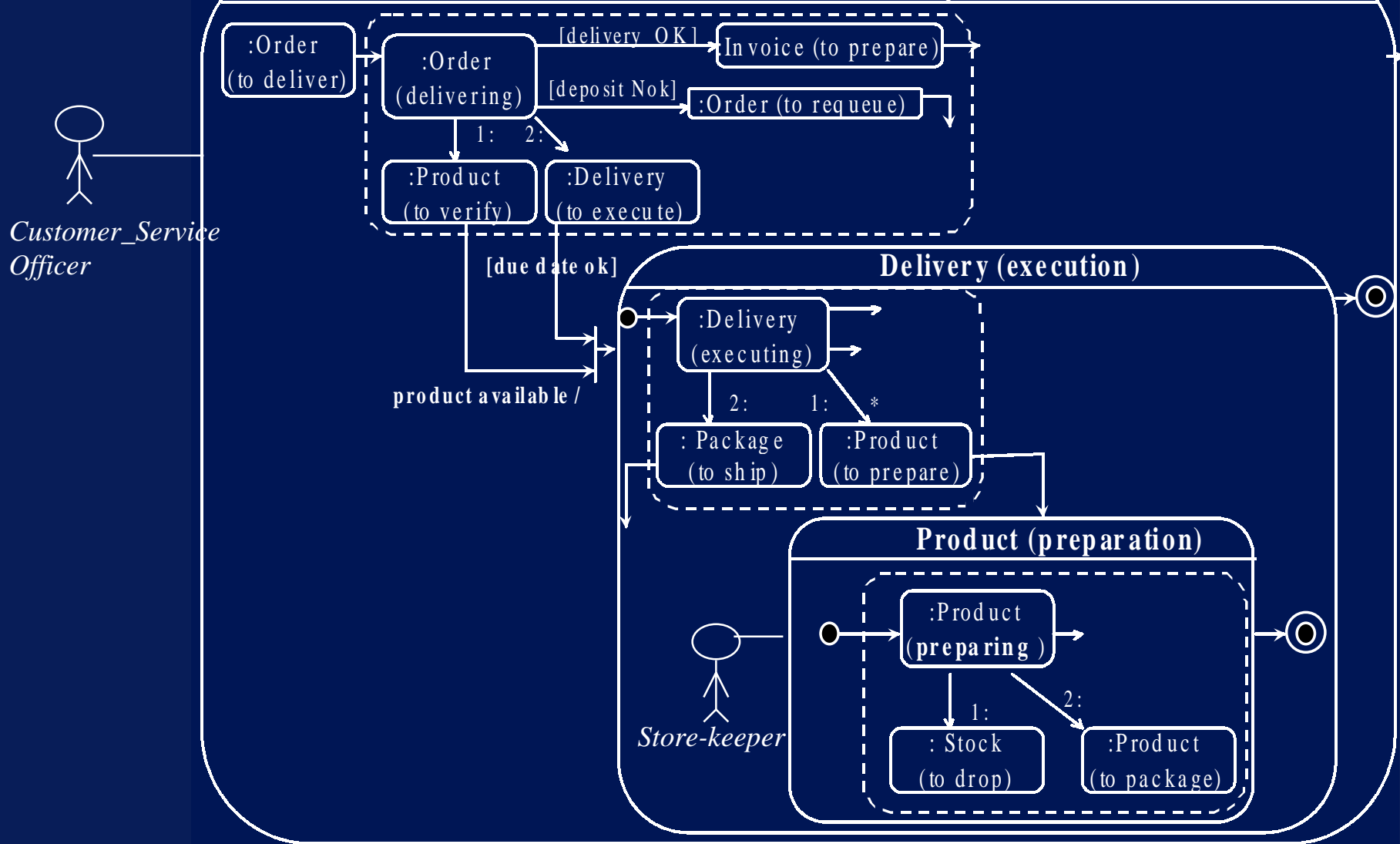
Goal Oriented Activity Diagram : details



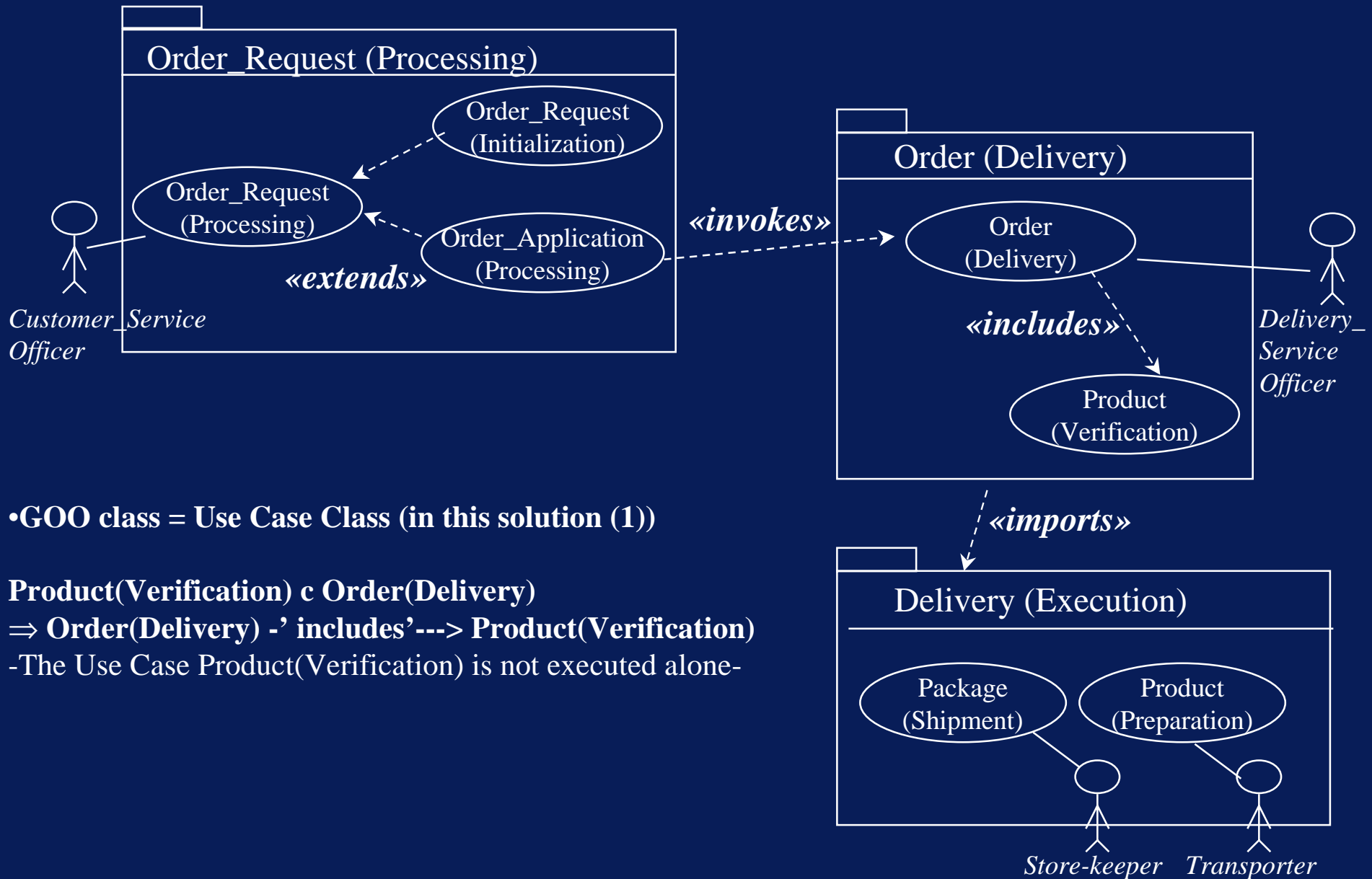
Detail of the GOO Component 'Order Delivery'

Behavioral State Transition Diagram (A Reusable and Executable Component)

ORDER (Delivery)



Business Process driven Use Cases and their Relationships



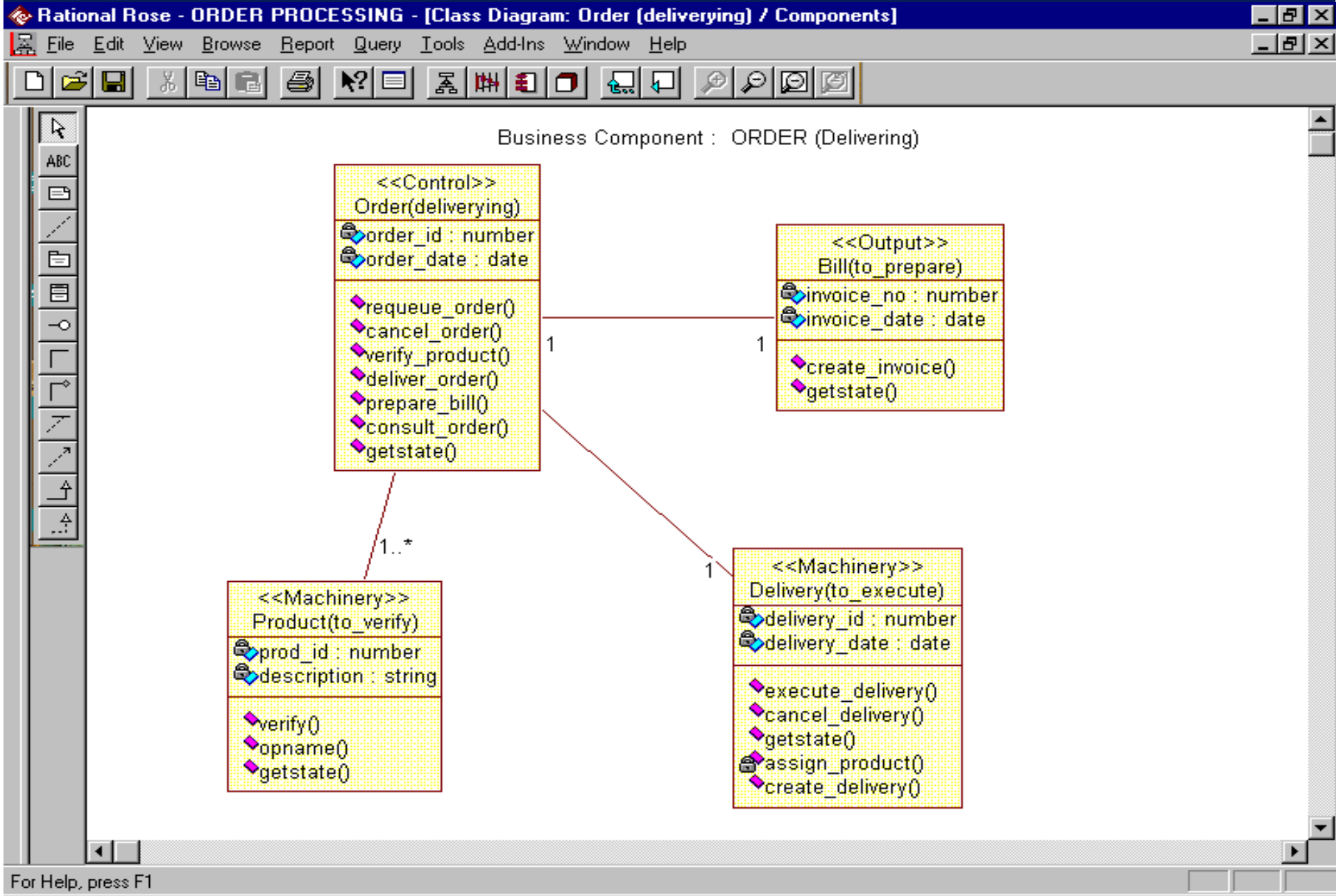
•GOO class = Use Case Class (in this solution (1))

Product(Verification) c Order(Delivery)

⇒ **Order(Delivery) -' includes' ---> Product(Verification)**

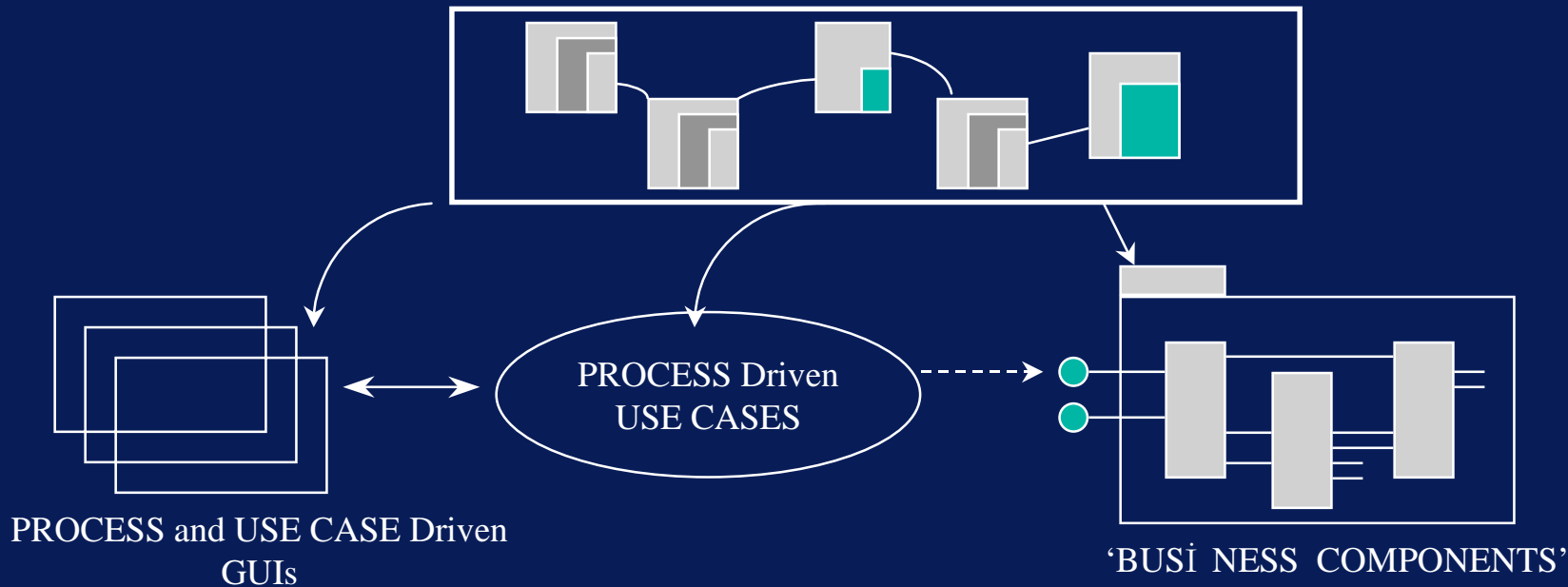
-The Use Case Product(Verification) is not executed alone-

Traceability towards Software Components



Goal Oriented Objects in the Development Life Cycle

Goal Oriented Activity Diagram



- **allows an automatic specification of Use Cases** with their relationships (includes, extends, 'invokes')
- very easy to understand and to validate by the users, **prototypable**
- **Goal-Oriented Activity Diagram** can drive a Distributed ARCHITECTURE
- **Task Repartition and Iterations Facilitated** (No unfinished GOO classes by the end of iterations within 'Goal driven development cycles')
- Unit and Integration Tests based on the **GOO components and their interfaces**