Dependability and Performance Assessment of Dynamic CONNECTed Systems

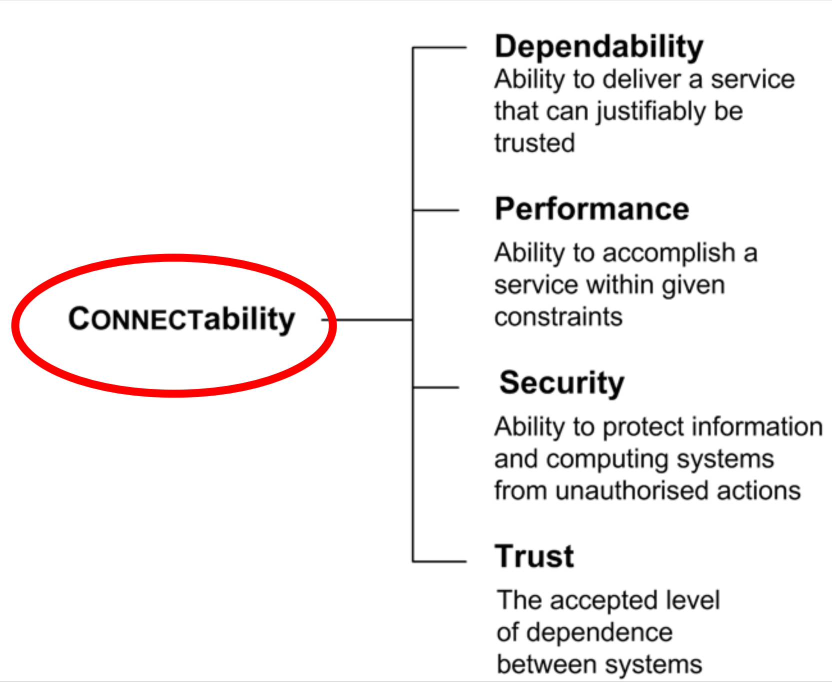
Antonia Bertolino, Felicita Di Giandomenico
ISTI-CNR
Joint work with
A. Calabro', F. Lonetti, M. Martinucci, P. Masci, N. Nostro, A. Sabetta

Outline

- V&V in CONNECT
- Introduction to Dependability and Performance
- Introduction to Monitoring
- Dependability and Performance Approaches in CONNECT
- Logical Architecture of DePer
- The GLIMPSE Monitoring Infrastructure
- GLIMPSE + DePer
- Case Study
- Demo

Today's Lecture

addresses the non-functional attributes of CONNECTed systems at synthesis time and at runtime



CONNECTability

Dependability
Ability to deliver a service that can justifiably be trusted

Performance
Ability to accomplish a service within given constraints

Security
Ability to protect information and computing systems from unauthorised actions

Trust
The accepted level of dependence between systems

- **On-line & Off-line V&V support**
  - Generic architecture for dependability analysis and verification
  - Interacts with monitor for runtime analyses
- **Security & Trust**
  - SxCxT paradigm
  - Interoperable trust management
- **Modeling NF properties**
  - Meta-model for CONNECT properties

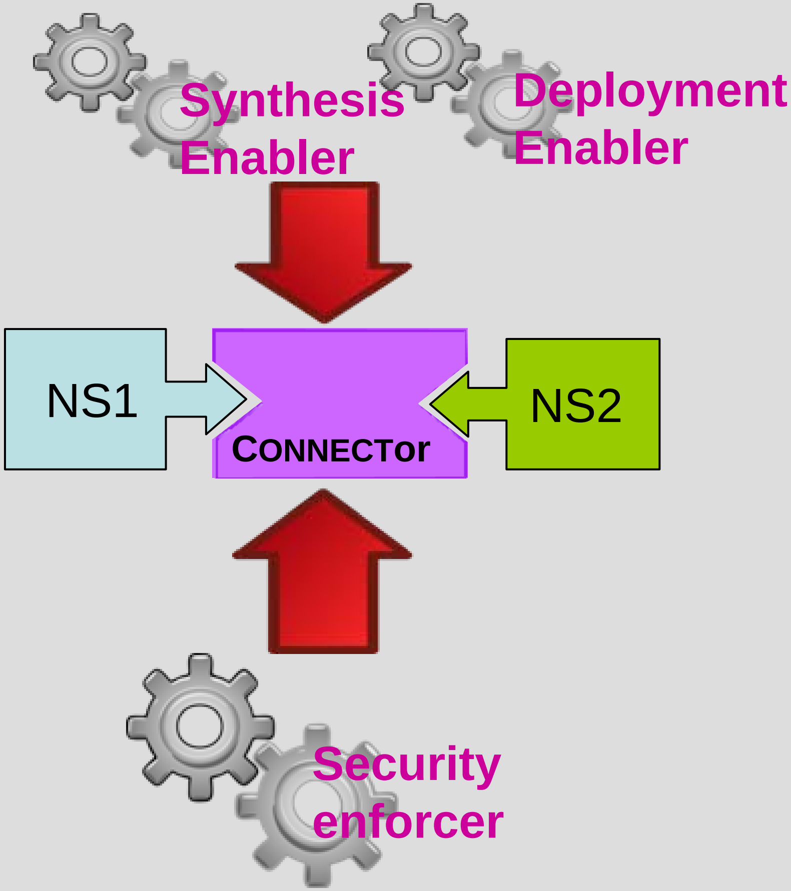
CONNECT Vision and V&V

- The very goal of CONNECT, ensuring interoperability in spite of changes, requires special attention on validation techniques
  - to ensure that the functionality of systems is as expected
  - to ensure that the desired non-functional properties are maintained
- An ambitious goal: achieving CONNECTability even in a highly dynamic setting

Challenges

- System assembled dynamically
- Reference specification of expected/correct operation not a-priori available
- Specifications are learnt/inferred, thus they can be incomplete, unstable, uncertain
- Assessment activities must accommodate change (and must be adaptable themselves)
- Special emphasis on run-time assessment (possibly coupled with off-line analysis techniques, whenever possible)

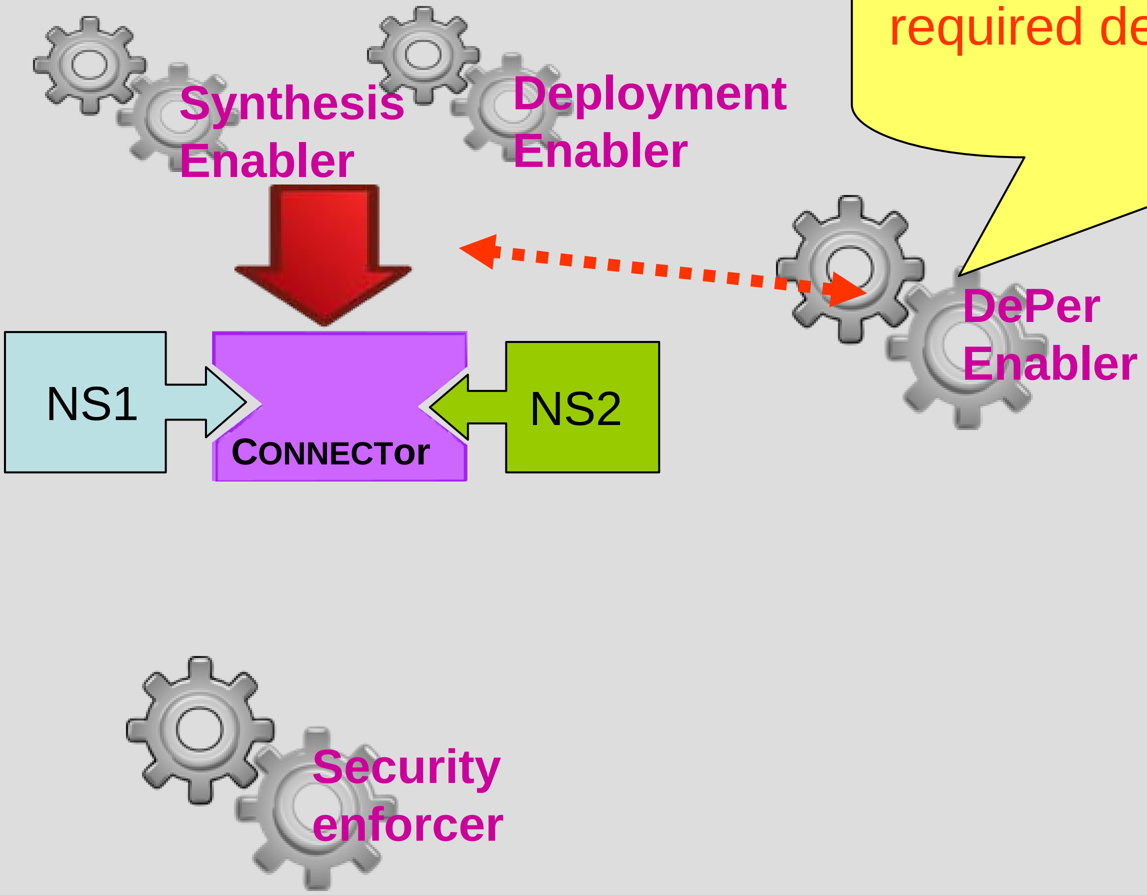
Overview of CONNECTability Assurance



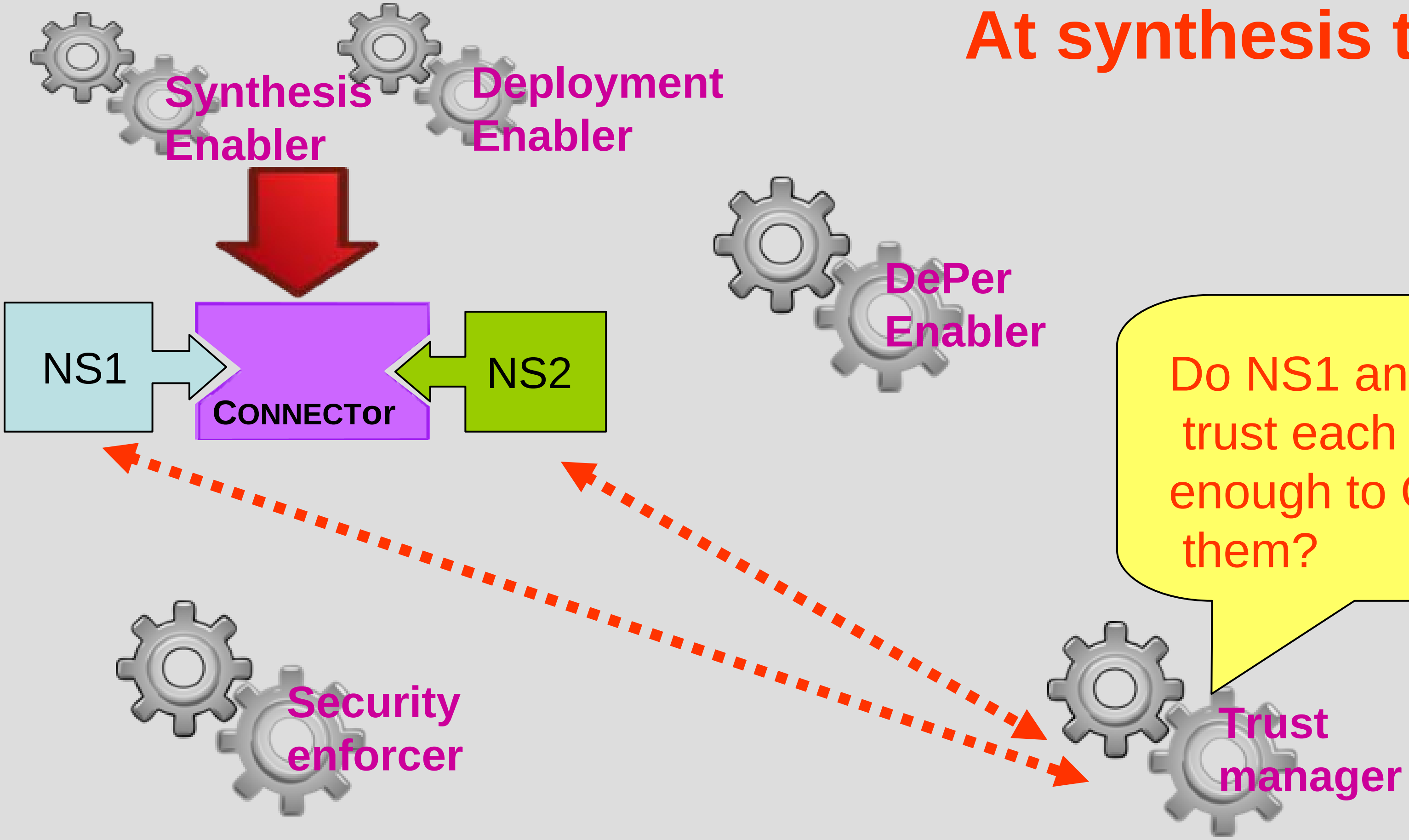
At synthesis time:

Overview of CONN

Will the CONNECTed system composed by NS1+CONNECTor+NS2 satisfy the required dep.&perf. properties ?

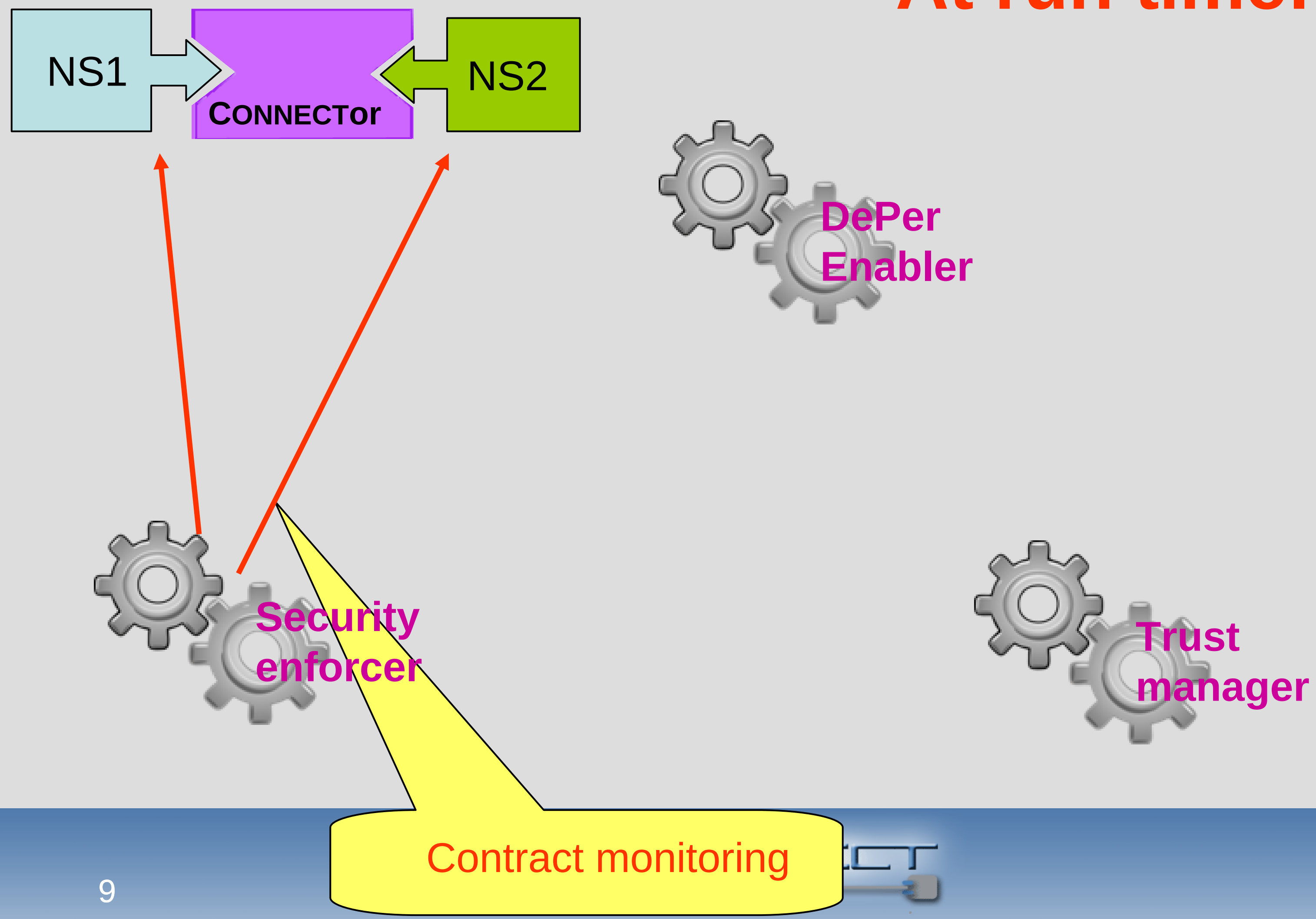


Overview of CONNECTability Assurance



Overview of CONNECTability Assurance

At run time:



NS1

CONNECTor

NS2

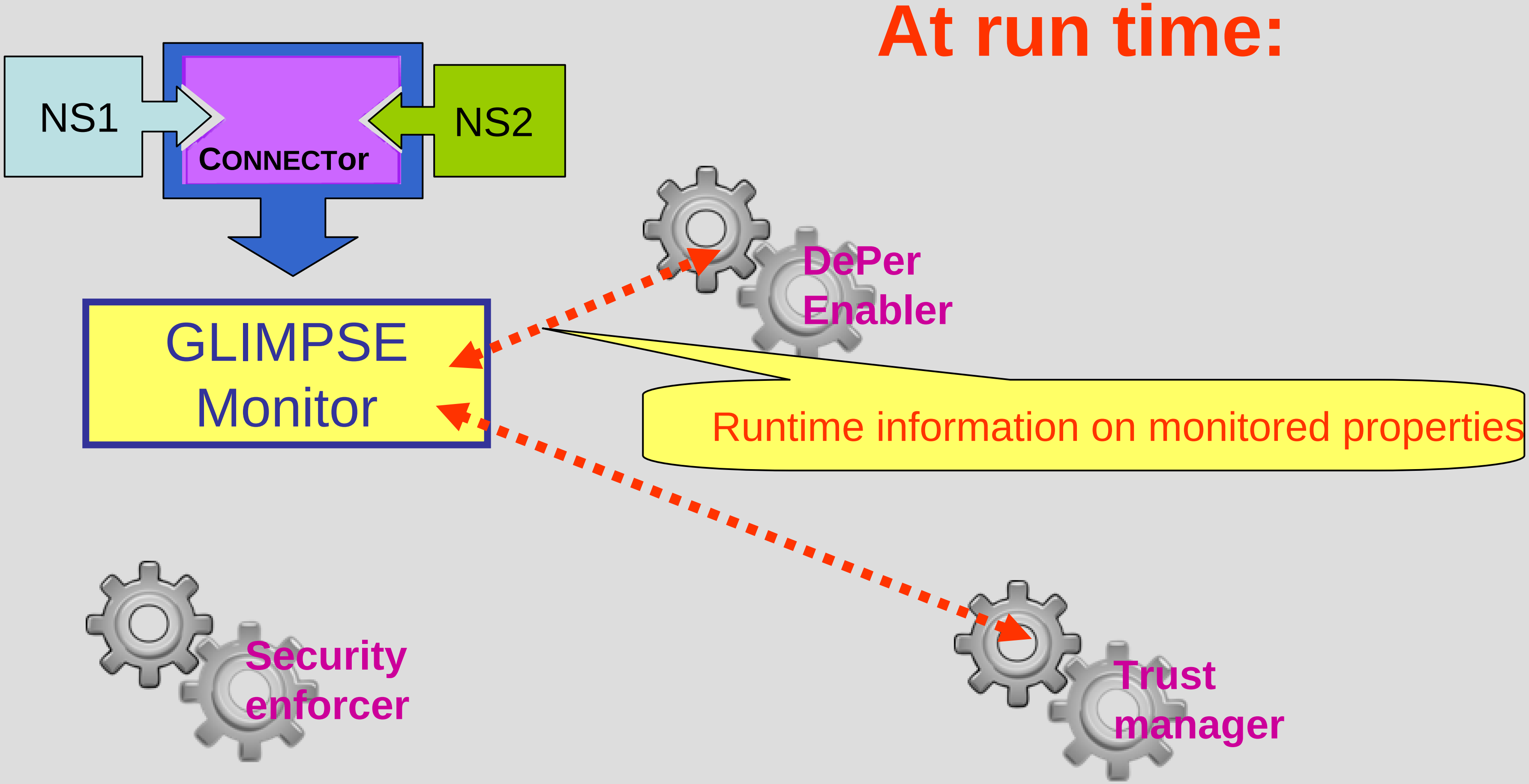
DePer Enabler

Security enforcer

Trust manager

Contract monitoring

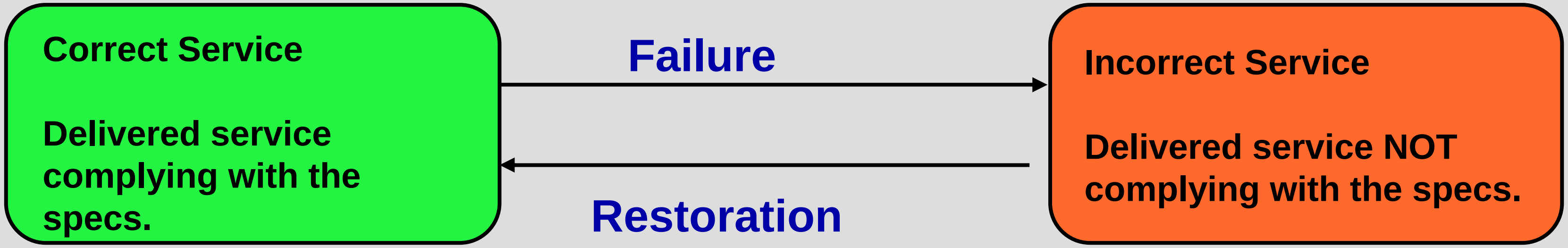
Overview of CONNECTability Assurance



Introduction to Dependability and Performance attributes

Dependability

- **Dependability** is the ability of a system to provide a service that can justifiably be trusted
- System service is classified as ***proper*** if it is delivered as specified; otherwise it is ***improper***.
  - System ***failure*** is a transition from proper to improper service.
  - System ***restoration*** is a transition from improper to proper service.



Correct Service

Delivered service complying with the specs.

Failure

Restoration

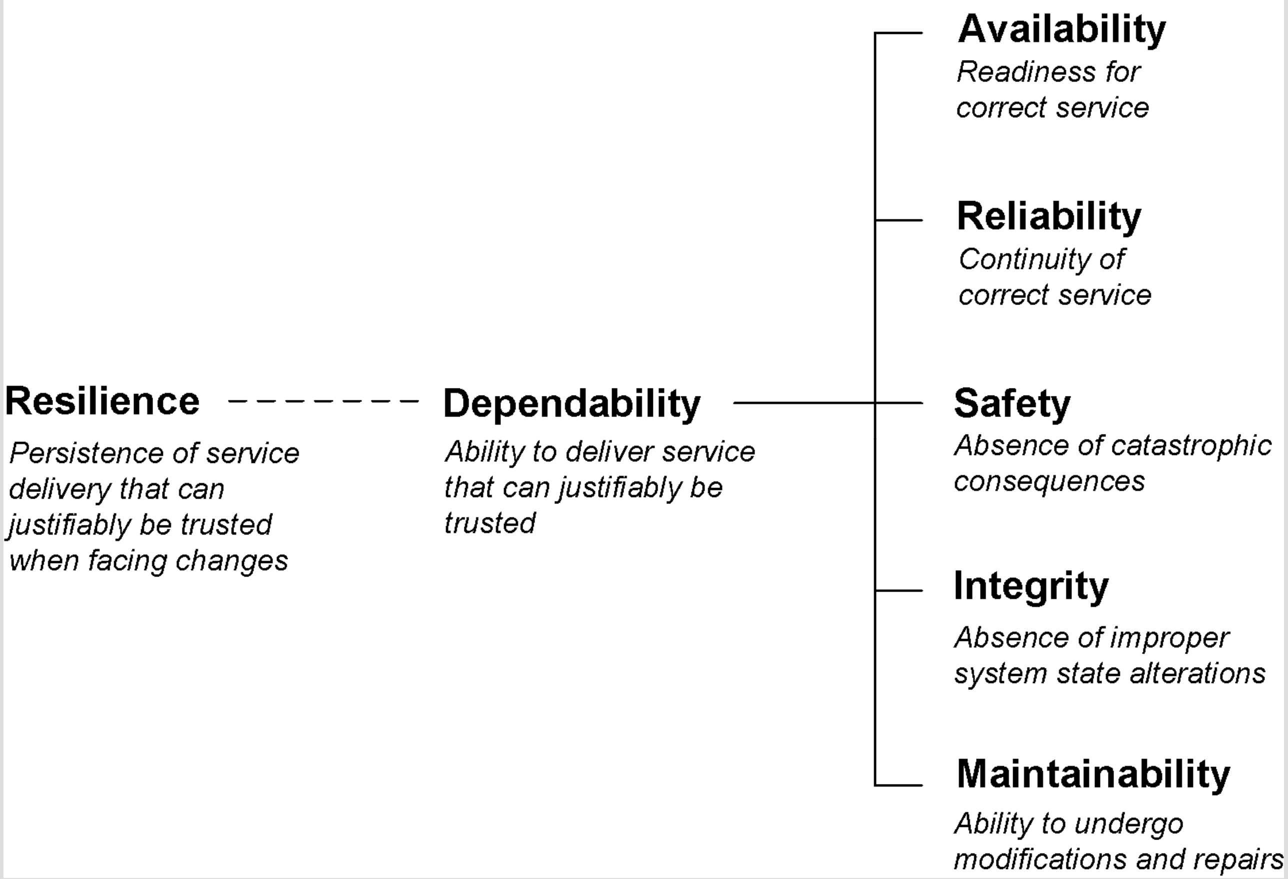
Incorrect Service

Delivered service NOT complying with the specs.

The "***properness***" of service depends on the user's viewpoint!

[J.C. Laprie (ed.), *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992].

Dependability attributes



Resilience

Persistence of service delivery that can justifiably be trusted when facing changes

Dependability

Ability to deliver service that can justifiably be trusted

Availability

Readiness for correct service

Reliability

Continuity of correct service

Safety

Absence of catastrophic consequences

Integrity

Absence of improper system state alterations

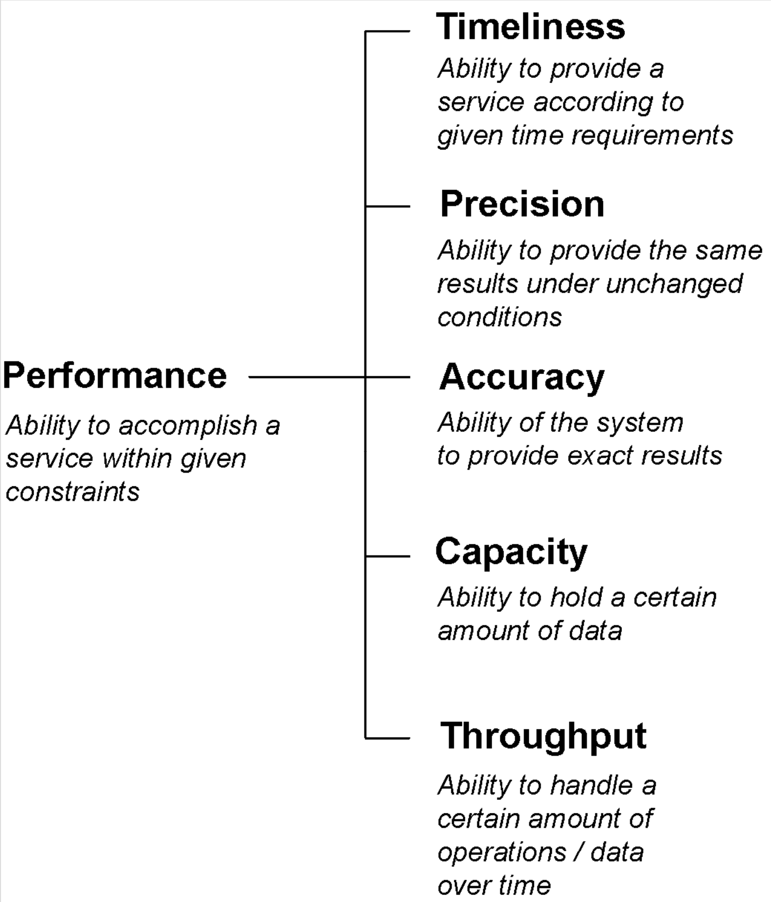
Maintainability

Ability to undergo modifications and repairs

In general, a number of Metrics can be defined for a given attribute, e.g.:

- **A(t)** at instant of time t
- **E[A(t)]** expected value
- **A(0,t)** in the [0,t] time interval

Performance attributes



Performance

Ability to accomplish a service within given constraints

Timeliness

Ability to provide a service according to given time requirements

Precision

Ability to provide the same results under unchanged conditions

Accuracy

Ability of the system to provide exact results

Capacity

Ability to hold a certain amount of data

Throughput

Ability to handle a certain amount of operations / data over time

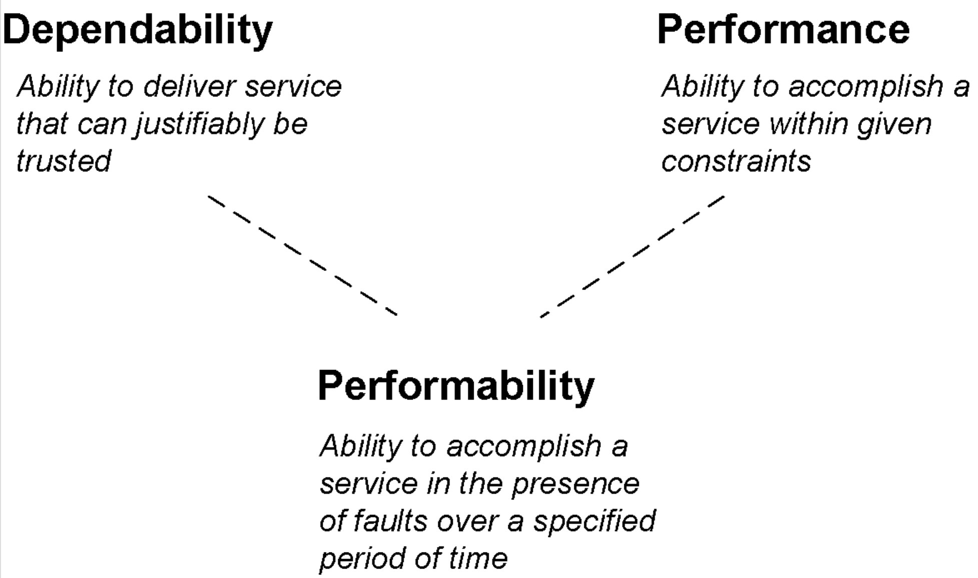
Performance is how well a system performs, provided that service is proper

Performance metrics typically include:

- # of jobs per time unit (throughput)
- time to process a job (response time)
- max # of jobs per time unit (capacity)

[IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology, 1990]

and Performability



Dependability

Ability to deliver service that can justifiably be trusted

Performance

Ability to accomplish a service within given constraints

Performability

Ability to accomplish a service in the presence of faults over a specified period of time

Typical evaluation measure for **degradable systems**, i.e. highly dependable systems which can undergo a graceful degradation of performance in the presence of faults (malfunctions) allowing continued "normal" operation.

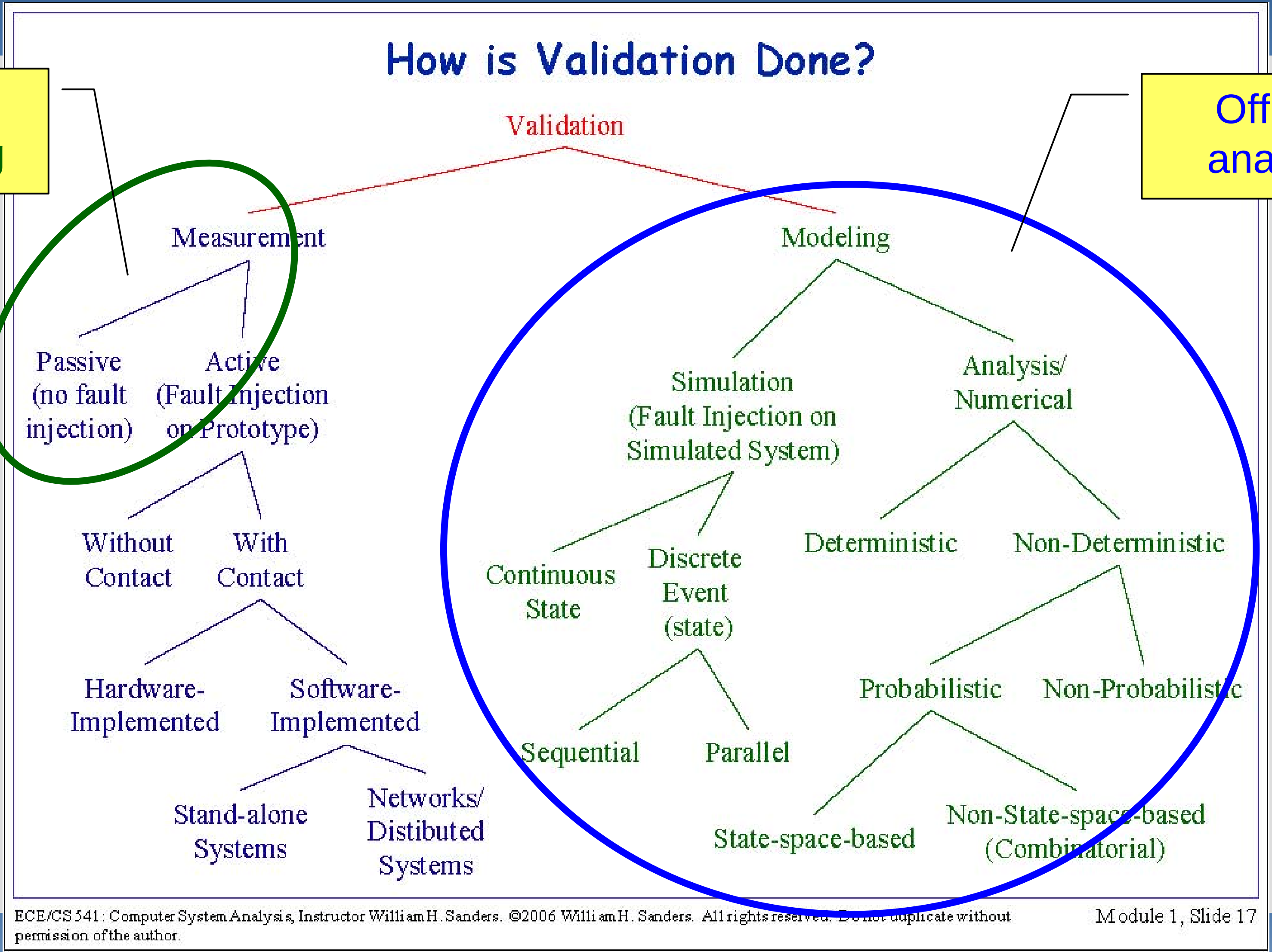
Examples of performability metrics:

- Work the system can be expected to accomplish before a failure
- Probability that the system operates above a certain level of efficiency during an observation period

Validation Methods

Runtime monitoring

Off-line analysis



How is Validation Done?

Validation

- Measurement
  - Passive (no fault injection)
  - Active (Fault Injection on Prototype)
    - Without Contact
    - With Contact
      - Hardware-Implemented
      - Software-Implemented
        - Stand-alone Systems
        - Networks/ Distibuted Systems
- Modeling
  - Simulation (Fault Injection on Simulated System)
    - Continuous State
    - Discrete Event (state)
      - Sequential
      - Parallel
  - Analysis/ Numerical
    - Deterministic
    - Non-Deterministic
      - Probabilistic
        - State-space-based
        - Non-State-space-based (Combinatorial)
      - Non-Probabilistic

Module 1, Slide 17

Stochastic Model-Based Approaches

Consist of 2 phases:

- The construction of a model of the system from the elementary stochastic processes that model the behavior of the components of the system and their interactions; these elementary stochastic processes mainly relate to failure, to service restoration and repair;
- Processing the model to obtain the expressions and the values of the dependability measures of the system.

Solution Methods

Dependability Model Solution Methods -- Method by which one determines measures from a model. Models can be solved by a variety of techniques:

Combinatorial Methods -- Structure of the model is used to obtain a simple arithmetic solution.

Analytical/Numerical Methods -- A system of linear differential equations or linear equations is constructed, which is solved to obtain the desired measures

Simulation -- The description of what the system is and does is executed, and estimates of the measures are calculated based on the resulting executions (known also as *sample paths* or *trajectories*.)

When does Validation take place?

In all the stages of the system development process:

- Specification - Combinatorial modeling, Analytic/Numerical modeling
- Design - Analytic/Numerical modeling, Simulation modeling
- Implementation - Detailed Simulation modeling, Measurement, including Fault Injection
- Operation - Combinatorial modeling, Analytic/Numerical modeling, Detailed Simulation modeling, Measurement, including runtime monitoring

Choosing Validation Techniques

- There are several choices, each with differing advantages and disadvantages

Choice of a validation method depends on:

- Stage of design (is it a proposed or existing system?)
- Time (how long until results are required)
- Tools available
- Accuracy
- Ability to compare alternatives
- Cost
- Scalability

Review of Stochastic Model-Based Methods

Variety of models, each focusing on particular levels of abstraction and/or system characteristics.

- **Combinatorial Methods**
  - Reliability Block Diagrams
  - Fault Trees
- **Model-checking**
- **State-space stochastic methods**

[David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: from dependability to security. IEEE TDSC, 1:48-65, January-March 2004.]

[A. Bondavalli, S. Chiaradonna, and F. Di Giandomenico. Model-based evaluation as a support to the design of dependable systems. In Diab and Zomaya, editors, Dependable Computing Systems: Paradigms, Performance Issues, and Applications, 57-86. Wiley,2005.]

Introduction to
Run-time Analysis via Monitoring

Validation @ runtime

- Relies on sensing what is happening and on timely collecting relevant information

→ We need to monitor systems behaviour

An over-loaded term

- Large (but fractioned) body of research, carried out over decades.
- Different authors use the term “monitoring" to indicate different things.
- A monitoring system is in fact an assembly of different pieces dealing with different concerns.

Monitoring: Definition

- the process of dynamic collection, interpretation, and presentation of information concerning objects or software processes under scrutiny

[J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. ACM Trans. Comput. Syst., 5(2):121–150, 1987]

Monitoring: purpose

- A monitor gathers information about a process as it executes
- This is always carried out with a purpose in mind
- The specialization of monitoring to the different purposes determines the type and the way in which information is collected

Monitoring: purpose

- Some uses:
  - Dependability
  - Performance evaluation
  - Security
  - Correctness checking
  - Debugging and testing
  - Control
  - Accounting
  - Resource utilisation analysis

Monitoring: purpose

- Some uses:

  - Dependability
  - Performance evaluation
  - Security
  - Correctness checking
  - Debugging and testing
  - Control
  - Accounting
  - Resource utilisation analysis

Example: Fault-monitoring

- A monitor takes a specification of desired software properties and observes an executing software system to check that the execution meets the properties, i.e., that the properties hold for the given execution.
- See e.g. Delgado et al.’s for a taxonomy

[N. Delgado, A. Quiroz Gates, and S. Roach. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools. IEEE TSE. 30(12) 2004, 859-872.]

“On-line” monitoring

- By default.
- Schroeder qualifies on-line as:
  - External observation
  - Monitored application is fully functioning
  - Intended to be permanent

[B. A. Schroeder. On-Line Monitoring: A Tutorial. Computer, 28(6):72-78, 1995]

Monitor types

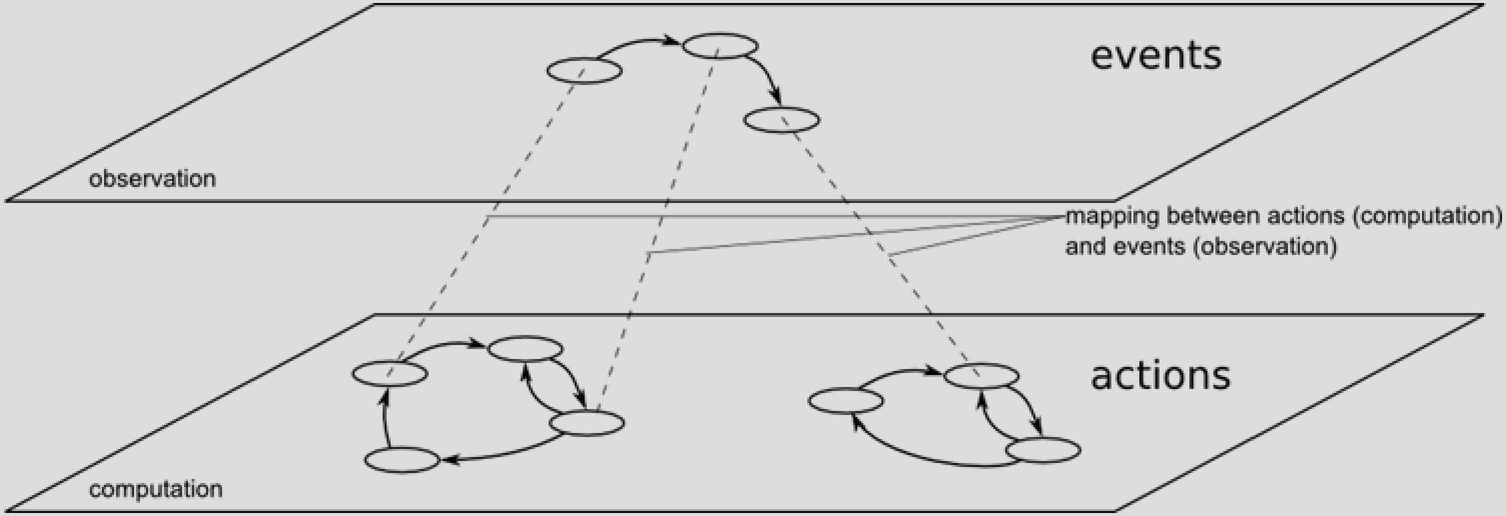
- Assertion based
- Property specification based
- Aspect-oriented programming
- Interception of exchanged messages
- Functional/Non-functional monitoring
- Data-driven vs. Event-driven

System observation

- The operation of a subject system is abstracted in terms of actions: we distinguish between actions which happen internally to components and those at the interfaces between components
- Communication actions are regulated by inter-component communication protocols that are independent of the components internals.

Event-based monitoring

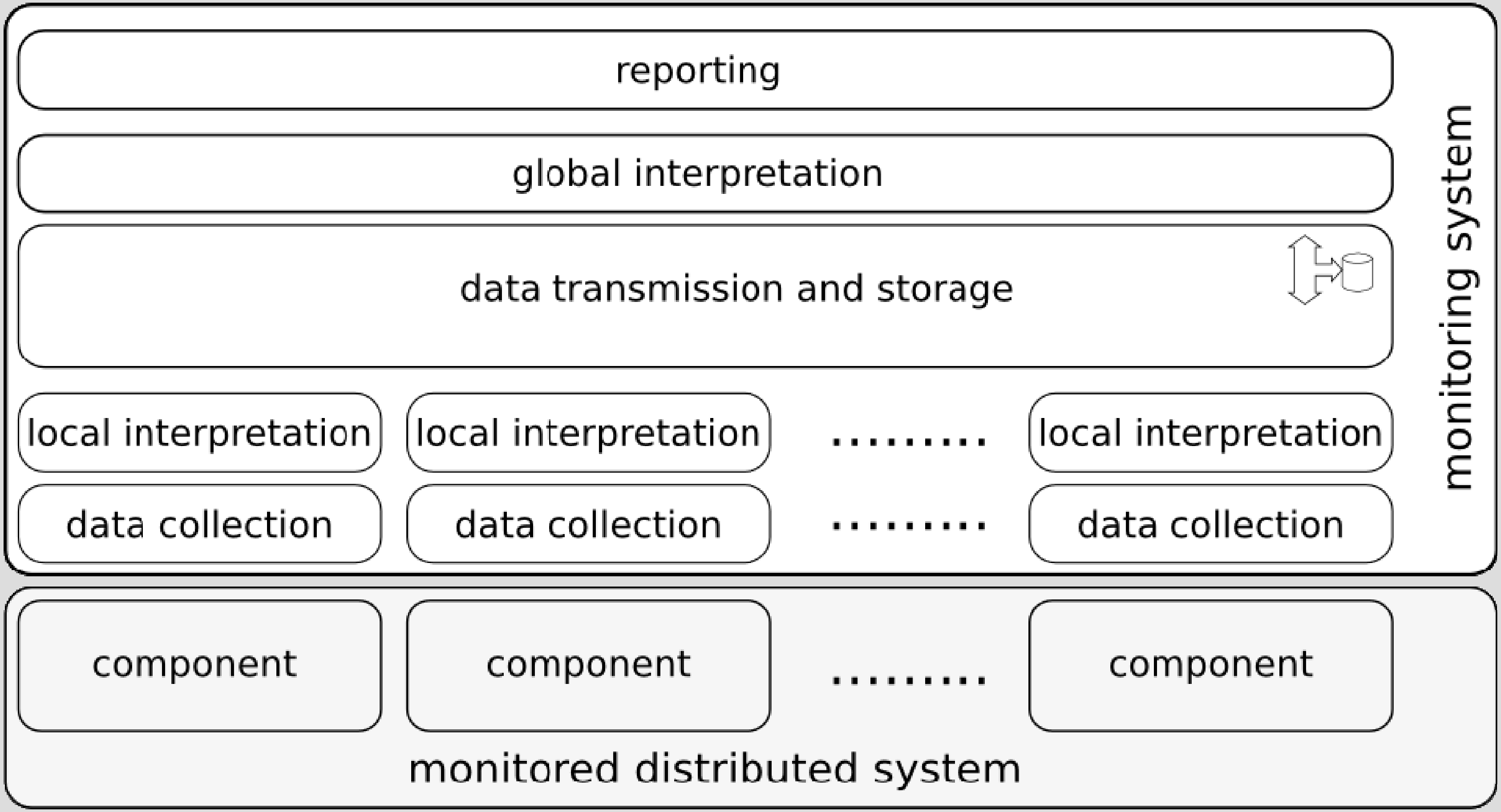
- In principle, a primitive event can be associated to the execution of each action; in practice, there is a distinction between the very subject of the observations (actions) and the way they are manifested for the purposes of the observation (events):
  - we have no means to observe actions but through the events that are associated to them


events
observation
mapping between actions (computation) and events (observation)
actions
computation

Event-based monitoring

- While actions just happen, firing of events depends on the decisions taken as part of the configuration of the monitoring system.
- Event specification is central to the overall setup of a monitoring system
  - Simple (“basic” or “primitive”) events : events that correspond to the completion of an action
  - Complex (“structured” or “composite”) events: happen when a certain combination of basic events and/or other composite events happen

Generic Monitoring Framework



Data collection

- Styles
  - Code instrumentation (off-line)
  - Runtime instrumentation (e.g. bytecode instrumentation, aspect-orientation)
  - Proxy-based (agent snoops communications to intercept
  - relevant events)
- Level of detail, target of the observation (hw-level, OS-level, middleware-level, application-level)
- Continuous Vs. sample-based (sample in time/space)

Local interpretation

- making sense of collected data (filter out uninteresting information)

Transmission

- Compression (may exploit semantics)
- Immediate Vs. delayed
- Buffering, resource consumption trade-offs
- Width of observation window (affects overhead as well as detection effectiveness), prioritisation.
- Lossy Vs. non-lossy

Global interpretation

aka “correlation”

- Put together information coming from different (distributed) processes to make sense of it globally
- May involve correlating concurrent events at multiple nodes
- Multi-layer architectures to increase scalability

Reporting

- Observed events might not be amenable for immediate use by the observer
- Either machine readable, or textual reports, graphics, animations and so on.

Distribution issues

- Physical separation:
  - No single point of observation, system partial failure, delays or communication failures,
- Concurrency
- Heterogeneity
- Federation
  - Crossing federation boundaries, different authorities, agreed policies
- Scaling
- Evolution

[Y. Hoffner, “Monitoring in distributed systems”, ANSA project 1994]

Natural Constraints

- Observability Problem
  - L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System, *CACM 21*, 7 (July 1978), 558-565.
  - *C. Fidge*. Fundamentals of Distributed System Observation. In IEEE Software, Volume 13, pp. 77-83, 1996.
- Probe Effect
  - *J. Gait. A Probe Effect in Concurrent Programs. Softw., Pract. Exper., 16(3):225–233, 1986.*

Relevant issues

- How data are collected/filtered from the source
- How info is aggregated/synchronized
- How to instruct the monitor

Events aggregation

- open-source event processing engines
  - Drools Fusion[1]
  - Esper[2]
  - can be fully embedded in existing Java architectures

[1] Drools Fusion: Complex Event Processor. http://www.jboss.org/drools/drools-fusion.html

[2] Esper: Event Stream and Complex Event Processing for Java. http://www.espertech.com/products/esper.php.

Some event based monitoring framework proposals

- **HiFi**[1]
  - event filtering approach
  - specifically targeted at improving scalability and performance for large-scale distributed systems
  - minimizing the monitoring intrusiveness
- **event-based middleware**[2]
  - with complex event processing capabilities on distributed systems
  - publish/subscribe infrastructure

[1] E. A. Hussein *Et al*. "HiFi: A New Monitoring Architecture for Distributed Systems Management", ICDCS, 171-178, 1999.

[2] E. P.R. Pietzuch, B. Shand, and J. Bacon. "Composite event detection as a generic middleware extension", Network, IEEE, 18(1):44-55, 2004.

Complex event monitoring specification languages

- **GEM**[1]
  - rule-based language
- **TESLA**[2]
  - simple syntax and a semantics based on a first order temporal logic
- **Snoop**[3]
  - event-condition-action approach supporting temporal and composite events specification
  - it is especially developed for active databases

[1] Samani and Sloman. “GEM: a generalized event monitoring language for distributed systems”, Distributed Systems Engineering, 4(2):96-108, 1997.

[2] G. Cugola and A. Margara. "TESLA: a formally defined event specification language", DEBS, 50-61, 2010.

[3] S. Chakravarthy and D. Mishra. “Snoop: An expressive event specification language for active databases", Data & Knowledge Engineering, 14(1) 1-26, 1994.

Non-functional monitoring approaches

- QoS monitoring[1]
  - distributed monitoring proposal for guaranteeing Service Level Agreements (SLA) in the web services
- monitoring of performance
  - Nagios[2]: for IT systems management (network, OS, applications)
  - Ganglia[3]: for high-performance computing systems, focused on scalability in large clusters

[1] A. Sahai *Et al*. “Automated SLA Monitoring for Web Services”, DSOM, 28-41, 2002.

[2] W. Barth. “Nagios. System and Network Monitoring”, 2006.

[3] M. L. Massie *Et al*. “The Ganglia distributed monitoring system: design, implementation, and experience”, Parallel Computing, 30(7):817-840, 2004.

Dependability and Performance Approach in CONNECT

Challenges of Dependability and Performance analysis in dynamically CONNECTed systems

- to deal with evolution and dynamicity of the system under analysis
  - impossibility/difficulty to analyze beforehand all the possible communication scenarios (through off-line analysis)
  - higher chance of inaccurate/unknown model parameters

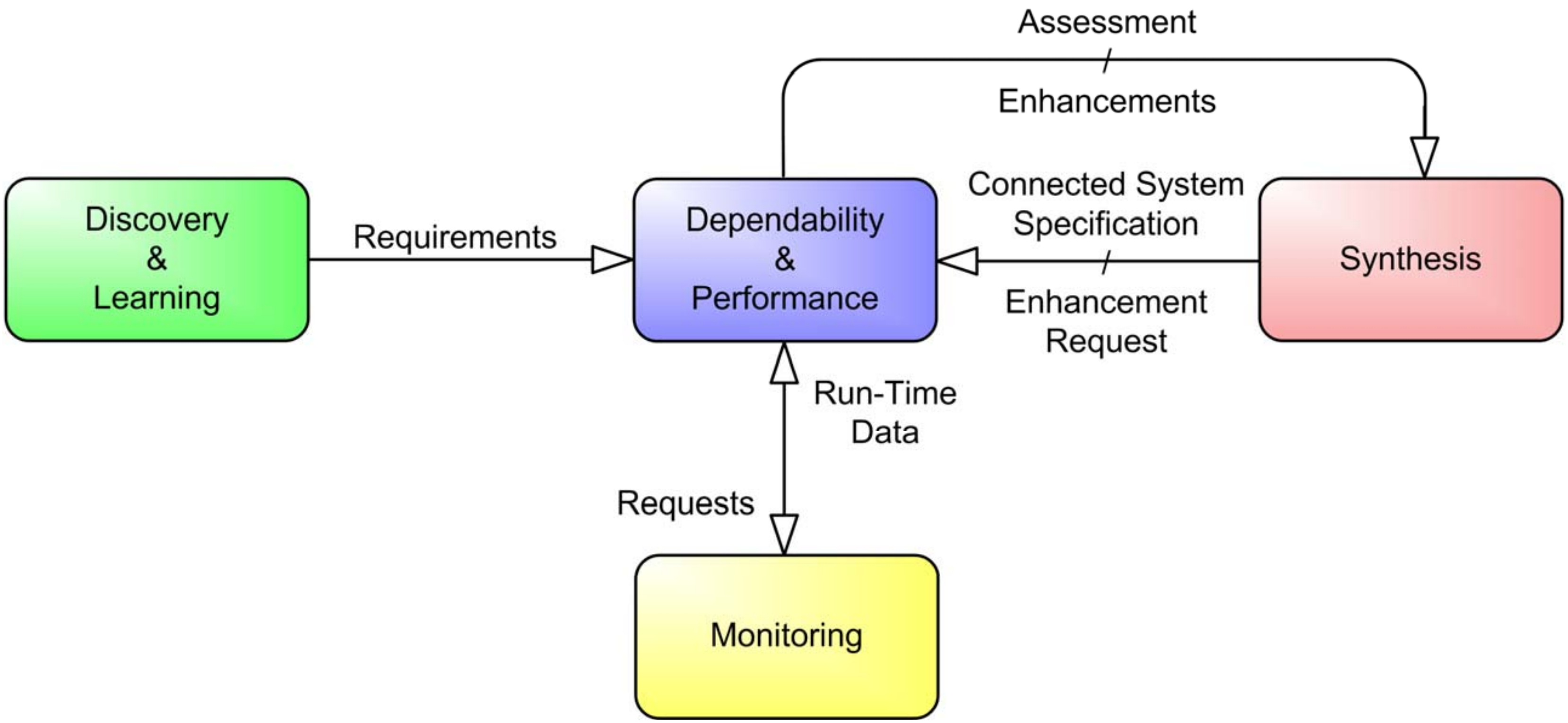
Approach in CONNECT:

➢ **off-line model-based analysis**, to support synthesis of quality connectors

➢ **refinement** step, based on real data gathered through on-line **monitoring** during executions

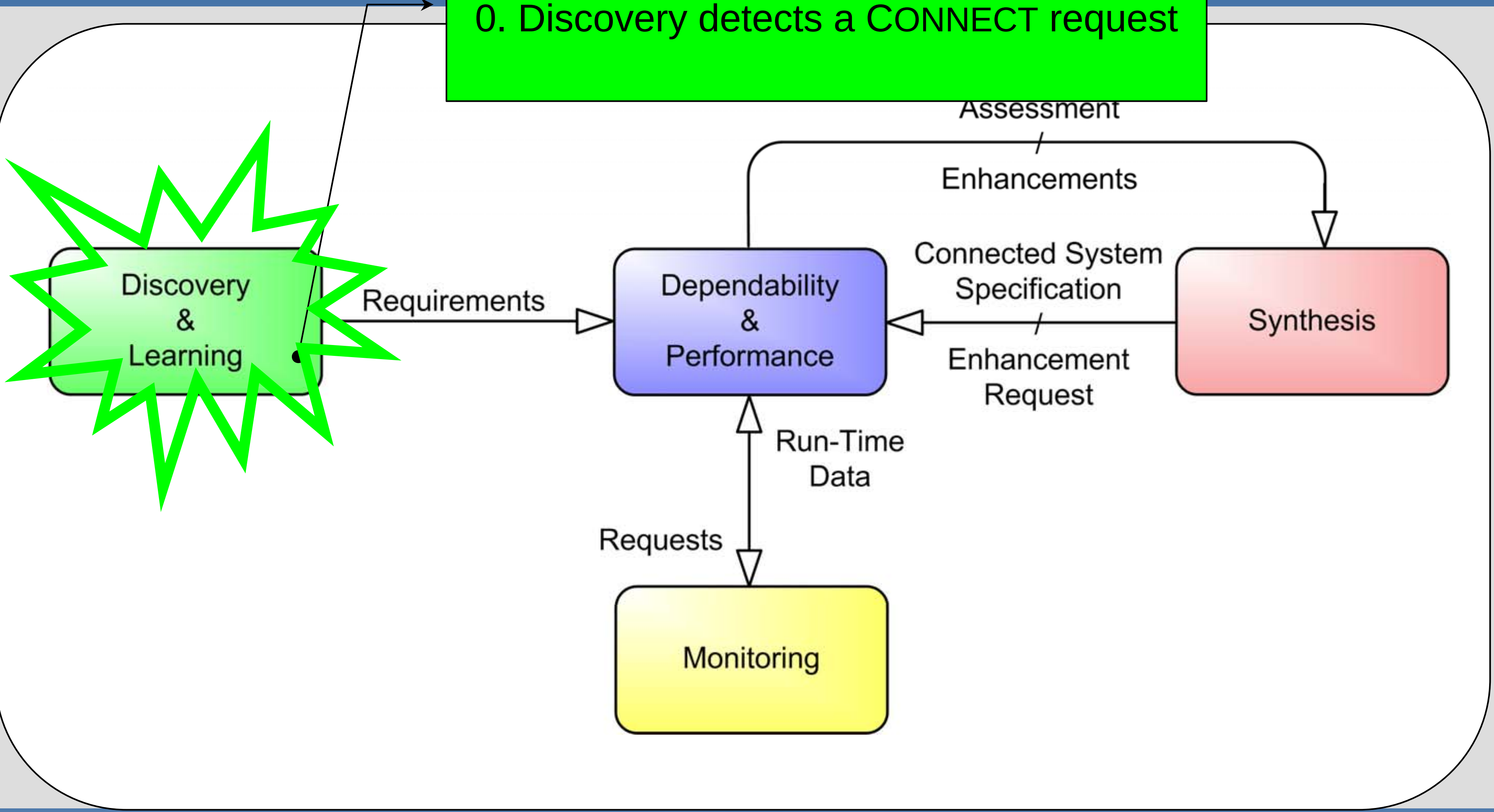
(plus **Incremental Verification** method, not addressed in this lecture)

Dependability Analysis-centric view in CONNECT

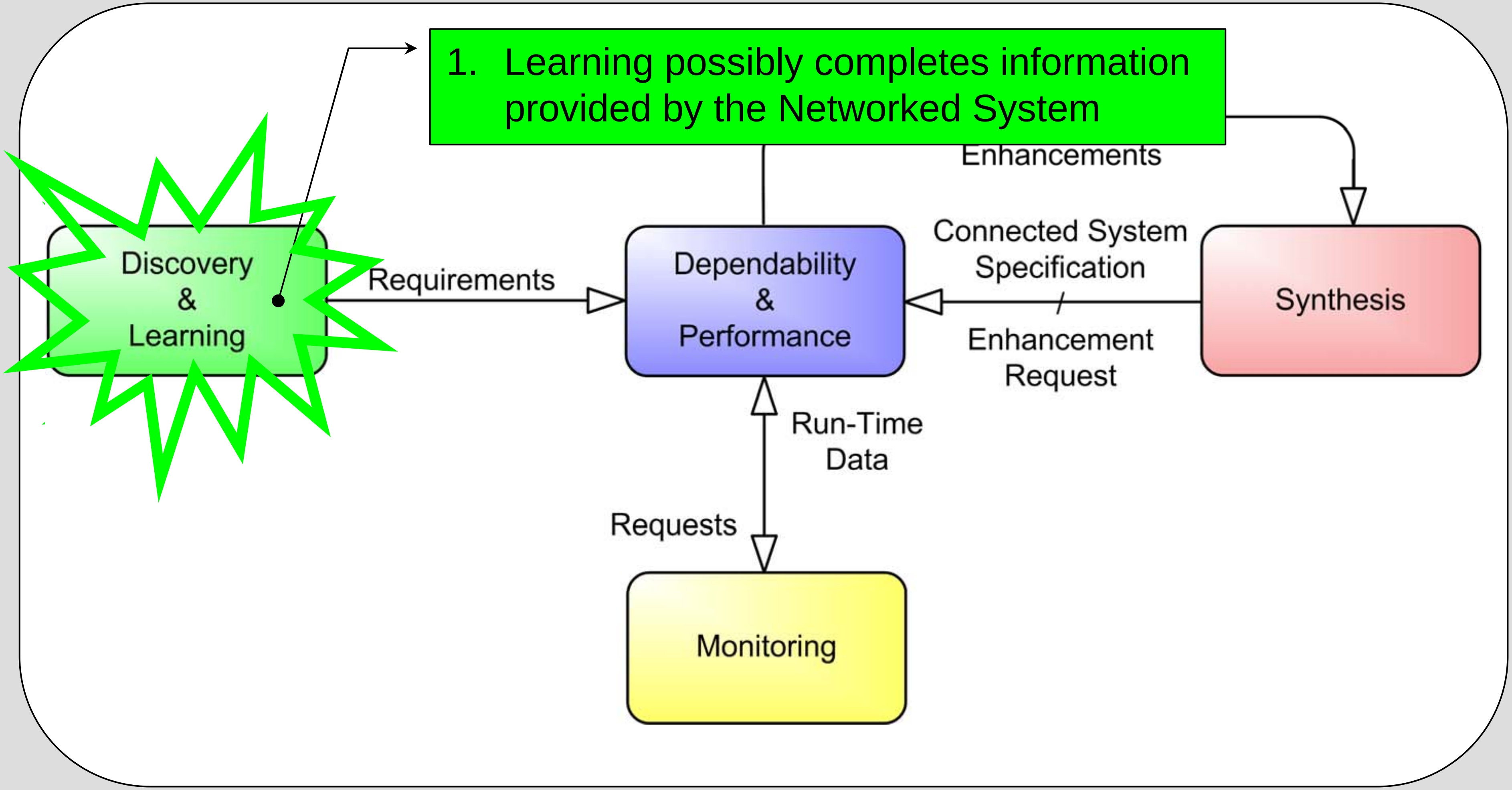


CONNECT in action

0. Discovery detects a CONNECT request



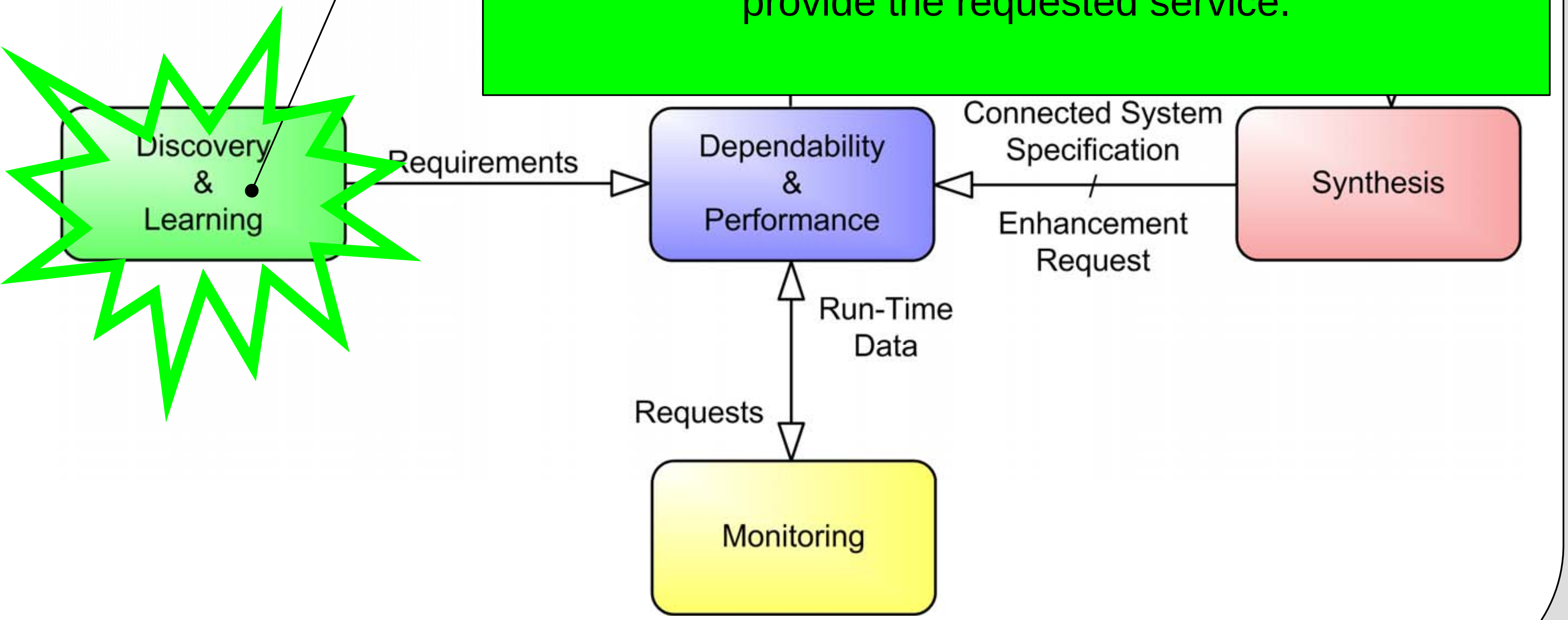
CONNECT in action



1. Learning possibly completes information provided by the Networked System

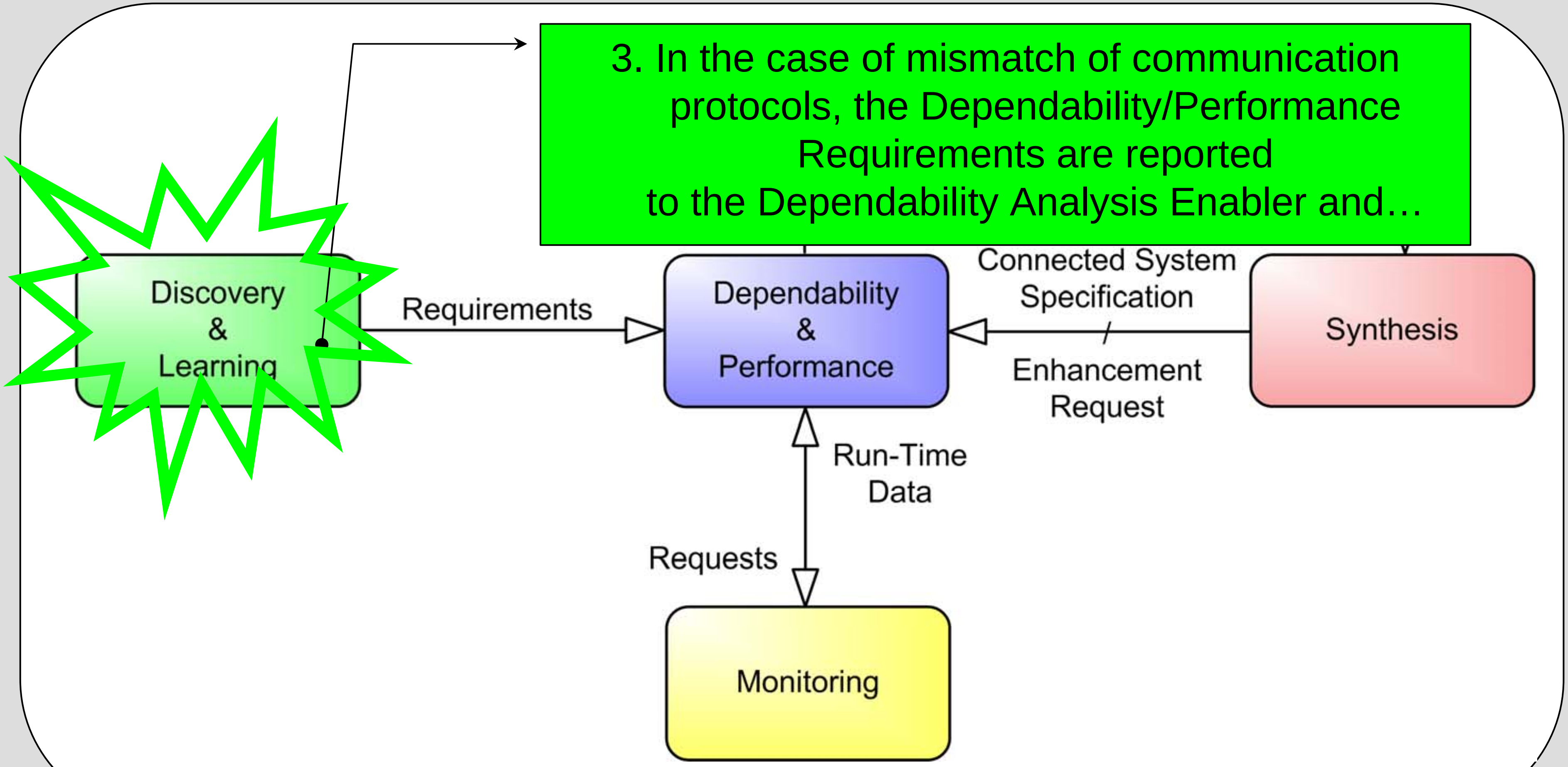
CONNECT in action

2. Discovery seeks a Networked System that can provide the requested service.

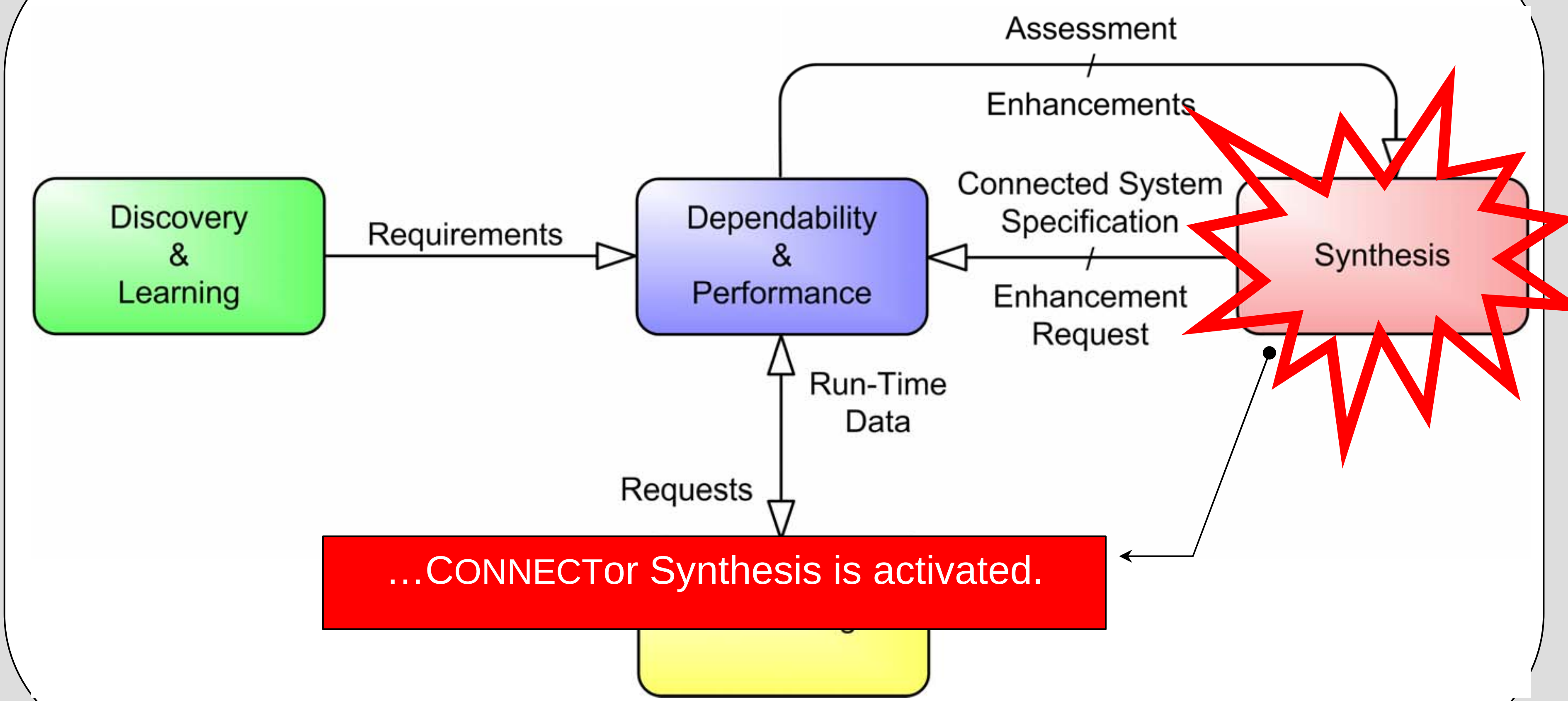


CONNECT in action

3. In the case of mismatch of communication protocols, the Dependability/Performance Requirements are reported to the Dependability Analysis Enabler and…


Discovery & Learning
Requirements
Dependability & Performance
Connected System Specification / Enhancement Request
Synthesis
Run-Time Data
Requests
Monitoring

CONNECT in action



Assessment / Enhancements

Discovery & Learning

Requirements

Dependability & Performance

Connected System Specification / Enhancement Request

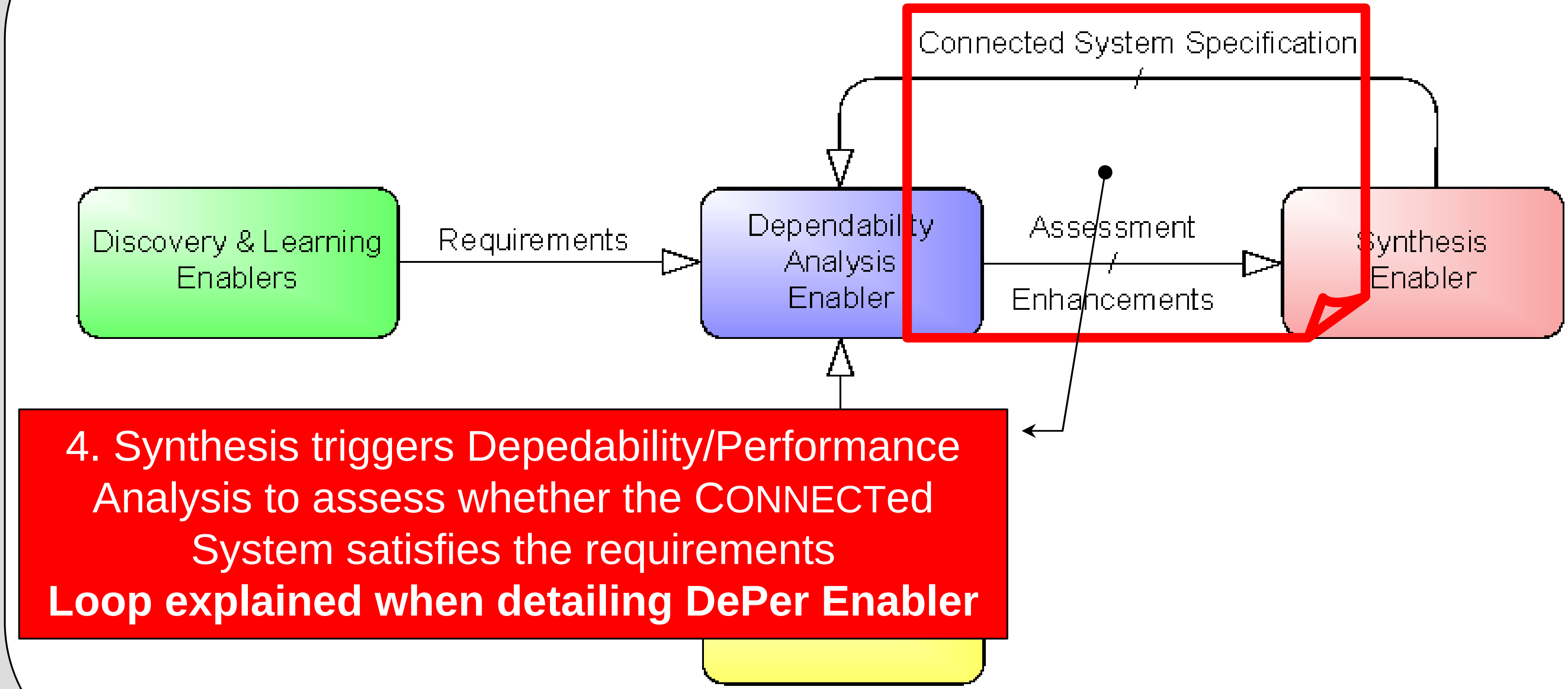
Synthesis

Run-Time Data

Requests

…CONNECTor Synthesis is activated.

CONNECT in action

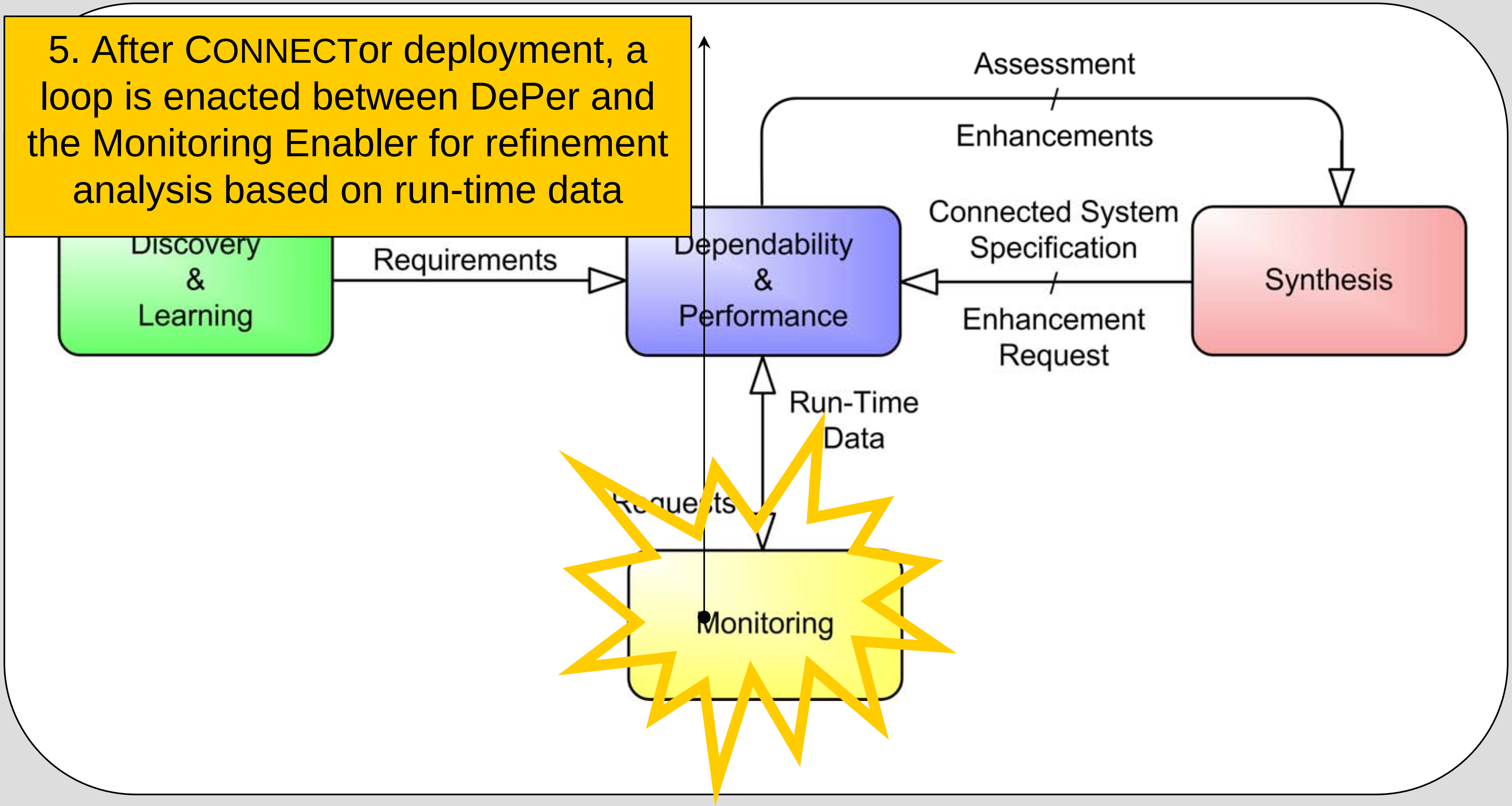

Connected System Specification
Discovery & Learning Enablers
Requirements
Dependability Analysis Enabler
Assessment / Enhancements
Synthesis Enabler

4. Synthesis triggers Depedability/Performance Analysis to assess whether the CONNECTed System satisfies the requirements

Loop explained when detailing DePer Enabler

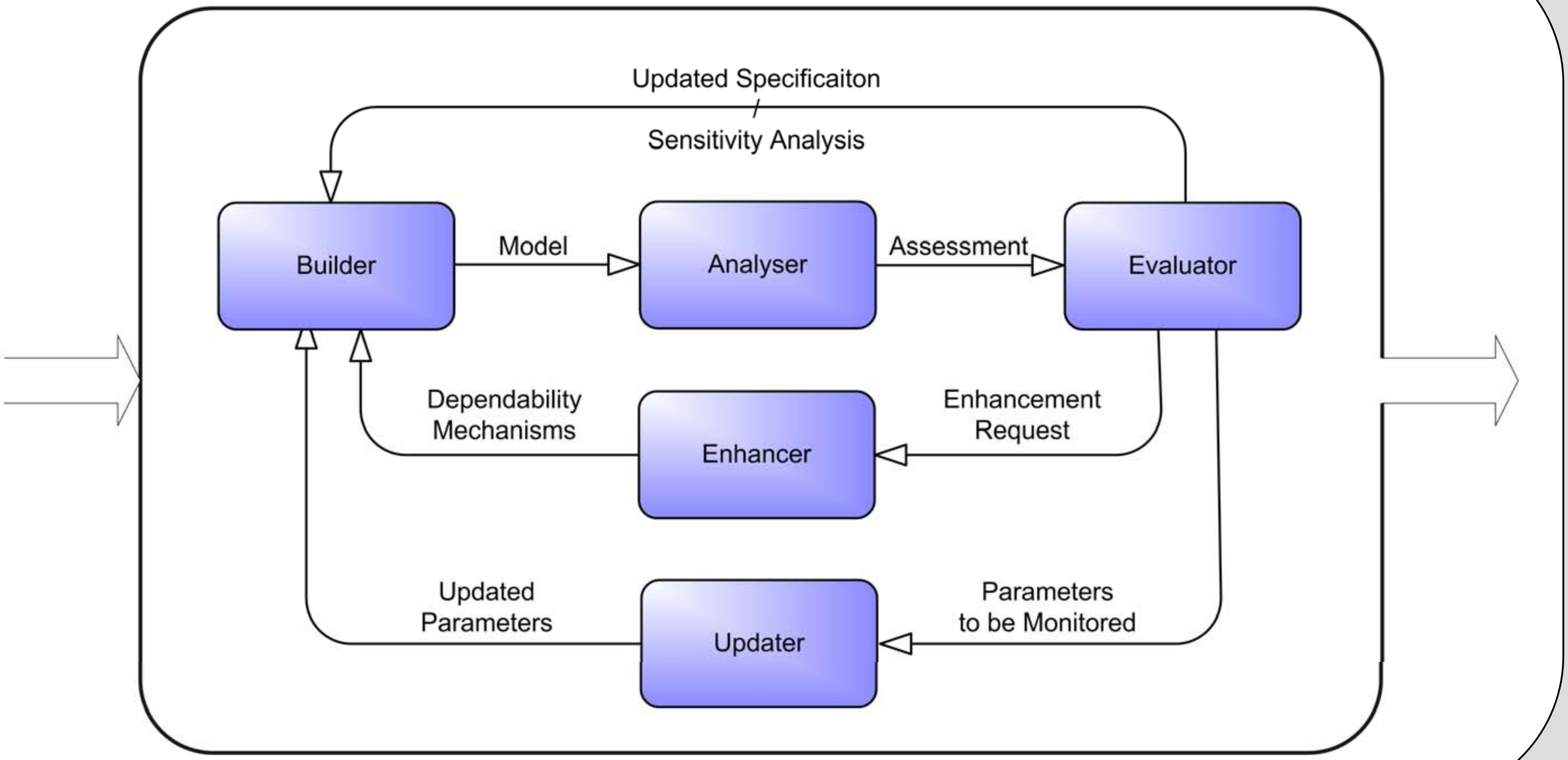
CONNECT in action

5. After CONNECTor deployment, a loop is enacted between DePer and the Monitoring Enabler for refinement analysis based on run-time data

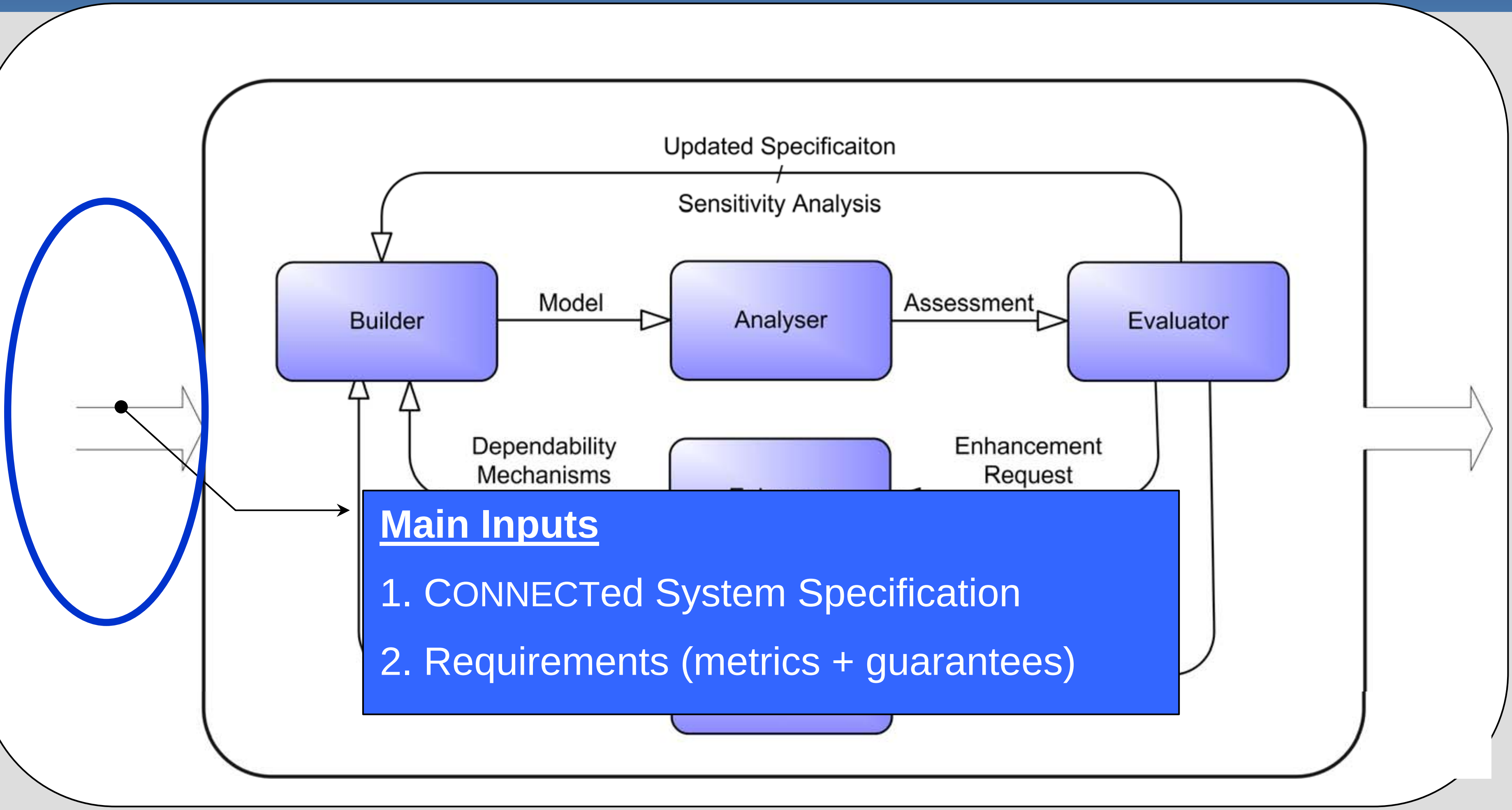


Logical Architecture of the Dependability and Performance Analysis Enabler (DePer)

DePer Architecture



DePer Architecture

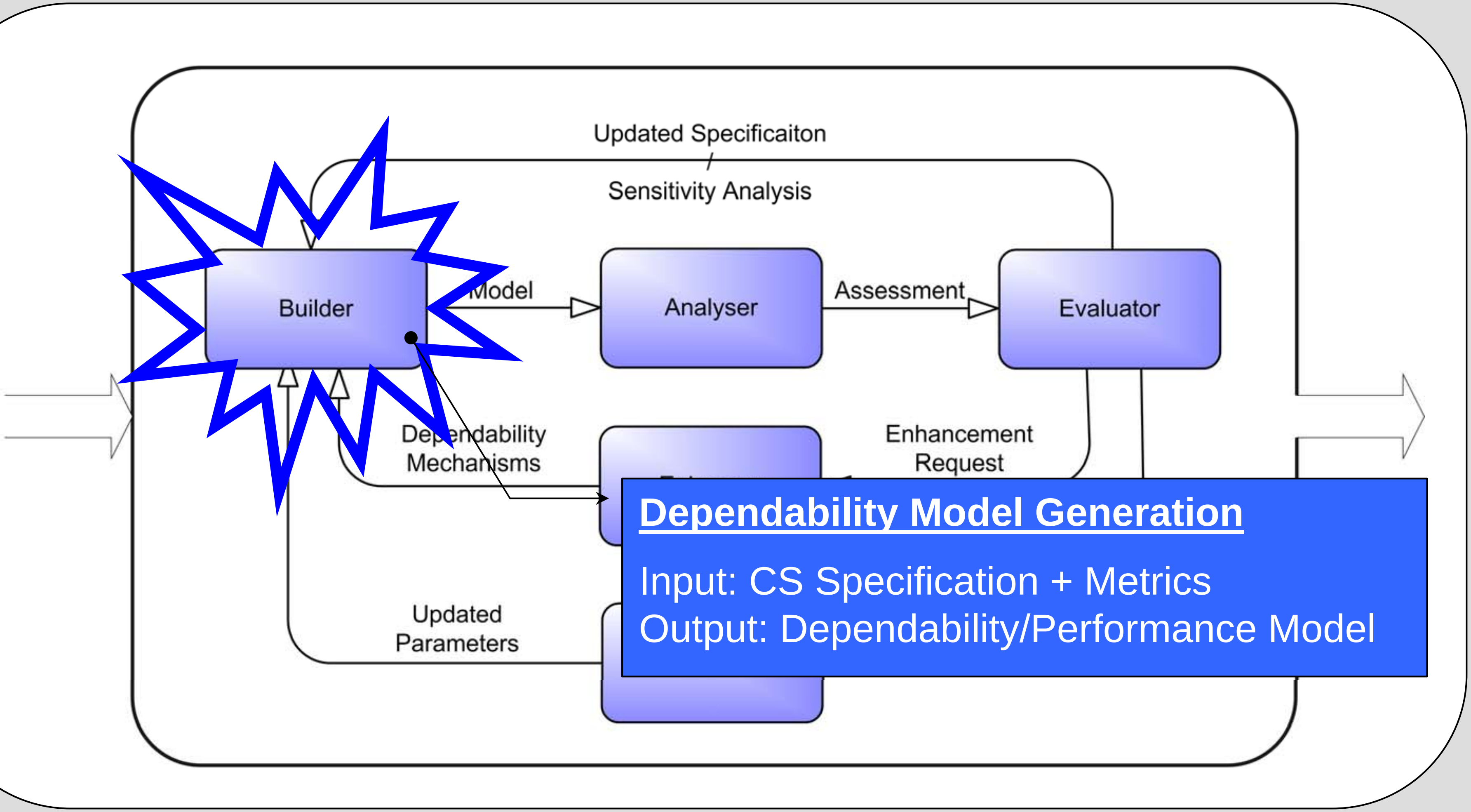


Main Inputs

1. CONNECTed System Specification

2. Requirements (metrics + guarantees)

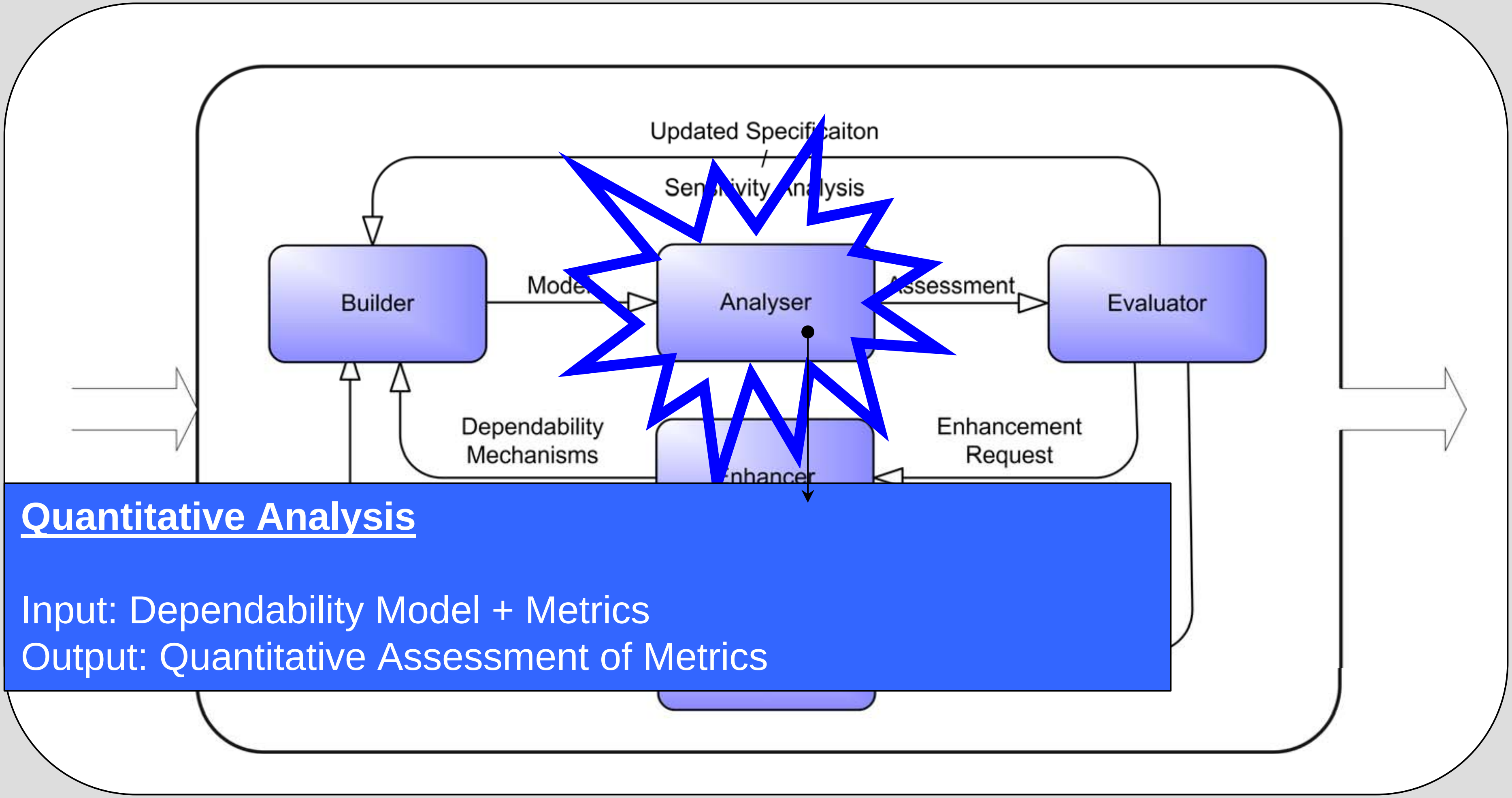
DePer Architecture



Dependability Model Generation

Input: CS Specification + Metrics
Output: Dependability/Performance Model

DePer Architecture



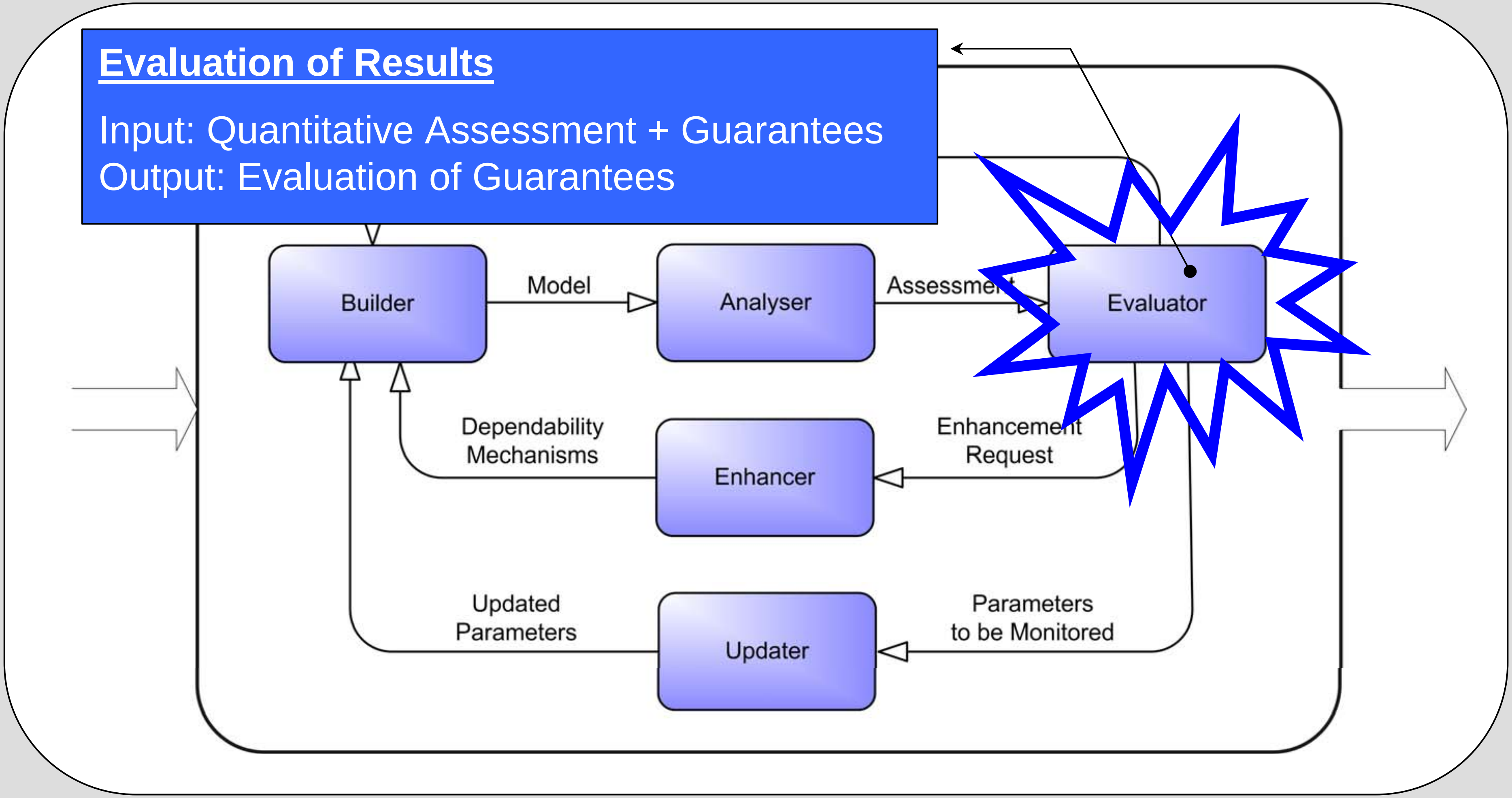
Quantitative Analysis

Input: Dependability Model + Metrics
Output: Quantitative Assessment of Metrics

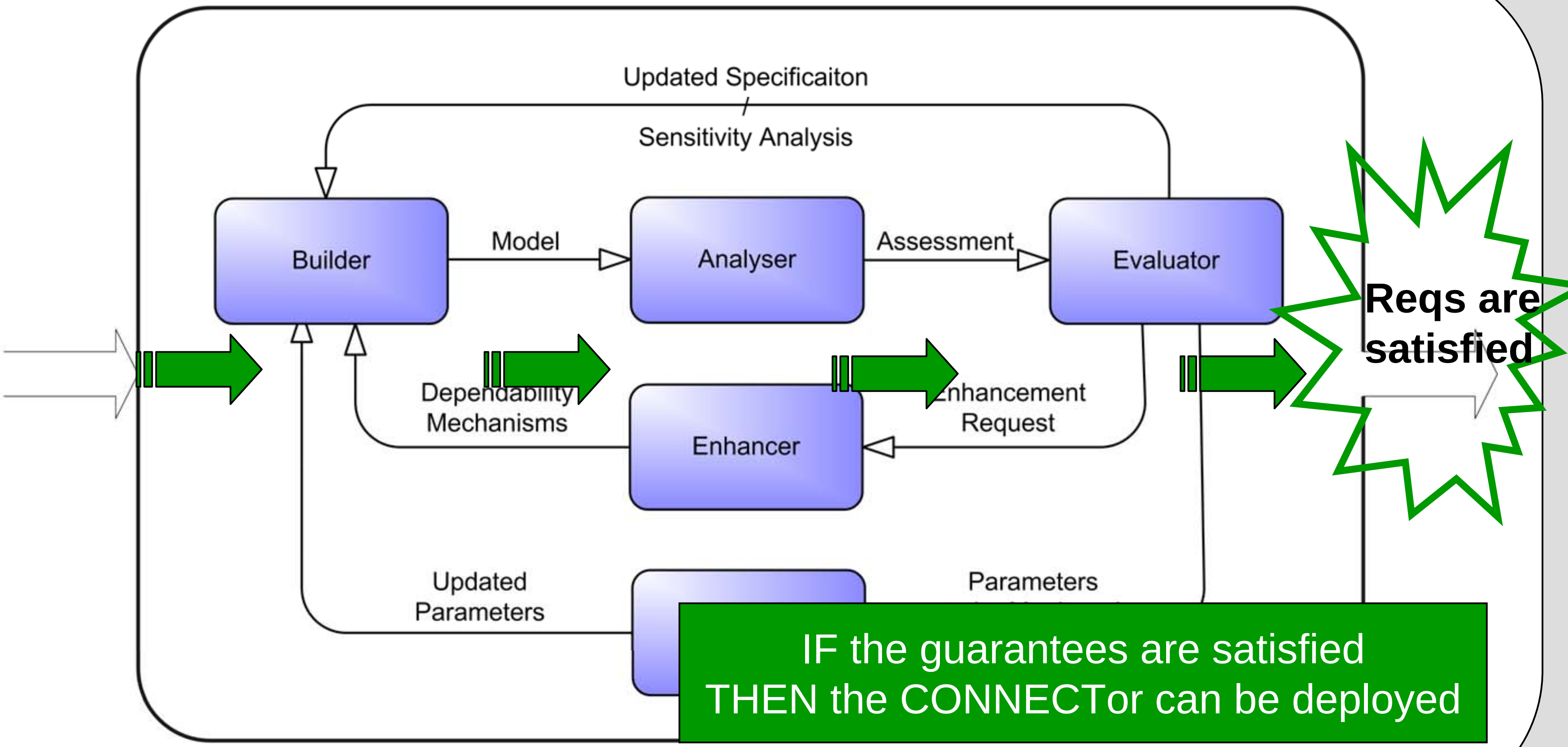
DePer Architecture

Evaluation of Results

Input: Quantitative Assessment + Guarantees
Output: Evaluation of Guarantees



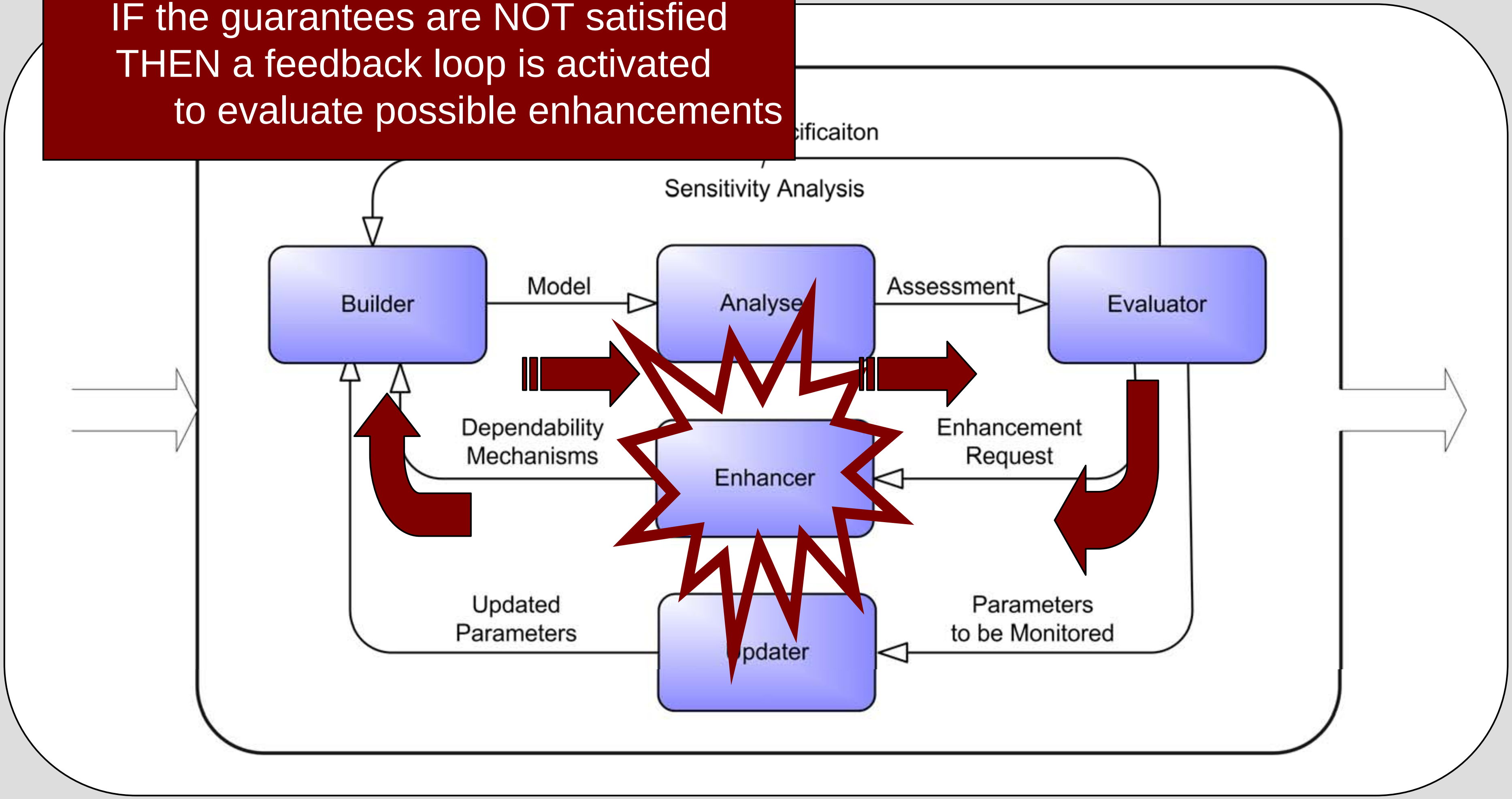
DePer Architecture



IF the guarantees are satisfied
THEN the CONNECTor can be deployed

DePer Architecture

IF the guarantees are NOT satisfied
THEN a feedback loop is activated
to evaluate possible enhancements

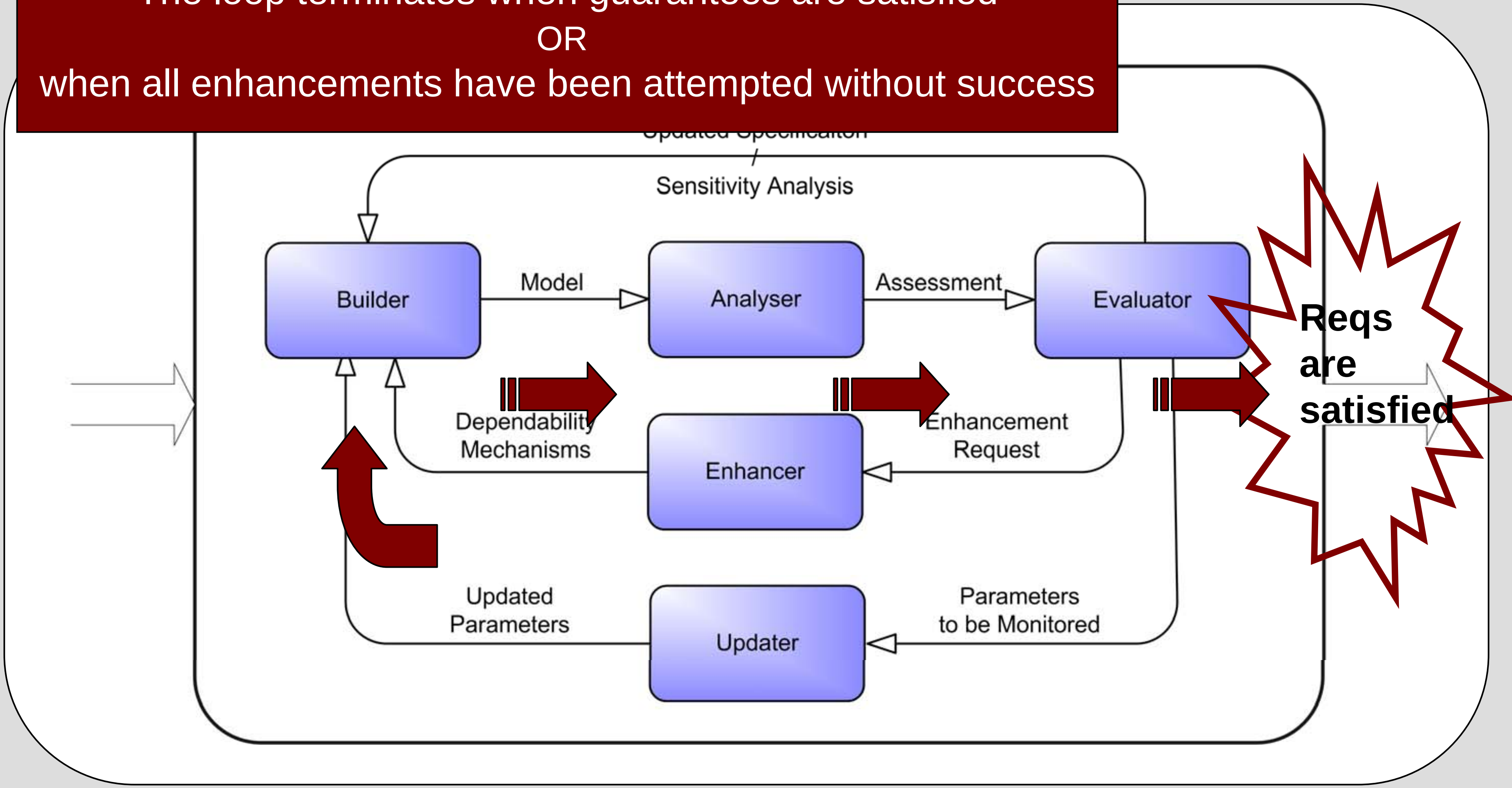


DePer Architecture

The loop terminates when guarantees are satisfied

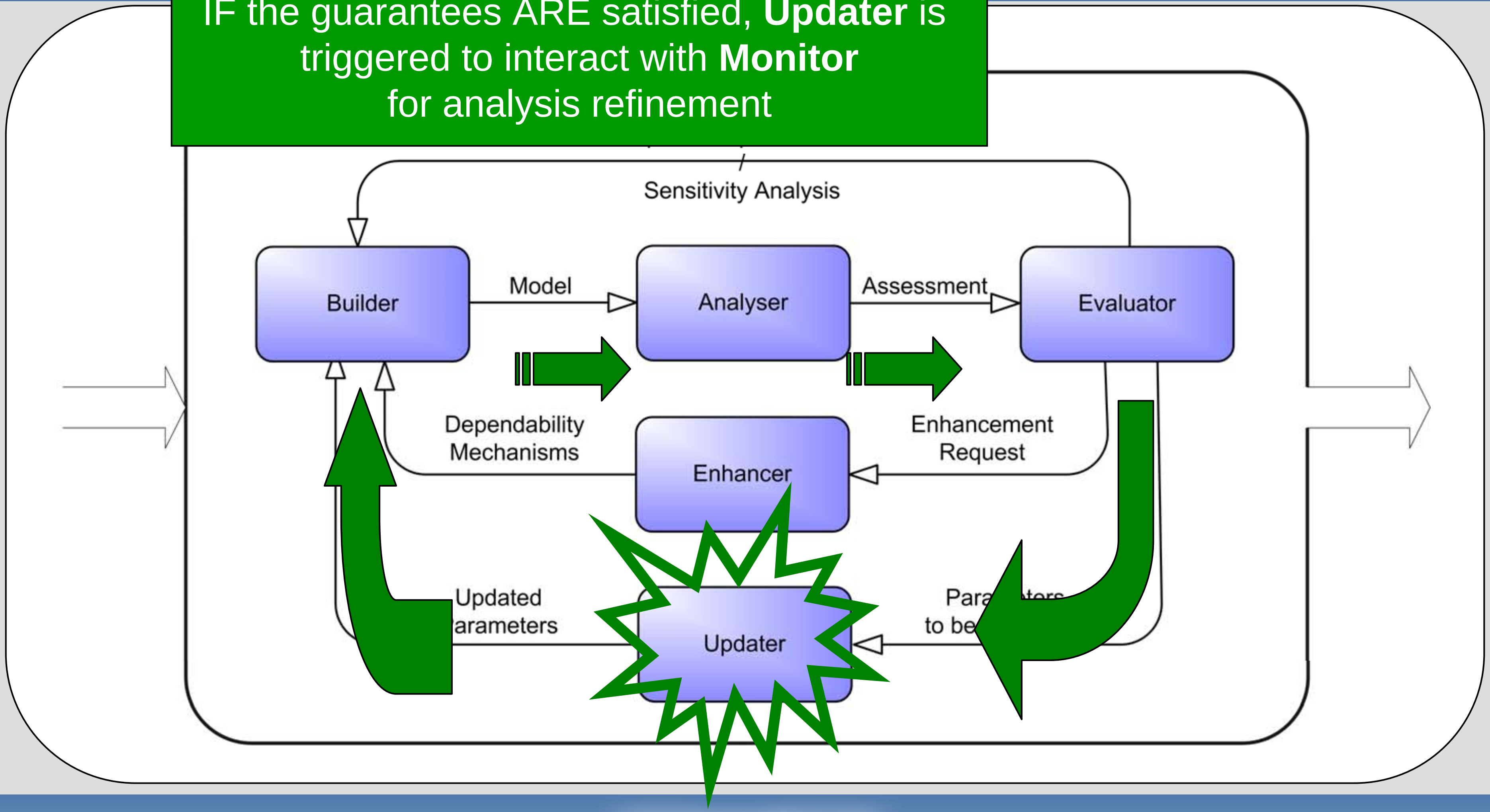
OR

when all enhancements have been attempted without success



DePer Architecture

IF the guarantees ARE satisfied, **Updater** is triggered to interact with **Monitor** for analysis refinement



(Partial) Prototype Implementation

- **DePer:** http://dcl.isti.cnr.it/DEA
- Modules implemented in Java
- I/O data format in XML
- Exploits features of existing tools
  - **GENET**: http://www.lsi.upc.edu/~jcarmona/genet.html
  - **Mobius**: https://www.mobius.illinois.edu/
    and SAN modeling formalism

The CONNECT Monitoring Infrastructure GLIMPSE

Monitoring into CONNECT

- A CONNECT-transversal functionality supporting on-line assessment for different purposes:
    - “assumption monitoring” for CONNECTors
    - QoS assessment and dependability analysis
    - learning
    - security and trust management

GLIMPSE solution

- GLIMPSE (Generic fLexIble Monitoring based on a Publish Subscribe infrastructurE)
  - flexible, generic, distributed
  - based on a publish-subscribe infrastructure
  - decouples the high-level event specification from observation and analysis

Model-driven approach

- Functional and non functional properties of interest can be specified as instances of an eCore metamodel
  - Advantages
    - an editor that users can use for specifying properties and metrics to be monitorated
    - automated procedures (Model2Code transformations) for instrumenting GLIMPSE

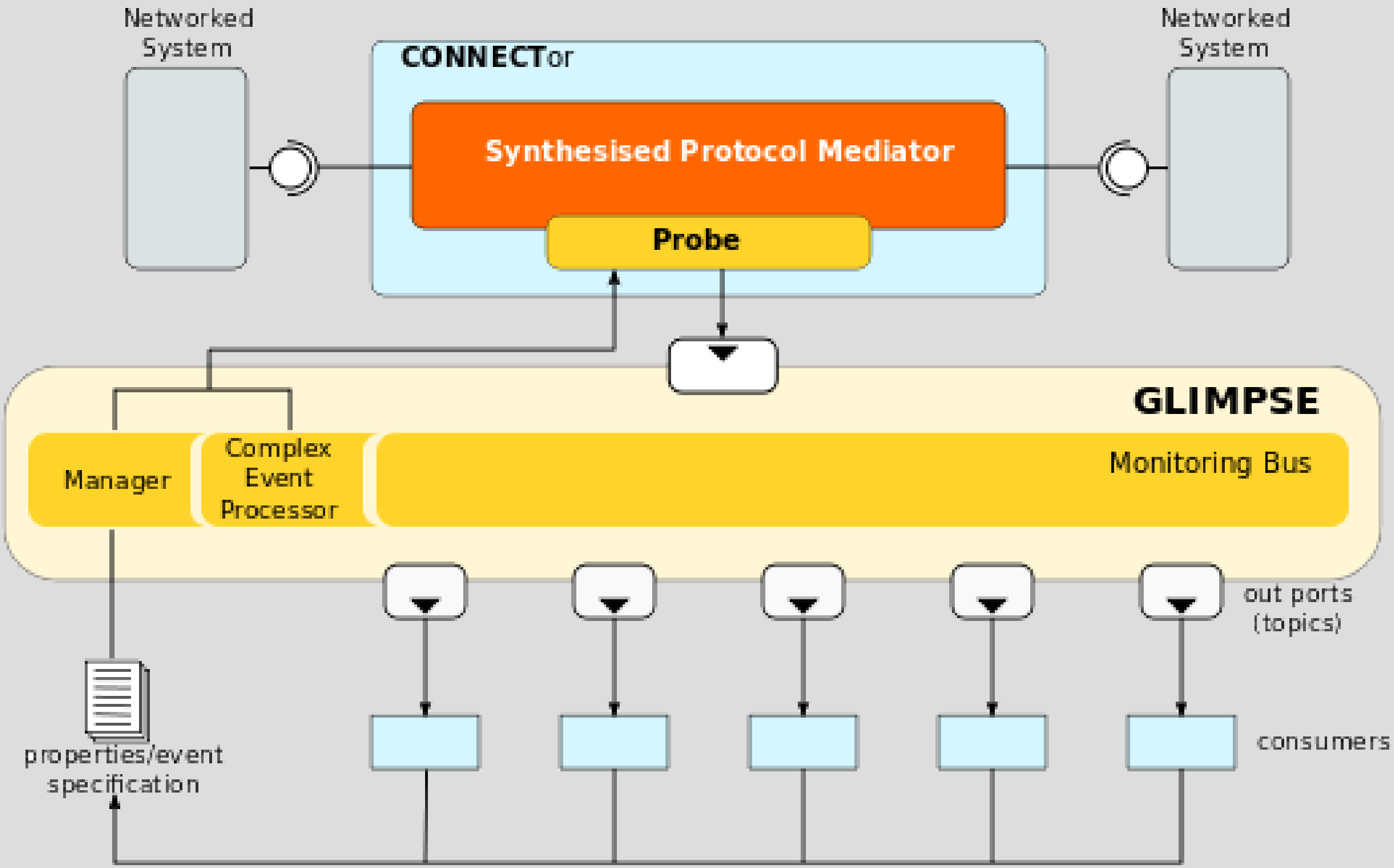
CONNECT Property Meta-Model (CPMM)

- Ongoing work: CONNECT Property Meta-Model (CPMM) expresses relevant properties for the project
  - prescriptive (required) properties
    - *The system S in average must respond in 3 ms in executing the e1 operation with a workload of 10 e2 operations*
  - descriptive (owned) properties
    - *The system S in average responds in 3 ms in executing the e1 operation with a workload of 10 e2 operations*

CONNECT Property Meta-Model (CPMM)

- Qualitative properties
  - events that are observed and cannot be measured
  - e.g., deadlock freeness or liveness
- Quantitative properties
  - quantiable/measurable observations of the system that have an associated metric
  - e.g., performance measures
- The models conforming to CPMM can be used to drive the instrumentation of the monitoring Enabler

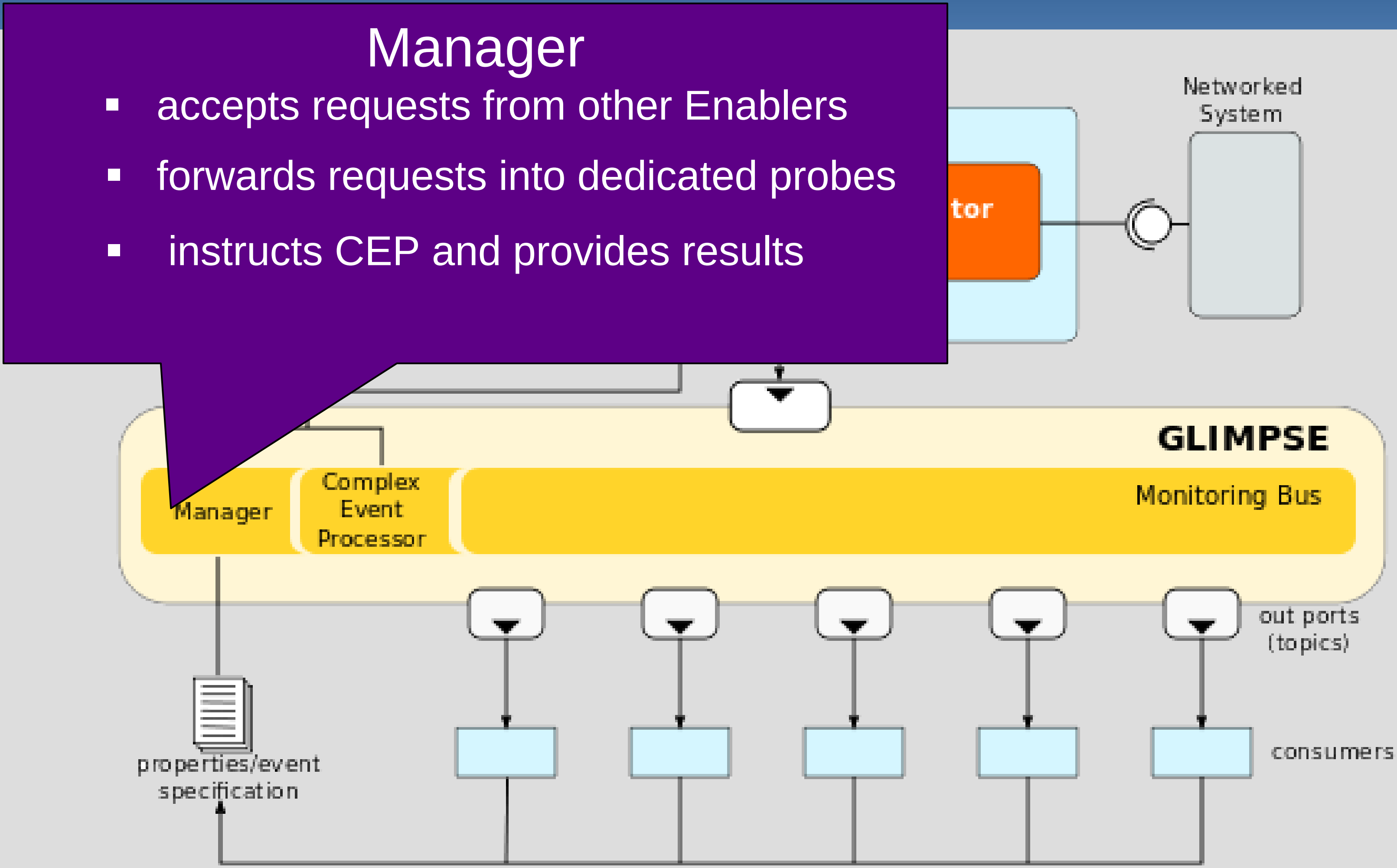
GLIMPSE architecture overview



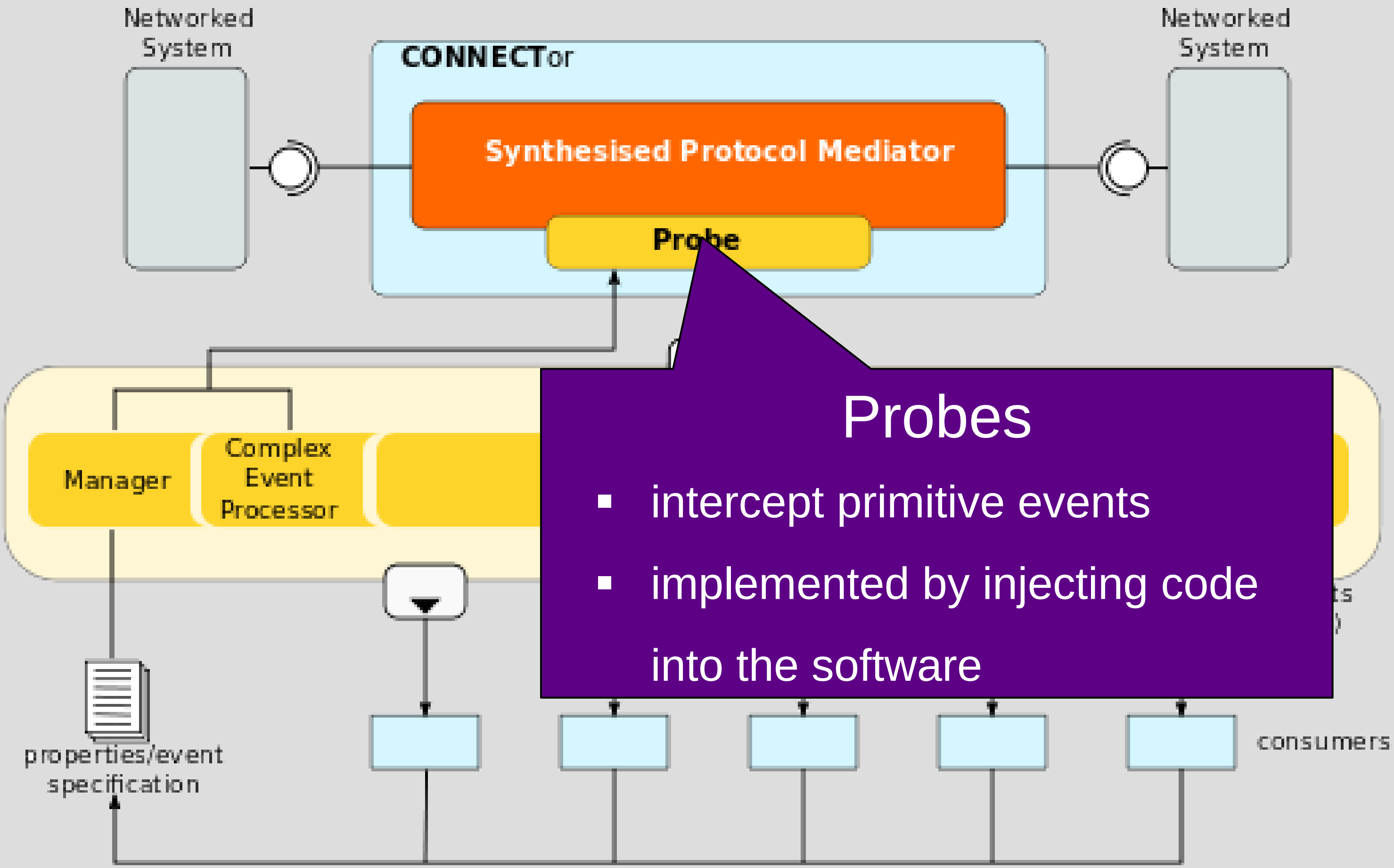
GLIMPSE architecture components

Manager

- accepts requests from other Enablers
- forwards requests into dedicated probes
- instructs CEP and provides results



GLIMPSE architecture components



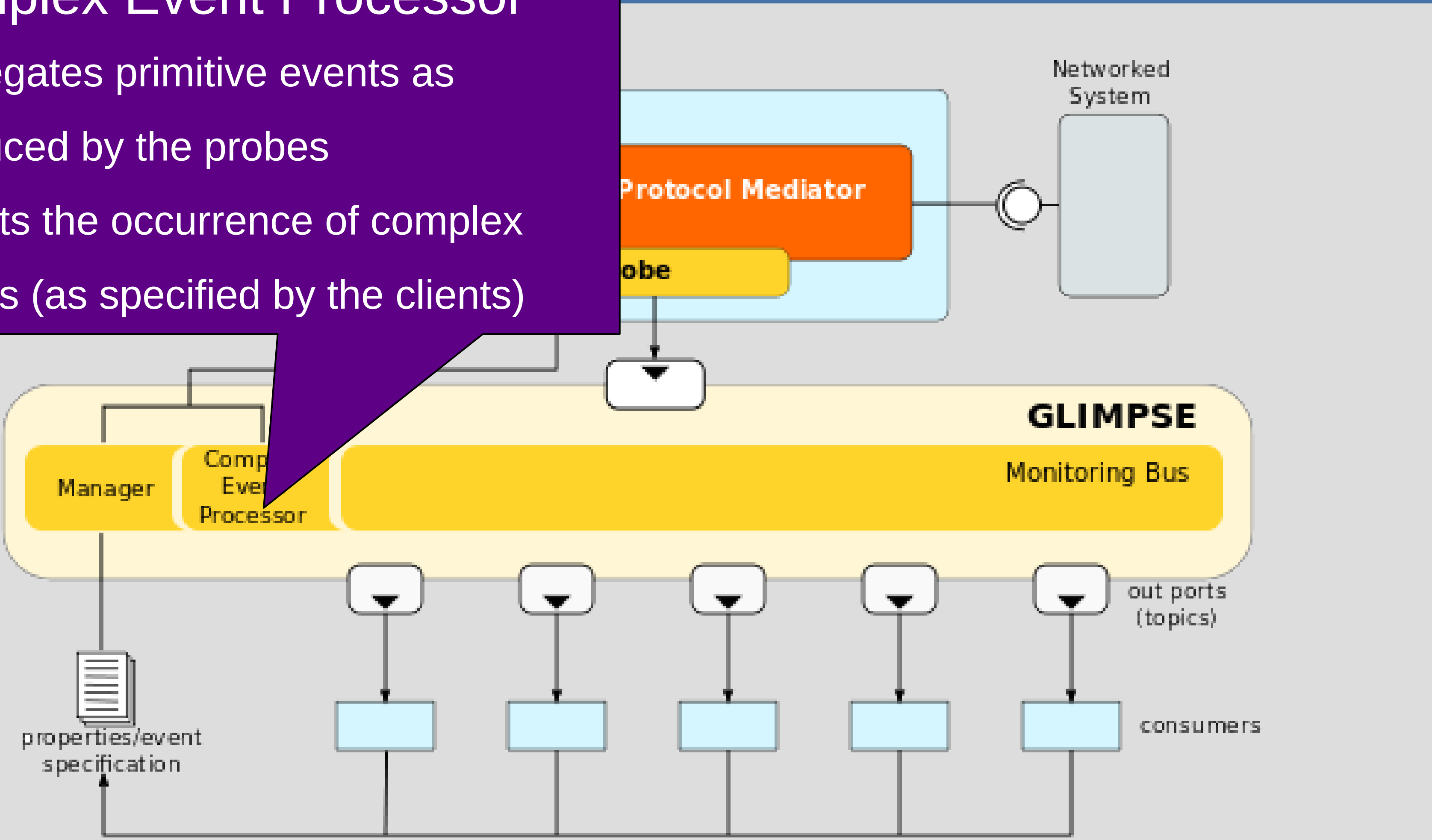
Probes

- intercept primitive events
- implemented by injecting code into the software

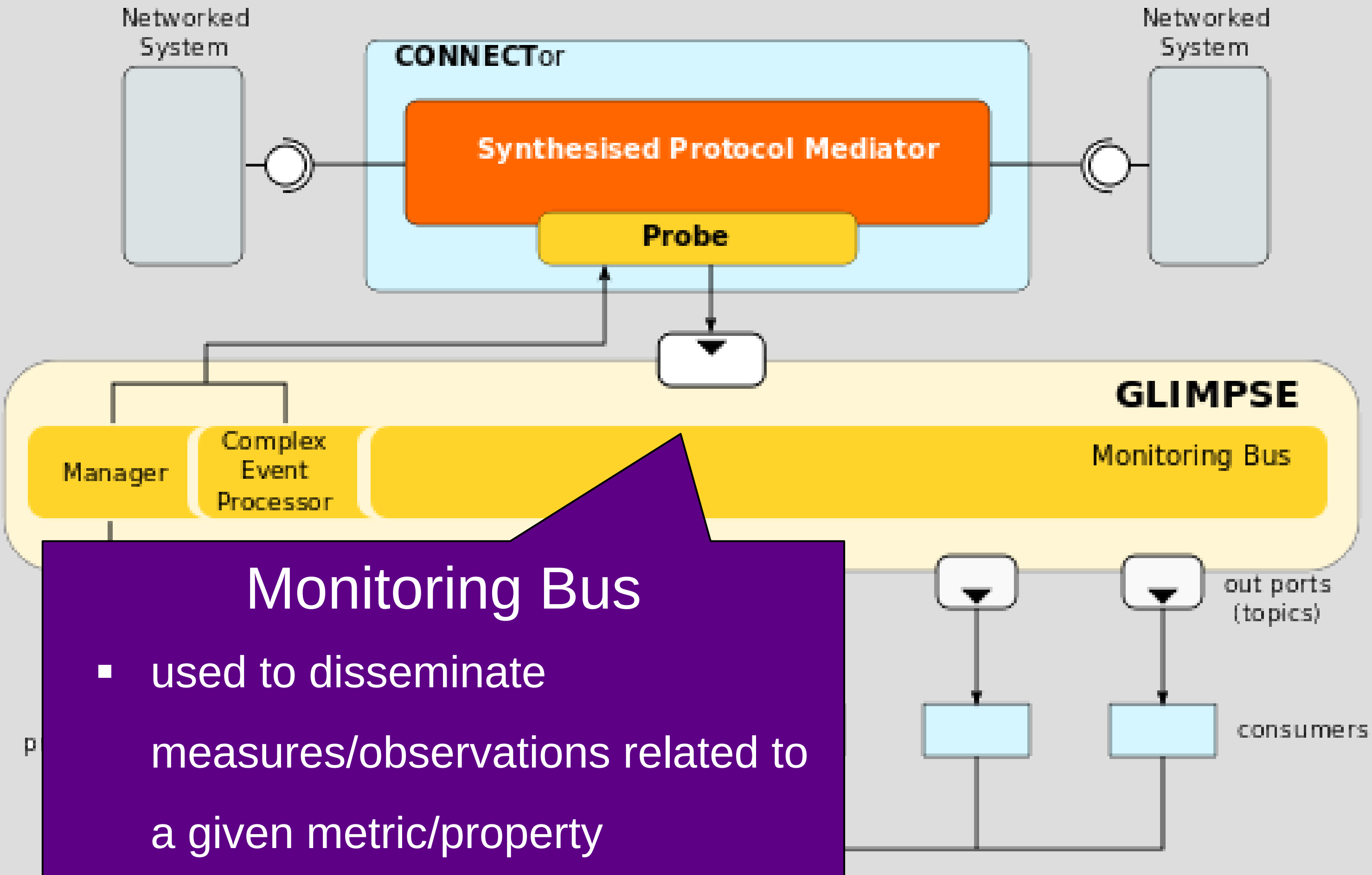
GLIMPSE architecture components

Complex Event Processor

- aggregates primitive events as produced by the probes
- detects the occurrence of complex events (as specified by the clients)



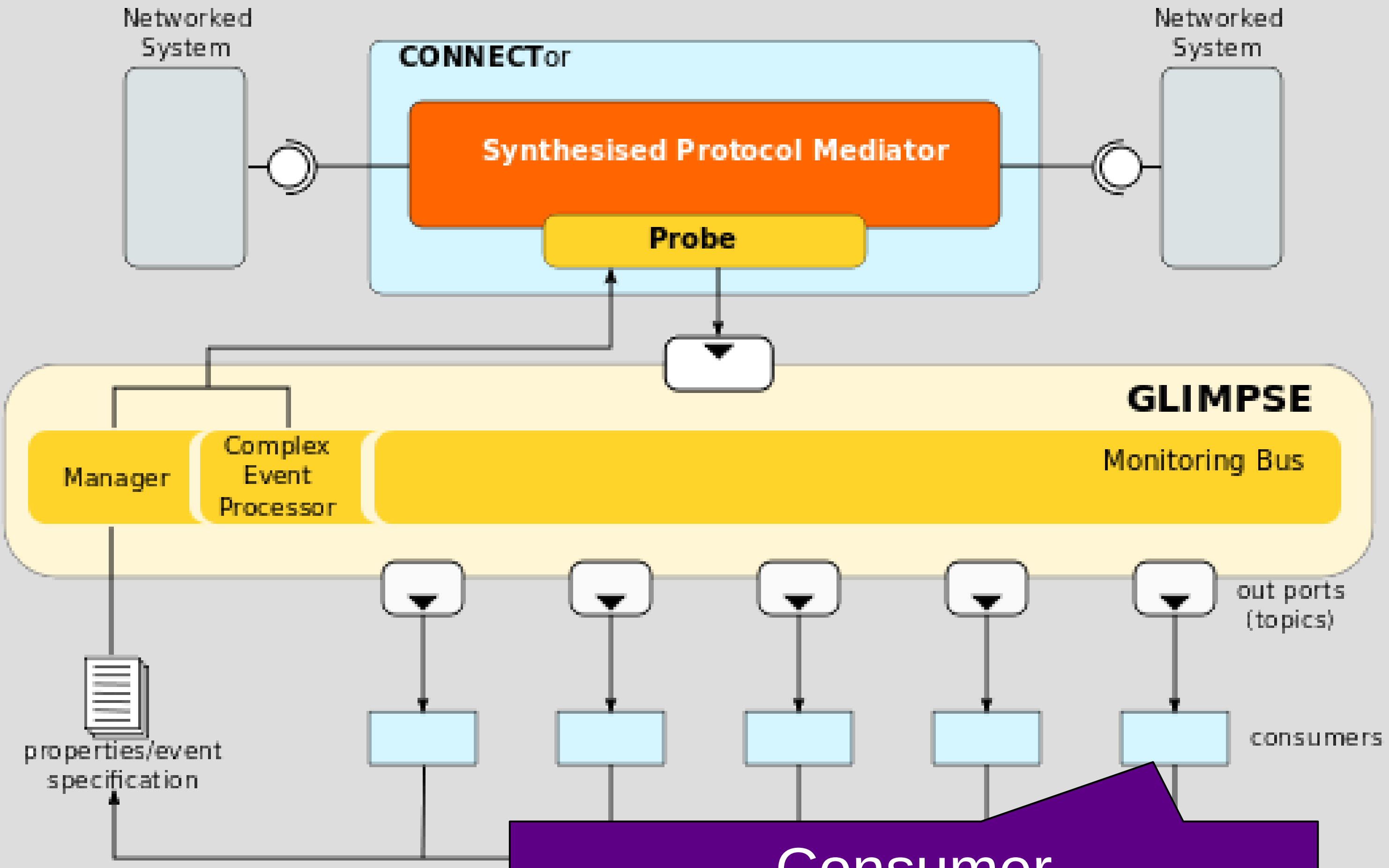
GLIMPSE architecture components



Monitoring Bus

- used to disseminate measures/observations related to a given metric/property
- publish-subscribe paradigm

GLIMPSE architecture components



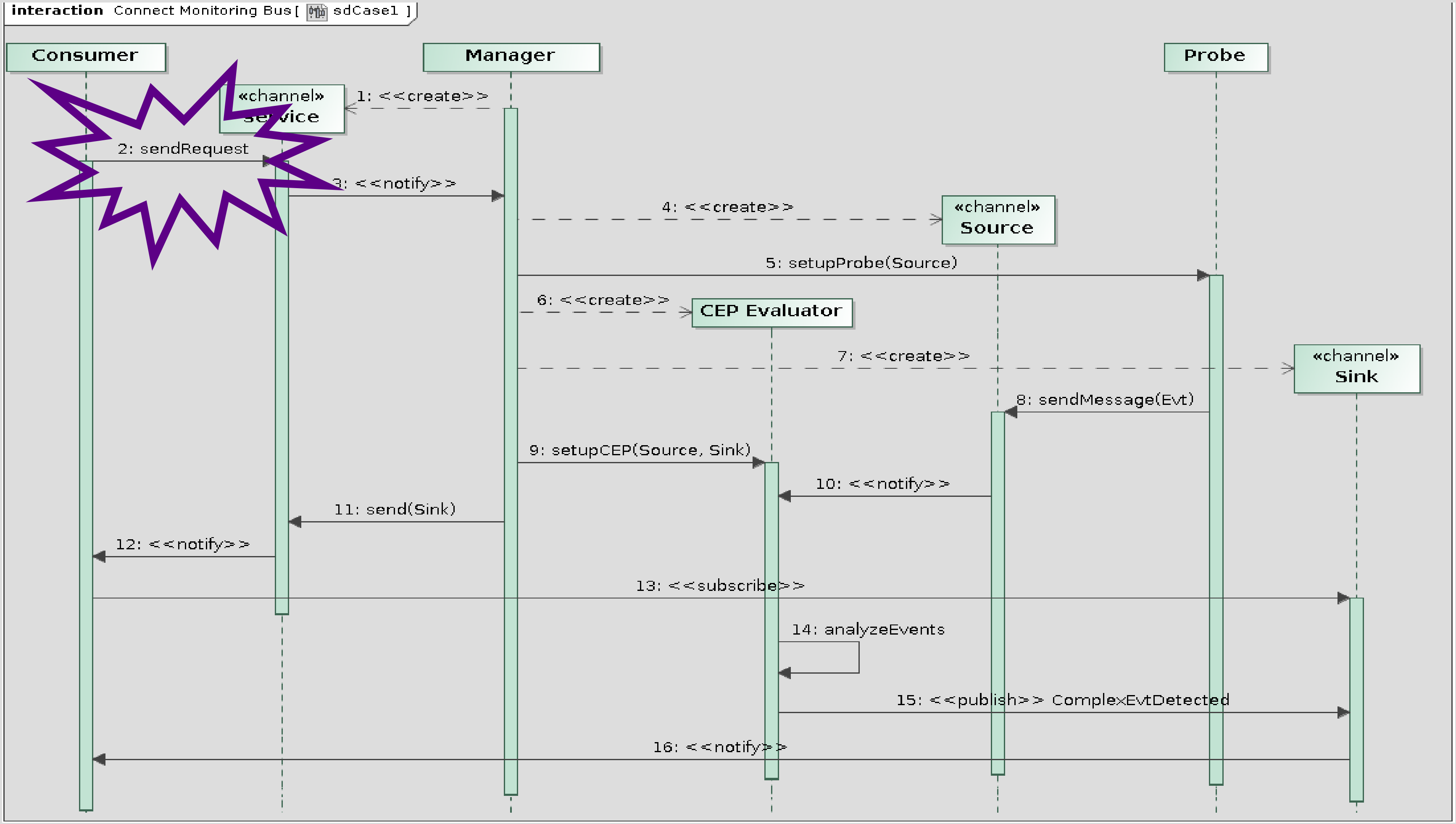
Consumer

- requests the information to be monitored

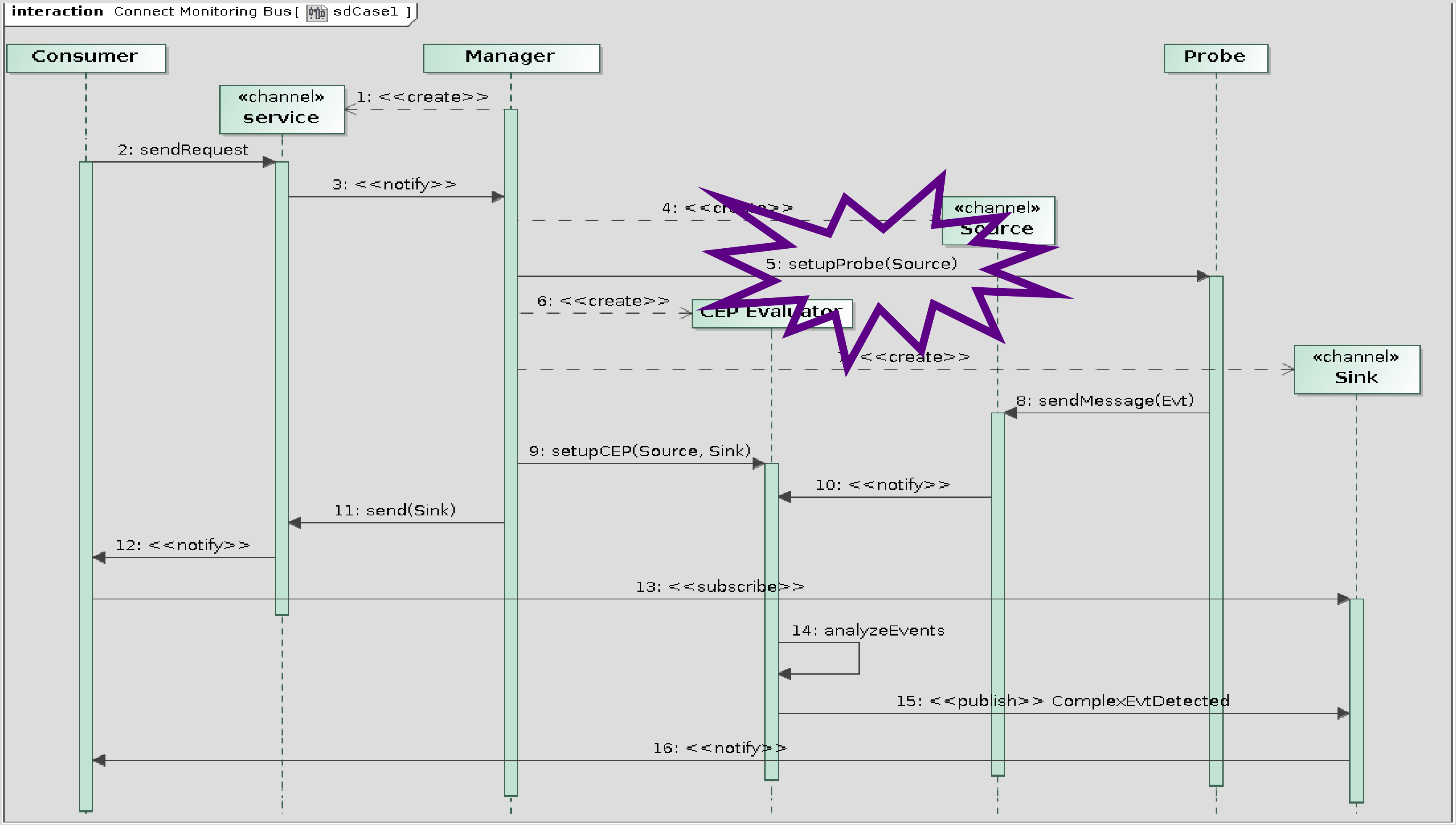
Used Technology

- Monitoring Bus
  - ServiceMix4
    - open source Enterprise Service Bus
    - supports an open source message broker like ActiveMQ
- Complex Event Processing
  - Jboss Drools Fusion
- Model-driven tools (Eclipse-based)
  - Model transformation languages (ATL, Acceleo)

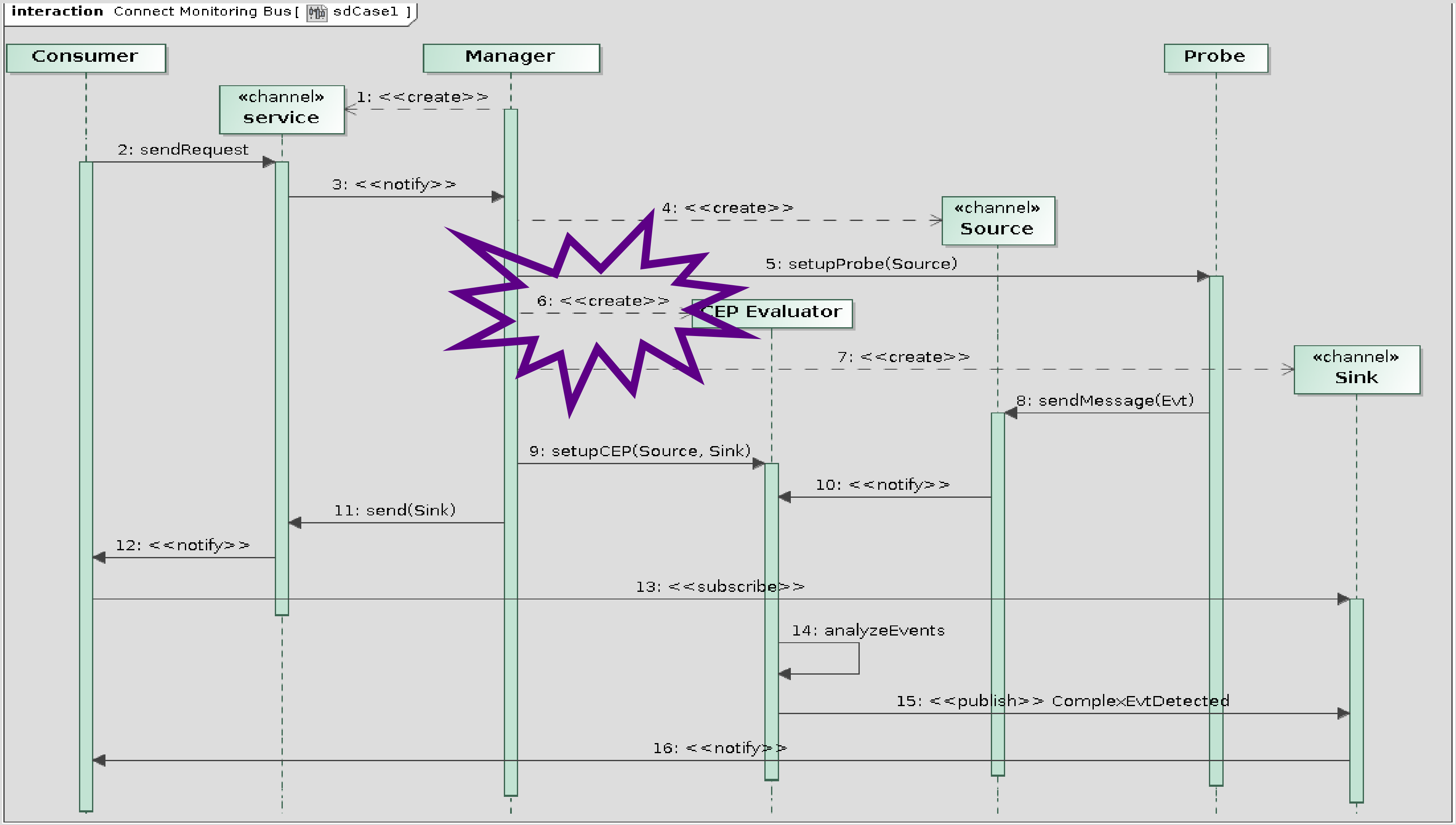
Interaction Pattern



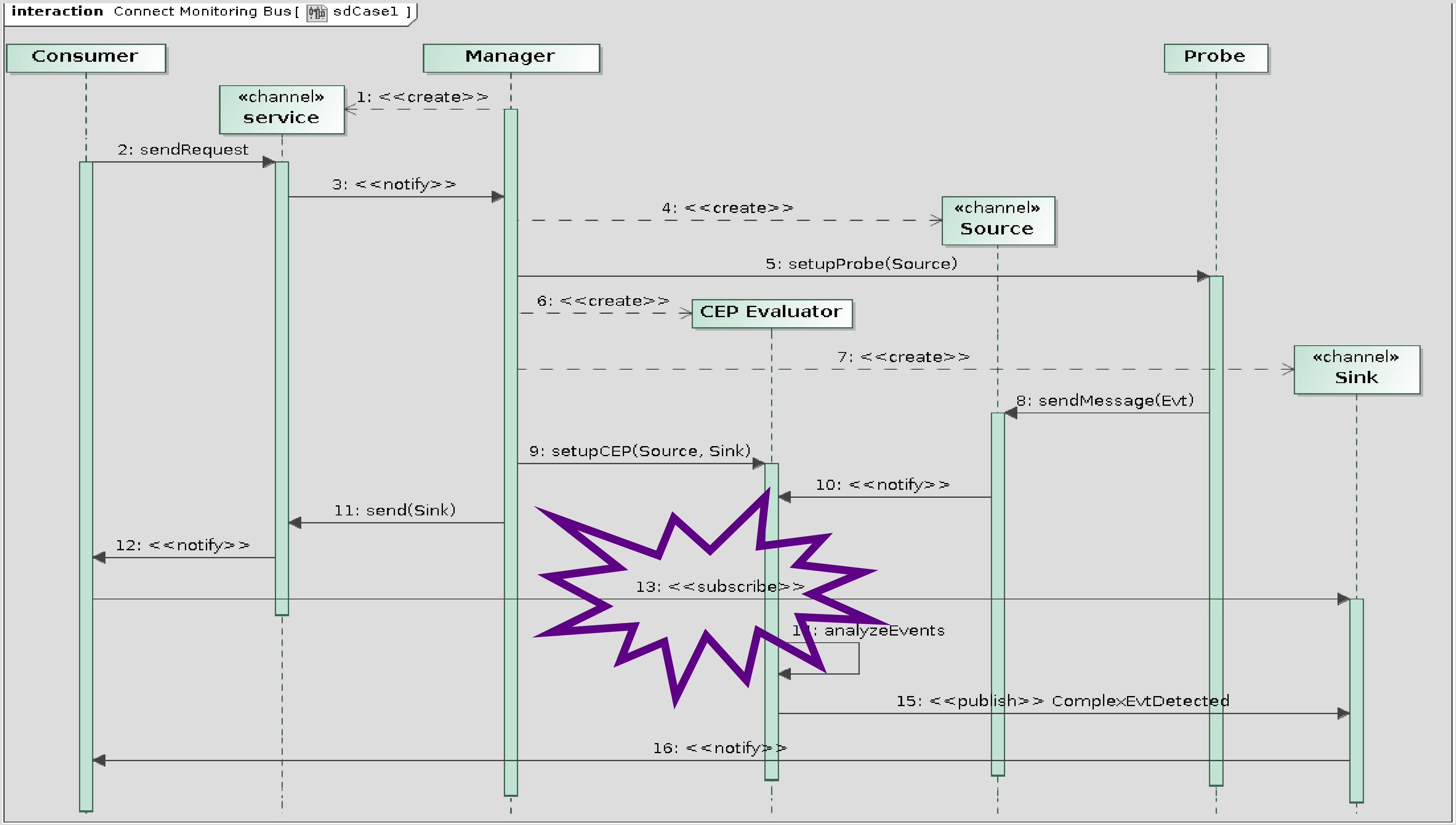
Interaction Pattern



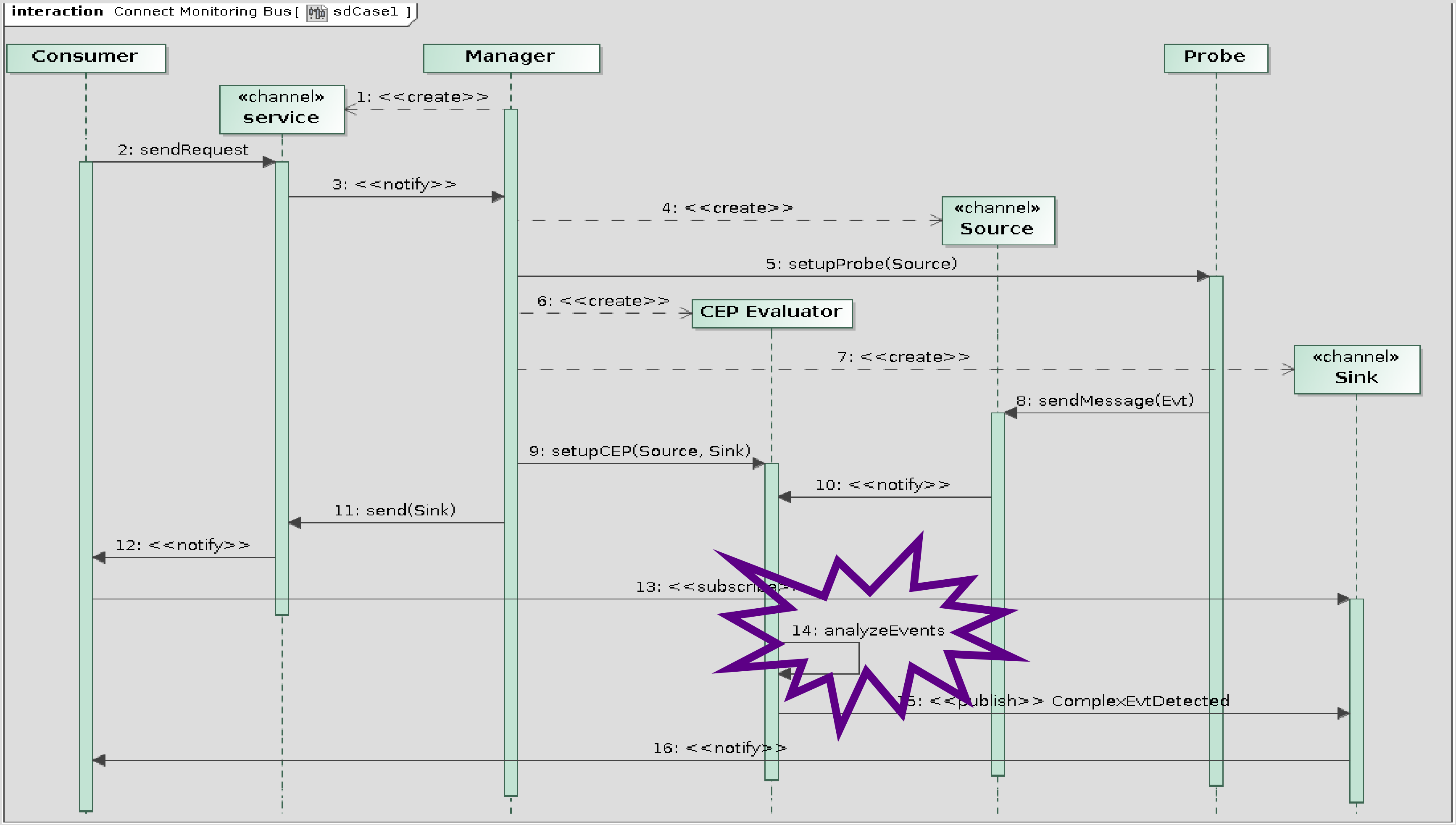
Interaction Pattern



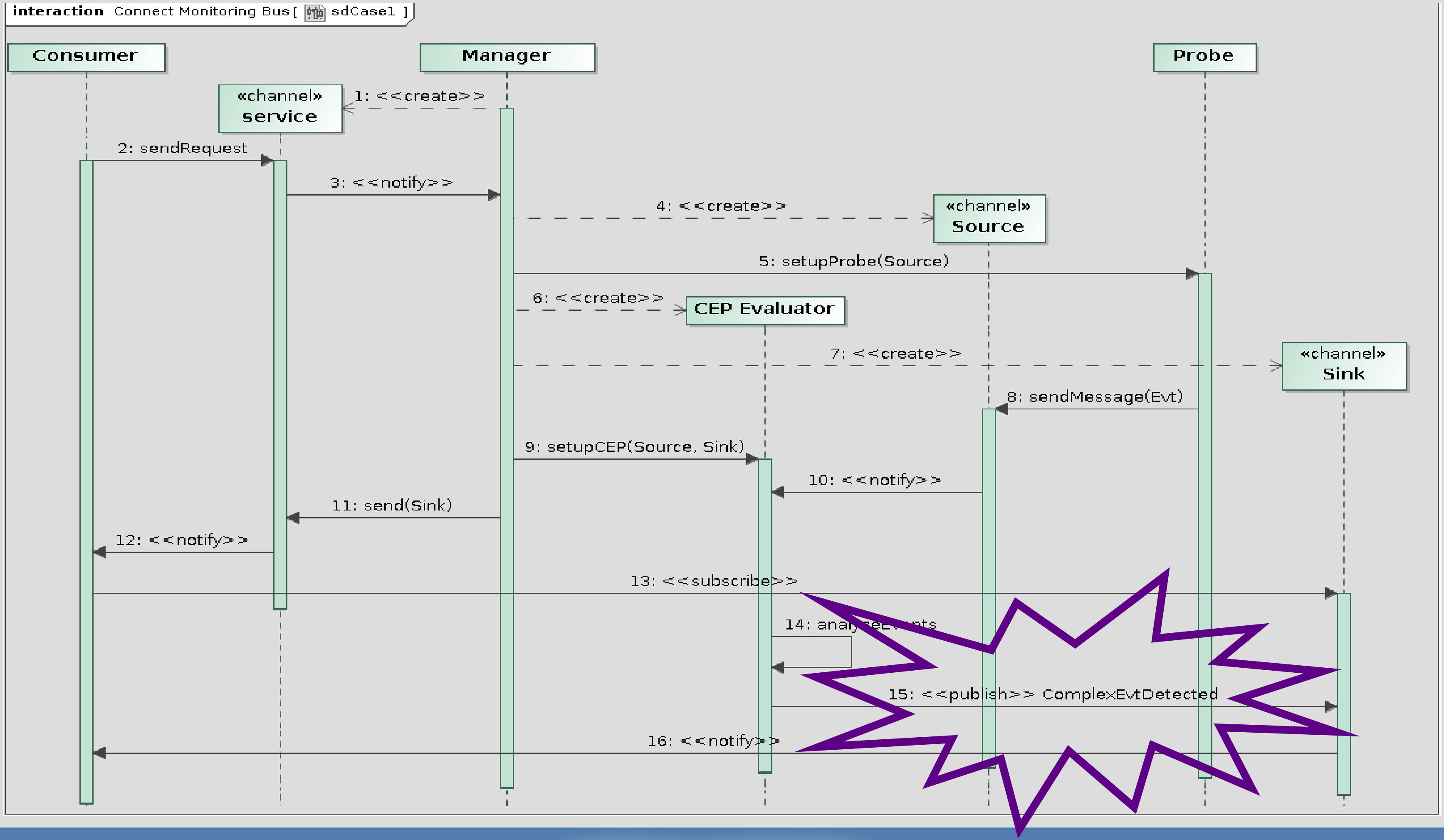
Interaction Pattern



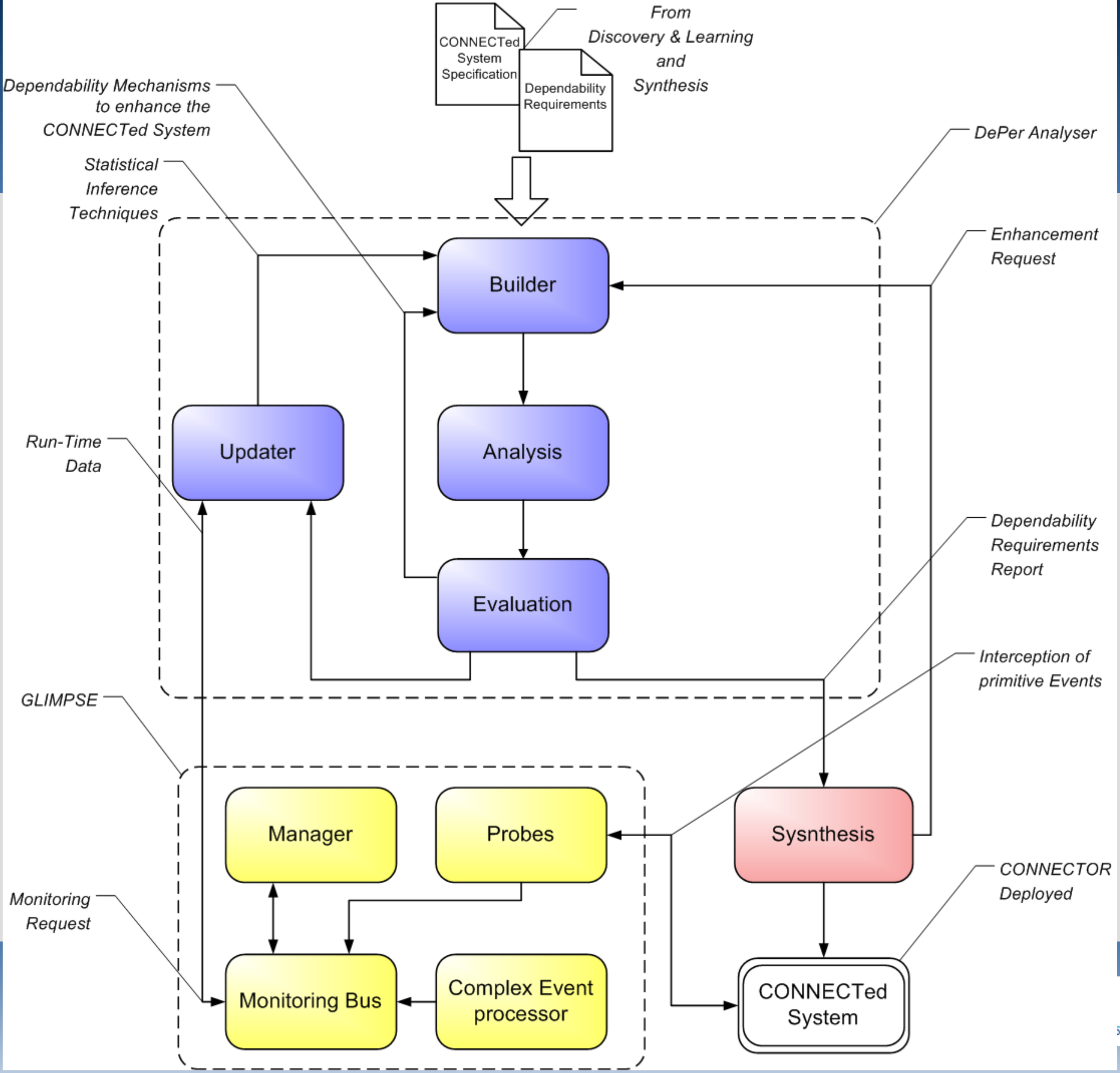
Interaction Pattern



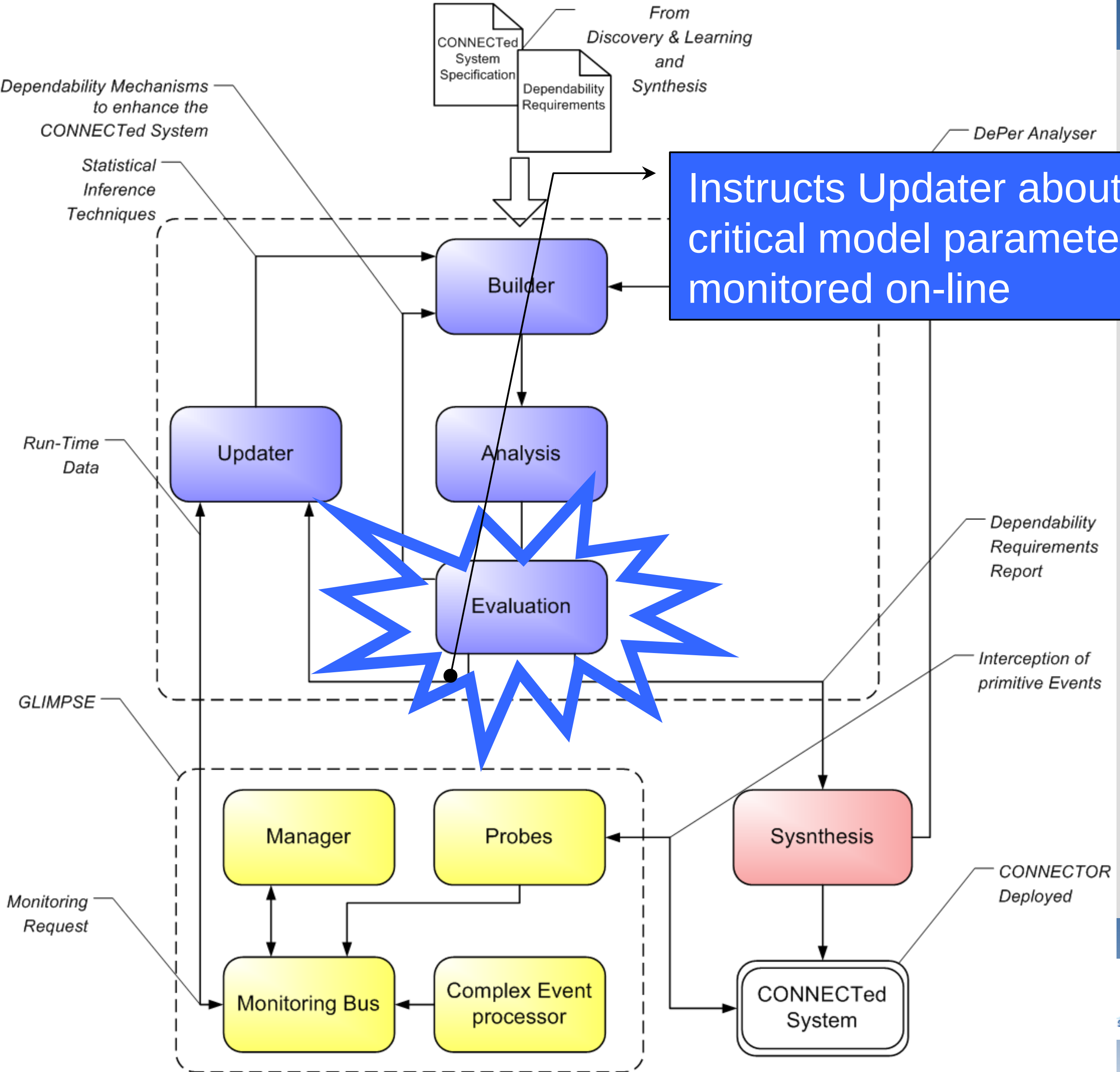
Interaction Pattern



Integrated
DePer + GLIMPSE
analysis

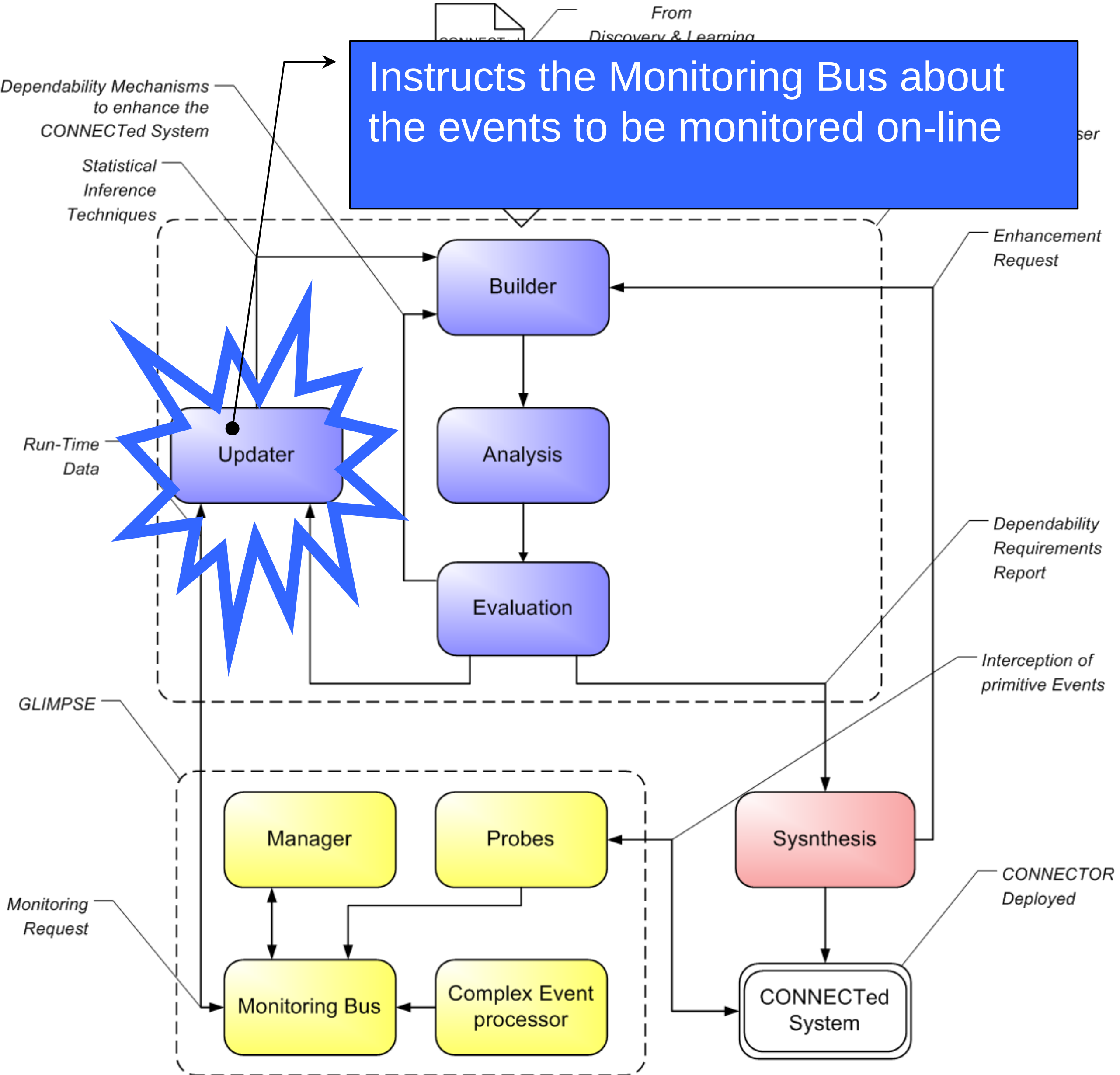

CONNECTed System Specification
Dependability Requirements
From Discovery & Learning and Synthesis
Dependability Mechanisms to enhance the CONNECTed System
Statistical Inference Techniques
DePer Analyser
Enhancement Request
Builder
Updater
Analysis
Evaluation
Run-Time Data
Dependability Requirements Report
Interception of primitive Events
GLIMPSE
Manager
Probes
Sysnthesis
CONNECTOR Deployed
Monitoring Request
Monitoring Bus
Complex Event processor
CONNECTed System

Synergy between DePer and GLIMPSE


CONNECTed System Specification
From Discovery & Learning and Synthesis
Dependability Requirements
Dependability Mechanisms to enhance the CONNECTed System
Statistical Inference Techniques
DePer Analyser
Builder
Updater
Analysis
Evaluation
Run-Time Data
Dependability Requirements Report
Interception of primitive Events
GLIMPSE
Manager
Probes
Sysnthesis
CONNECTOR Deployed
Monitoring Request
Monitoring Bus
Complex Event processor
CONNECTed System

Instructs Updater about the most critical model parameters, to be monitored on-line

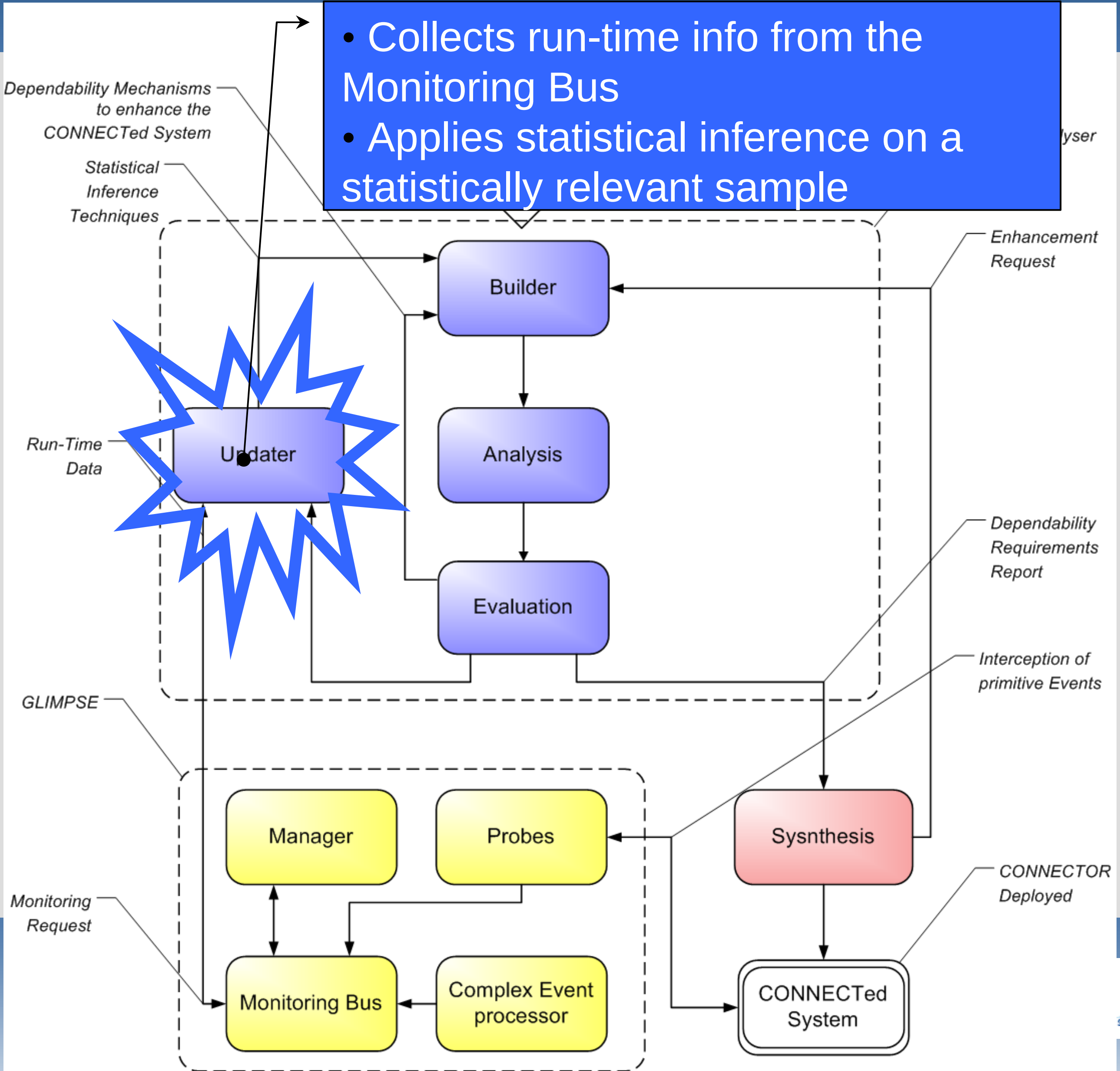
Synergy between DePer and GLIMPSE



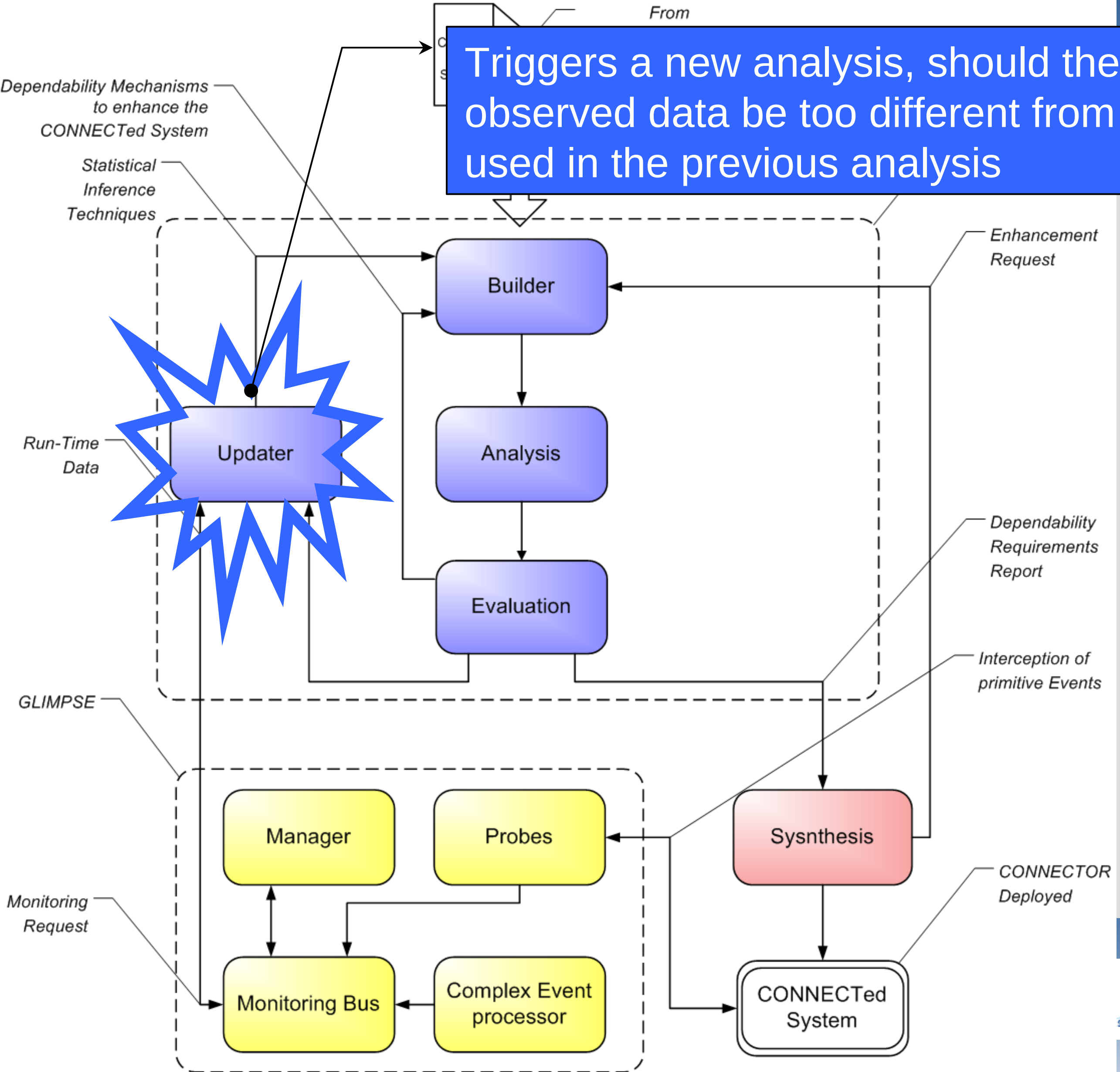
Instructs the Monitoring Bus about the events to be monitored on-line

Synergy between DePer and GLIMPSE

- Collects run-time info from the Monitoring Bus
- Applies statistical inference on a statistically relevant sample

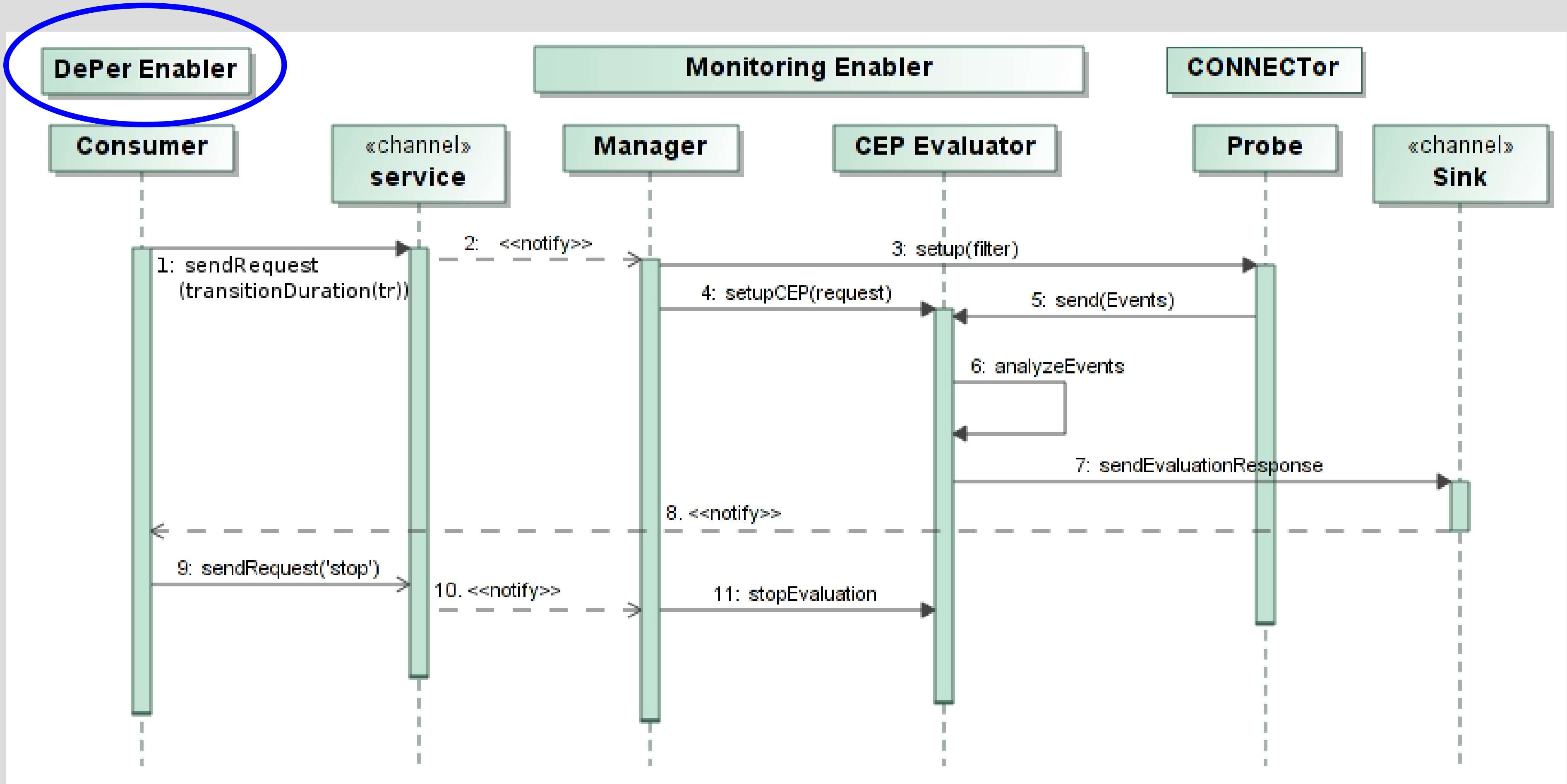


Analysis Refinement to account for inaccuracy/adaptation



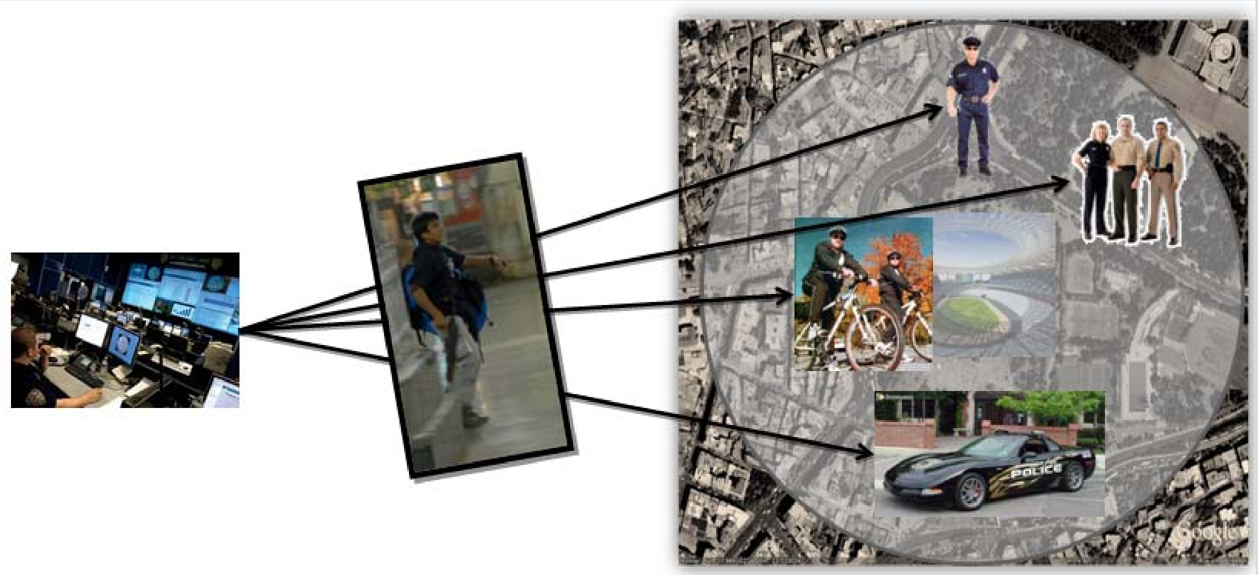
Triggers a new analysis, should the observed data be too different from those used in the previous analysis

Sequence Diagram of the basic interactions between DePer and GLIMPSE



Case Study

Case Study: The Terrorist Alert Scenario

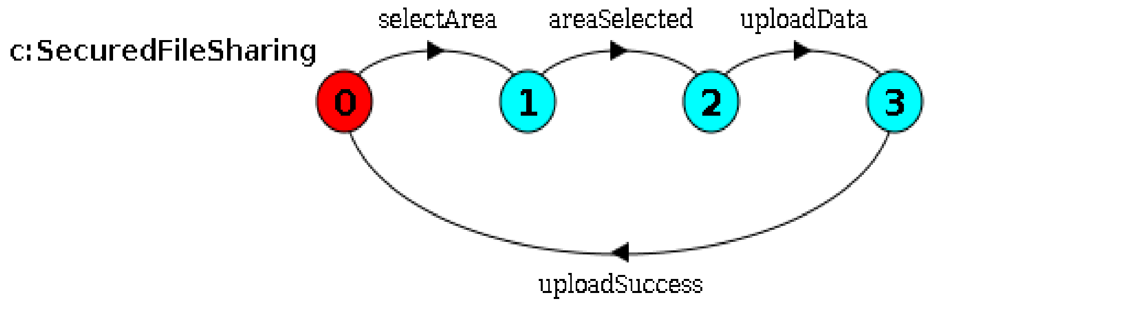


Alarm dispatched from policeman to civilian security guards, by distributing the photo of a suspect terrorist

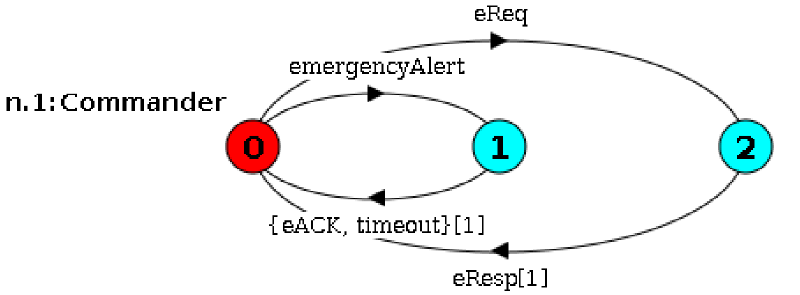
- CONNECT bridges between the police handheld device to the guards smart radio transmitters

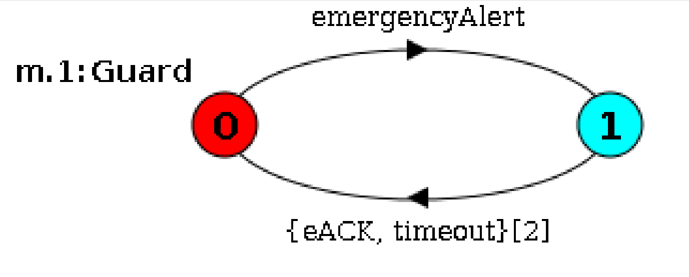
In more details…

- **NS1:** *SecuredFileSharing Application -* to receive msgs and documents between policemen and the police control center

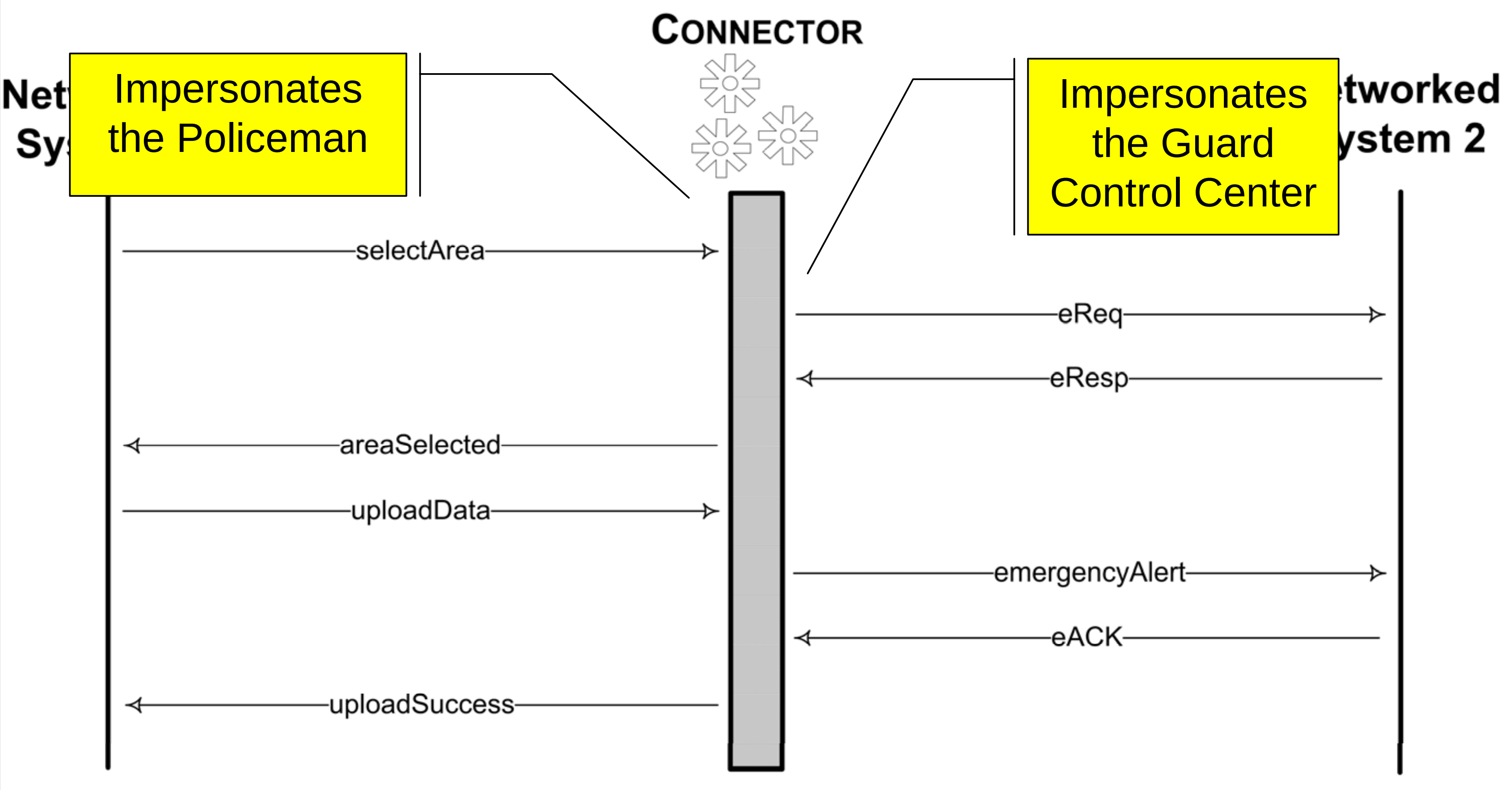


- **NS2:** *EmergencyCall Application -* 2 step protocol with first a request msg sent from the guard control center to the guards commander and successive alert msg to all the guards





Interoperability through CONNECT



Impersonates the Policeman

Impersonates the Guard Control Center

Examples of Dependability and Performance metrics

- **Dependability-related:** Coverage, e.g., the ratio between the # of guard devices (n) and the # of those sending back an ack after receiving the alert message, in a given time interval.

- **Performance-related:** Latency, e.g., the min/average/max time of reaching a set percentage of guard devices.

- For each metric of interest, it is provided:
- The *arithmetic expression* that describes how to compute the metric (in terms of transitions and states of the LTS specification)
- The corresponding *guarantee*, i.e. the boolean expression to be satisfied on the metric

Off-line Dependability and Performance Analysis

- Activation of the DePer Enabler
- Input:

  LTS of the Connected system + Metrics

- Transformation of LTS in SAN Model
- Transformation of Metrics in Reward Functions amenable to quantitative assessment
- Model solution through the MOBIUS Simulator
- Output:

  Result of comparison of the evaluated metrics with the requirements (*guarantees*) -> towards Synthesis

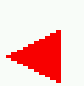
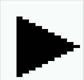
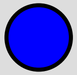
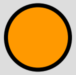
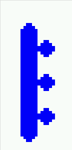
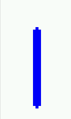
  Instruct the Monitor Enabler wrt properties to monitor on-line

  The Enhancer module is not considered in this case-study

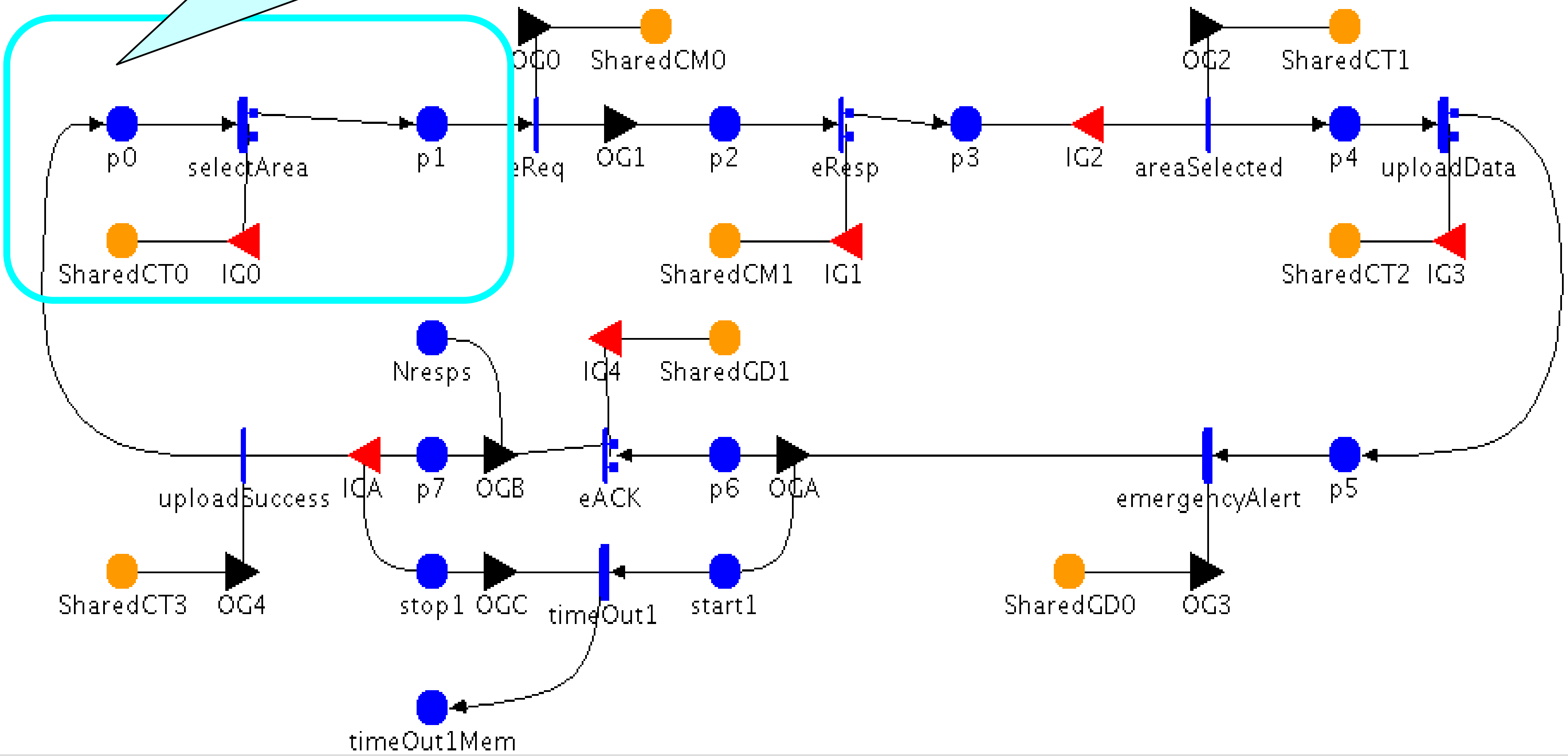
Stochastic Activity Networks

- *Stochastic activity networks (SAN)* are one extension to stochastic Petri Nets.
- SAN have the following properties:
  - A general way to specify that an activity (transition) is enabled
  - A general way to specify a completion (firing) rule
  - A way to represent zero-timed events
  - A way to represent probabilistic choices upon activity completion
  - State-dependent parameter values
  - General delay distributions on activities

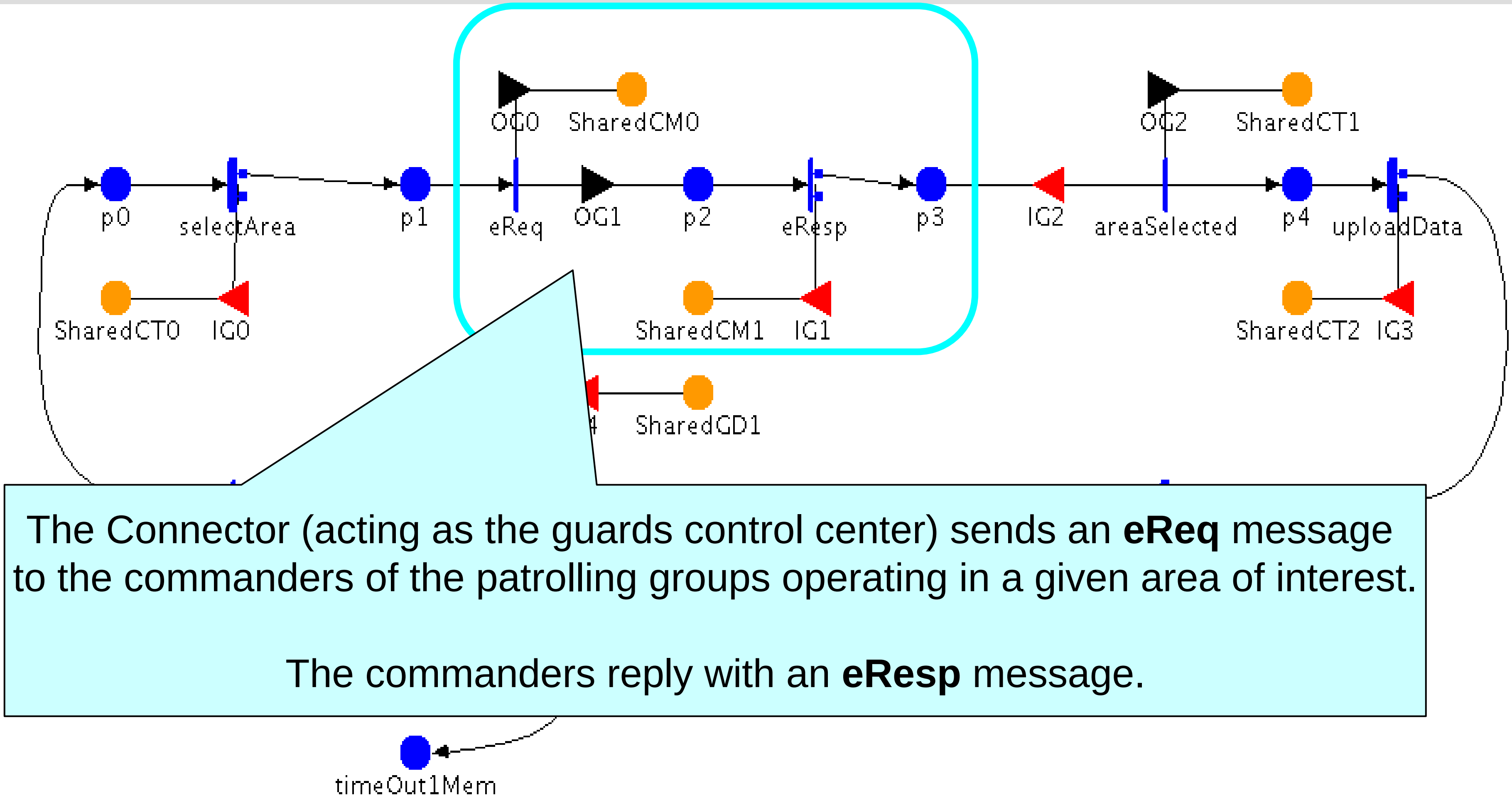
SAN Symbols

- SANs have four primitive objects:
  - Input gate:  used to define complex enabling predicates and changes of marking at activity completion
  - Output gate:  used to define complex completion functions
  - Places:   to represent the states of the system
  - Activities: timed  (with case probabilities) and instantaneous 

NS1 (Police control center) sends a **selectArea** message to NS2 (guards commander)
operating in a specified area of interest.



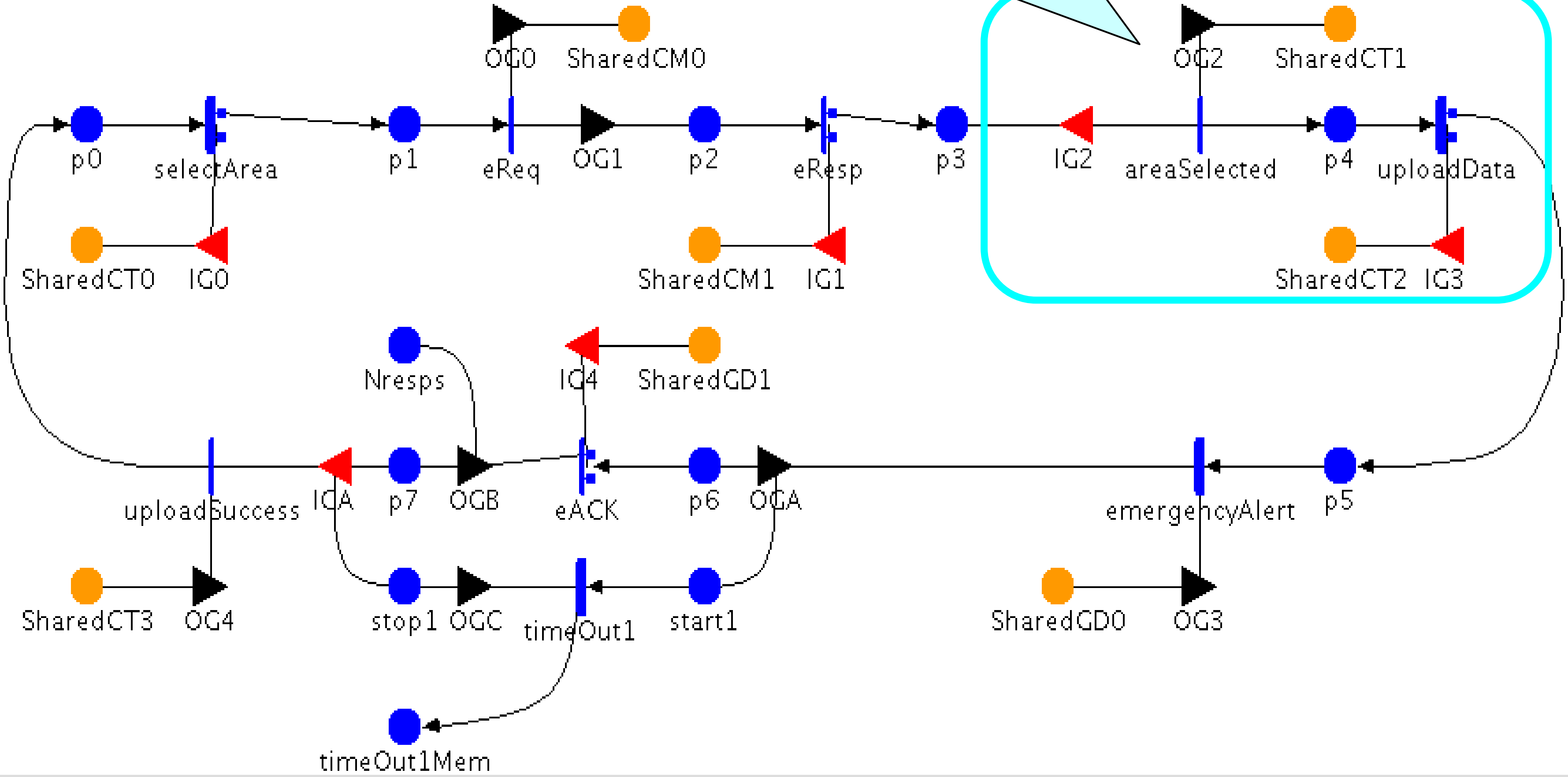
SAN of the CONNECTor



The Connector (acting as the guards control center) sends an **eReq** message to the commanders of the patrolling groups operating in a given area of interest.

The commanders reply with an **eResp** message.

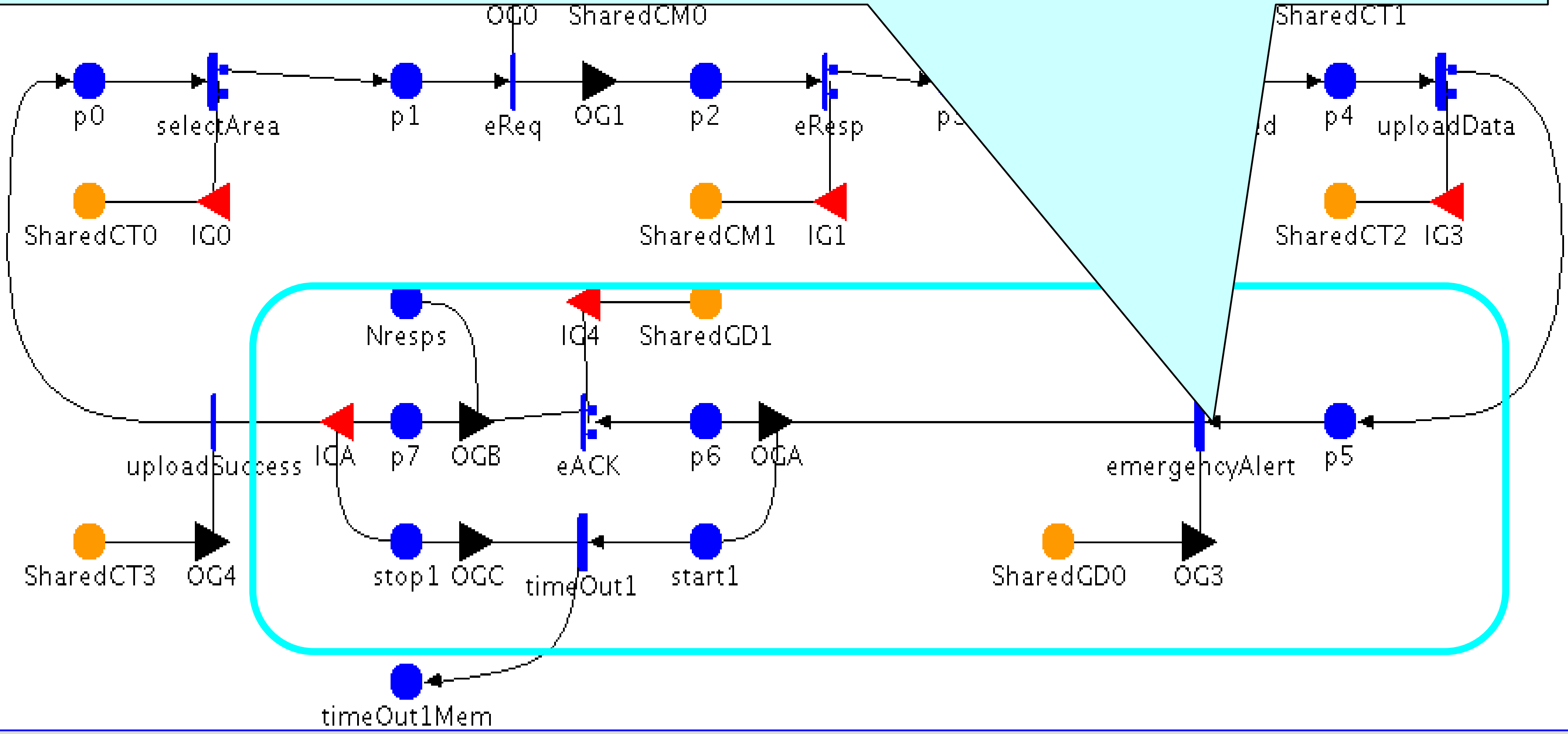
The selected commanders reply with an **eResp** msg, which is translated by the CONNECTor into an **areaSelected** msg.



The guards control center sends an **emergencyAlert** message to all guards of the commander's group.

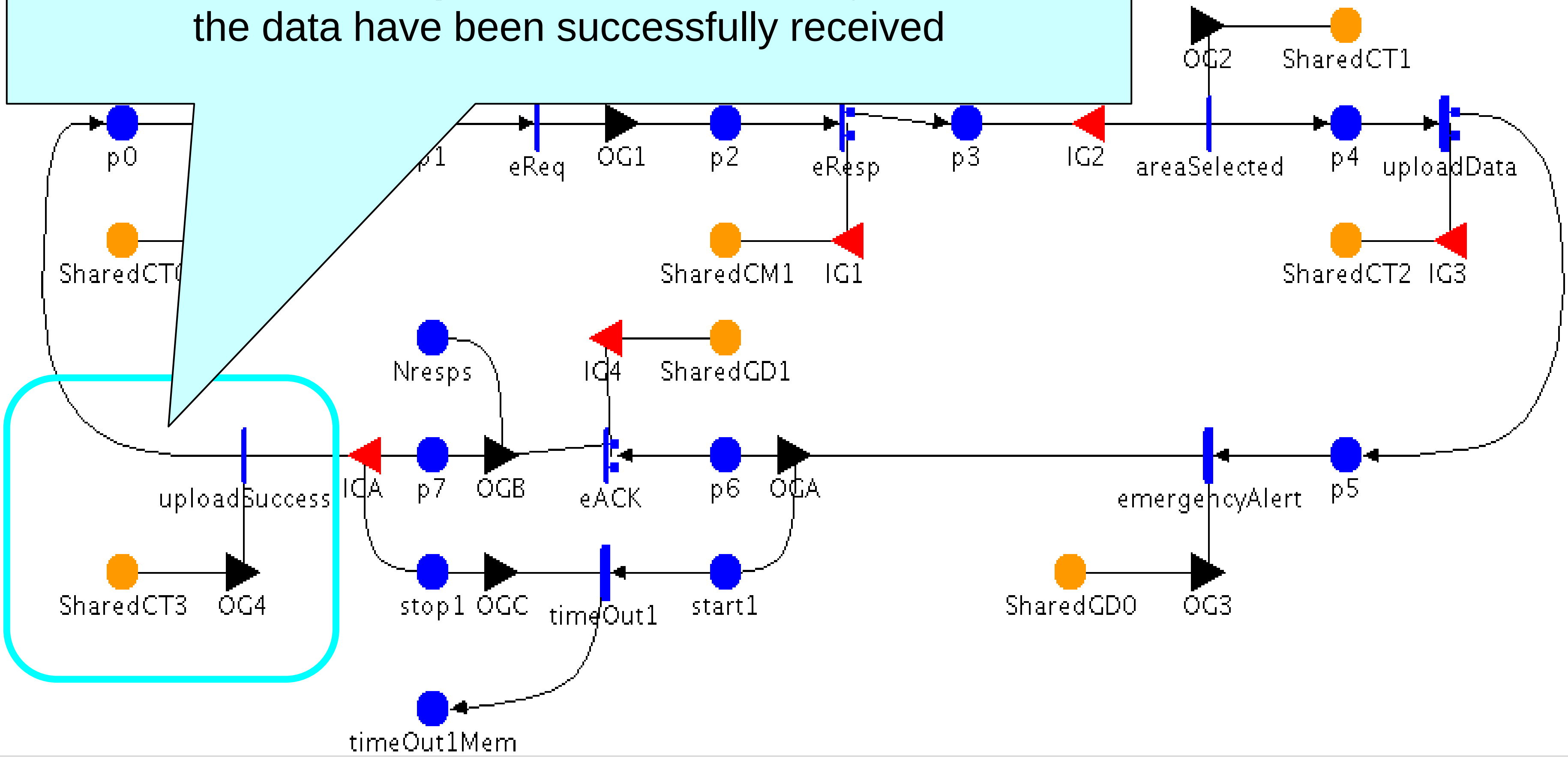
Each guard's device notifies the guards control center with an **eACK** message

The **timeout** represents the maximum time that the CONNECTor can wait for the **eACK** message from the guards.

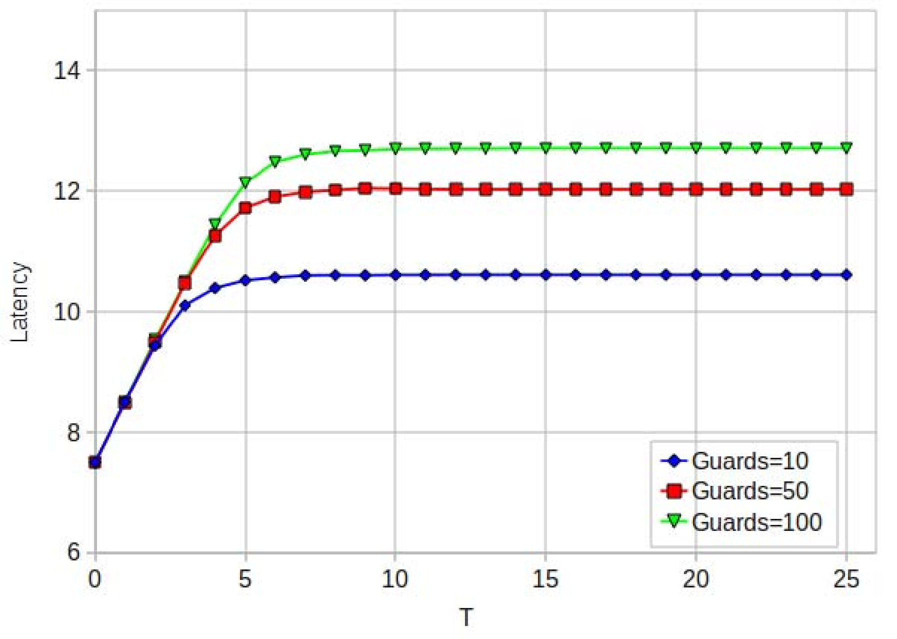


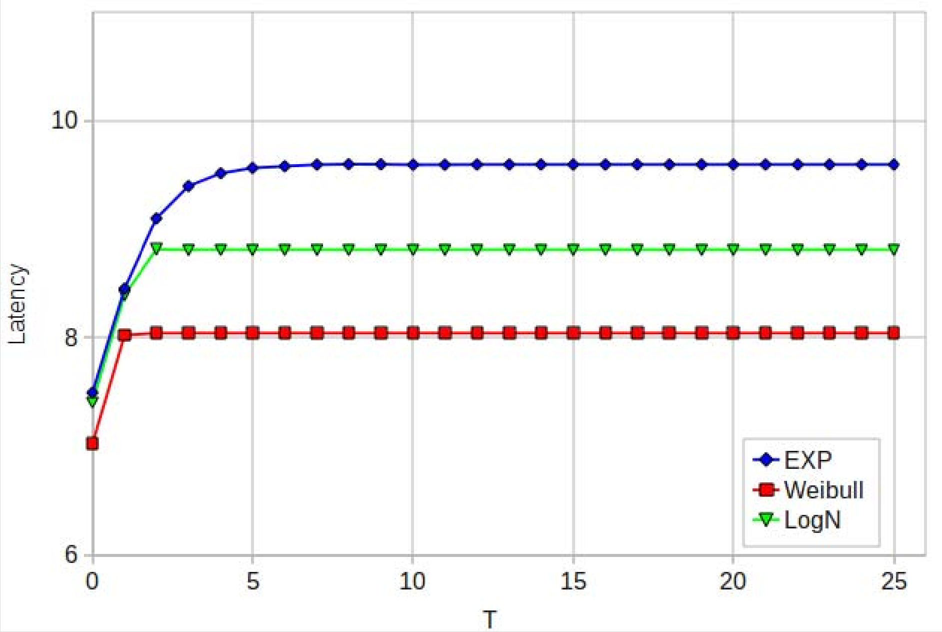
SAN of the CONNECTor

Each selected guard automatically notifies the police control center with an **uploadSuccess** message when the data have been successfully received



Latency

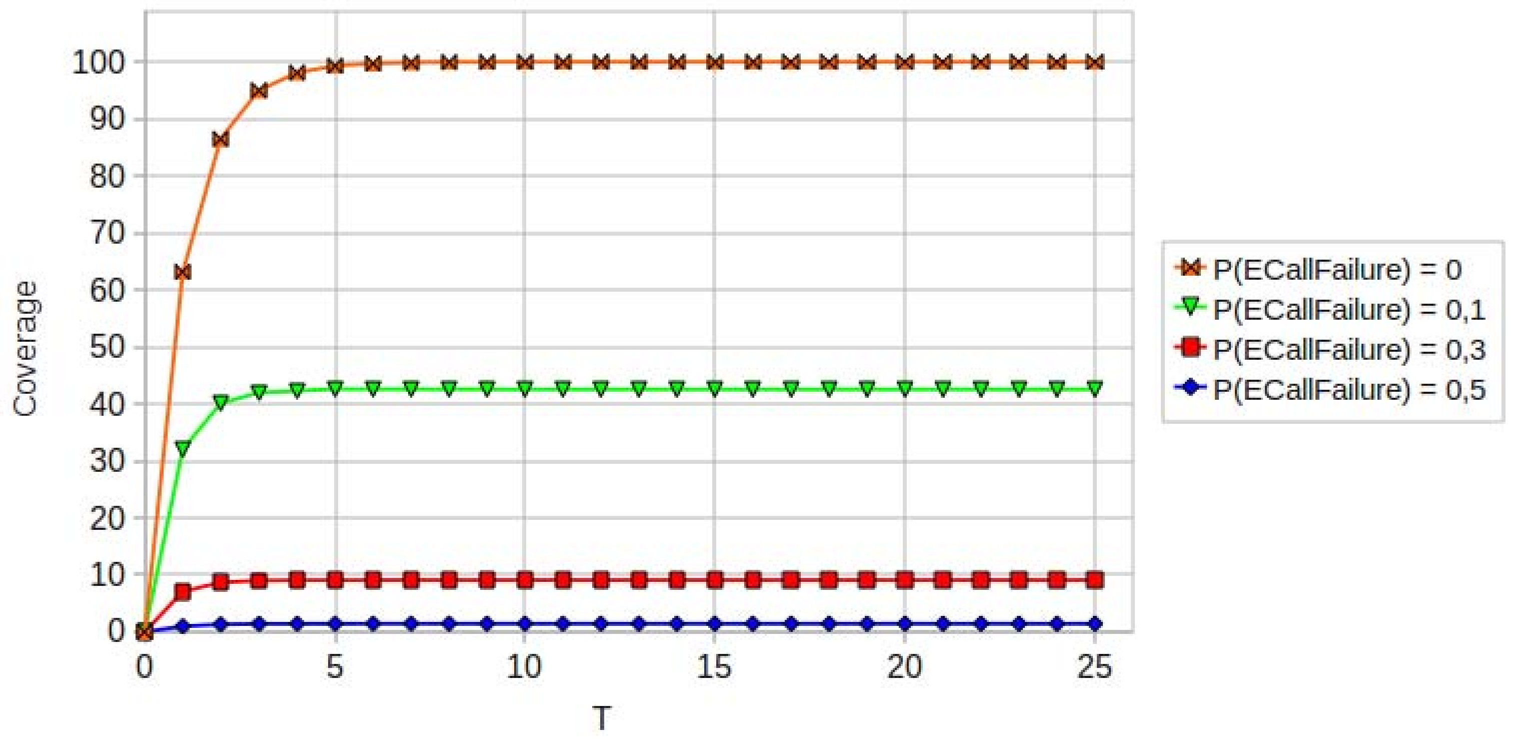




At increasing the number of guards

And for different traffic pattern

Coverage



For different omission failure probabilities of EmergencyCall communications