Deriving Architectural Specifications from KAOS Specifications: A Research Case Study

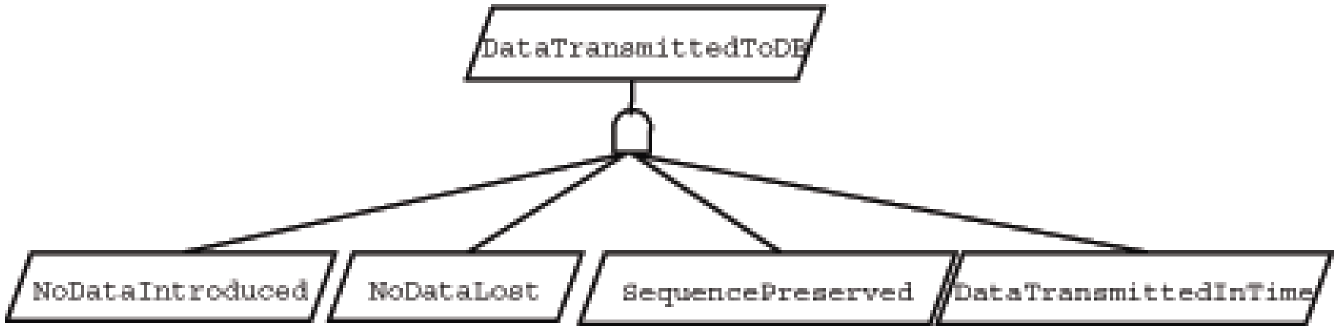
Divya Jani, Damien Vanderveken, and Dewayne E Perry
Empirical Software Engineering Laboratory
ECE, The University of Texas at Austin

Introduction

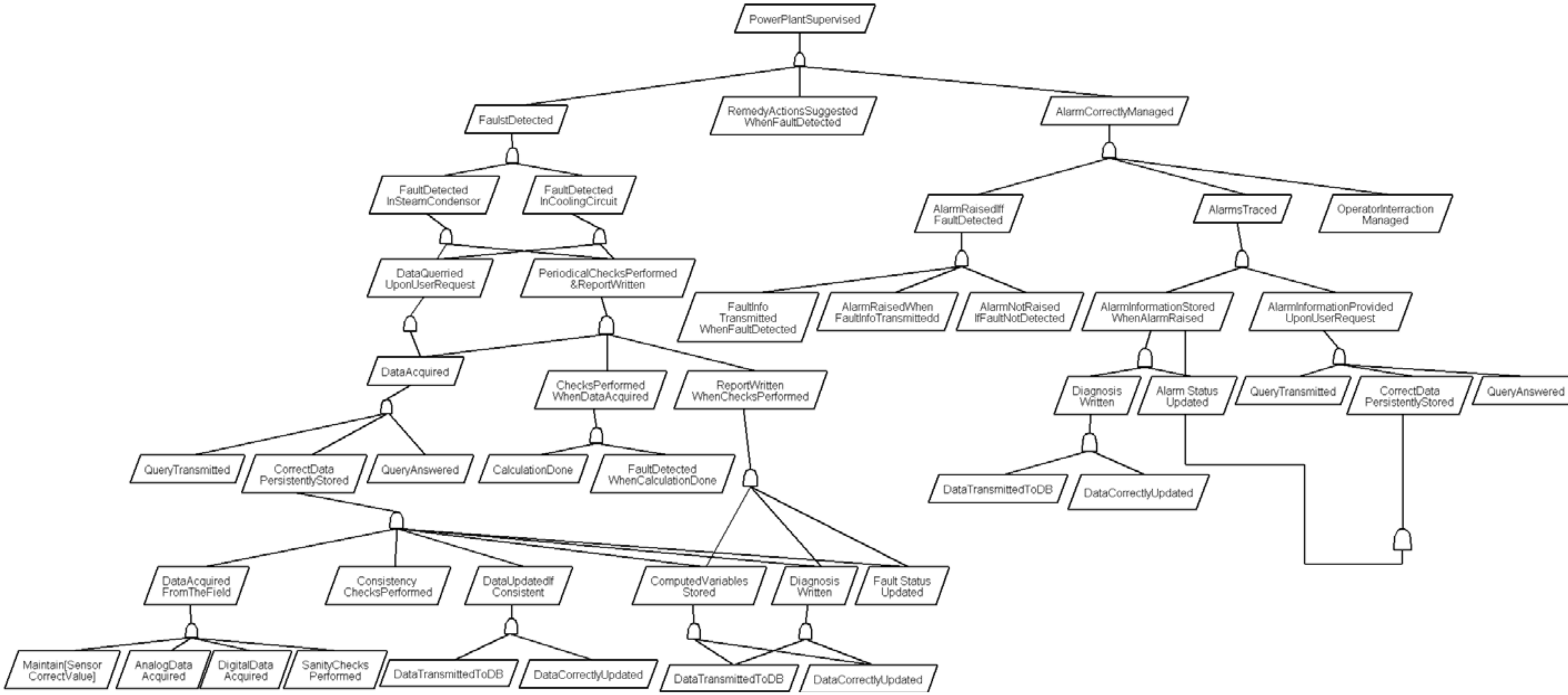
- History
  - ICSE 2000, Limerick – Axel van Lamsweerde's Keynote talk
  - Axel and I started talking about transforming R to A
  - Two independent threads of research
    - Perry/Brandozzi – 2002/2003
    - Van Lamsweerde - 2003
- Case study background
  - Divya Jani – my MS student
  - Damien Vanderveken – Axel's student, visiting for a semester
- Case study strategy:
  - Create a new KAOS goal-oriented requirements specification
  - Two cases –
    - DJ and DV as method users; DEP as oracle and observer
    - Use the van Lamsweerde method to create an architecture
    - Use the Brandozzi/Perry method to create an architecture
  - Compare the two methods and resulting architectures

Power Plant Specification

- Based on Trio based design specifications
- Created KAOS specifications
  - But paper descriptions were incomplete
  - Extended it in terms of non-functional characteristics
- KAOS Goal Directed Requirements Specification
  - Goal Model
    - Goals from TRIO Spec → informal → temporal 1st order logic
    - Refinement patterns to expand the specification (eg, milestone)
    - Iterative until reach leaf goals
    - Robustness goals added: eg, DataTransmissionToDB



Power Plant Specification – Goal Model

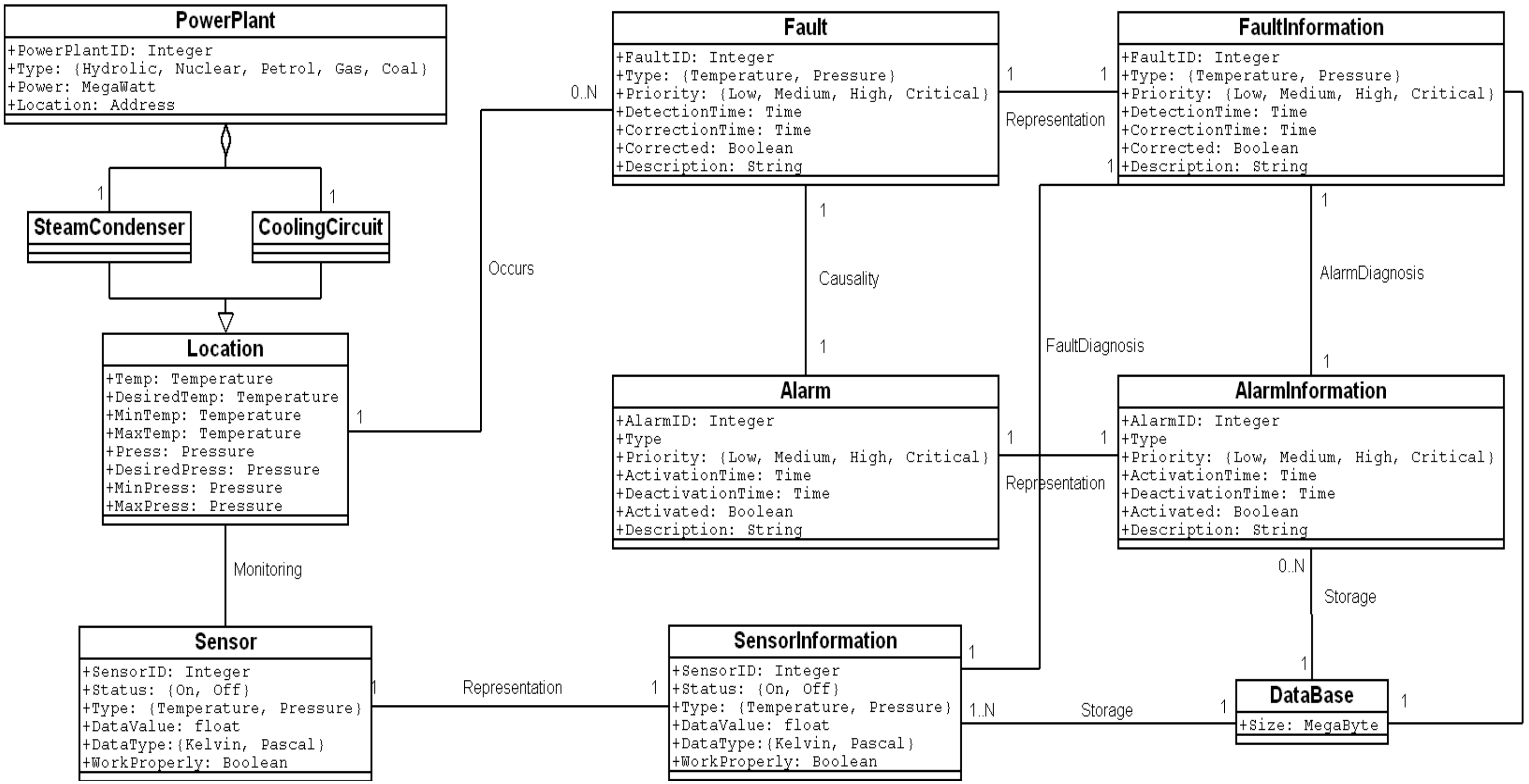


- PowerPlantSupervised :: FaultsDetected & RemedyActionsWhenFaultsDetected & AlarmsCorrectlyManaged
- FaultsDetected :: FaultDetectedInSteamCondensor & FaultsDetectedInCoolingCircuit
- AlarmsCorrectlyManaged :: AlarmsRaisedIffFaultDetected & AlarmTraced & . . .

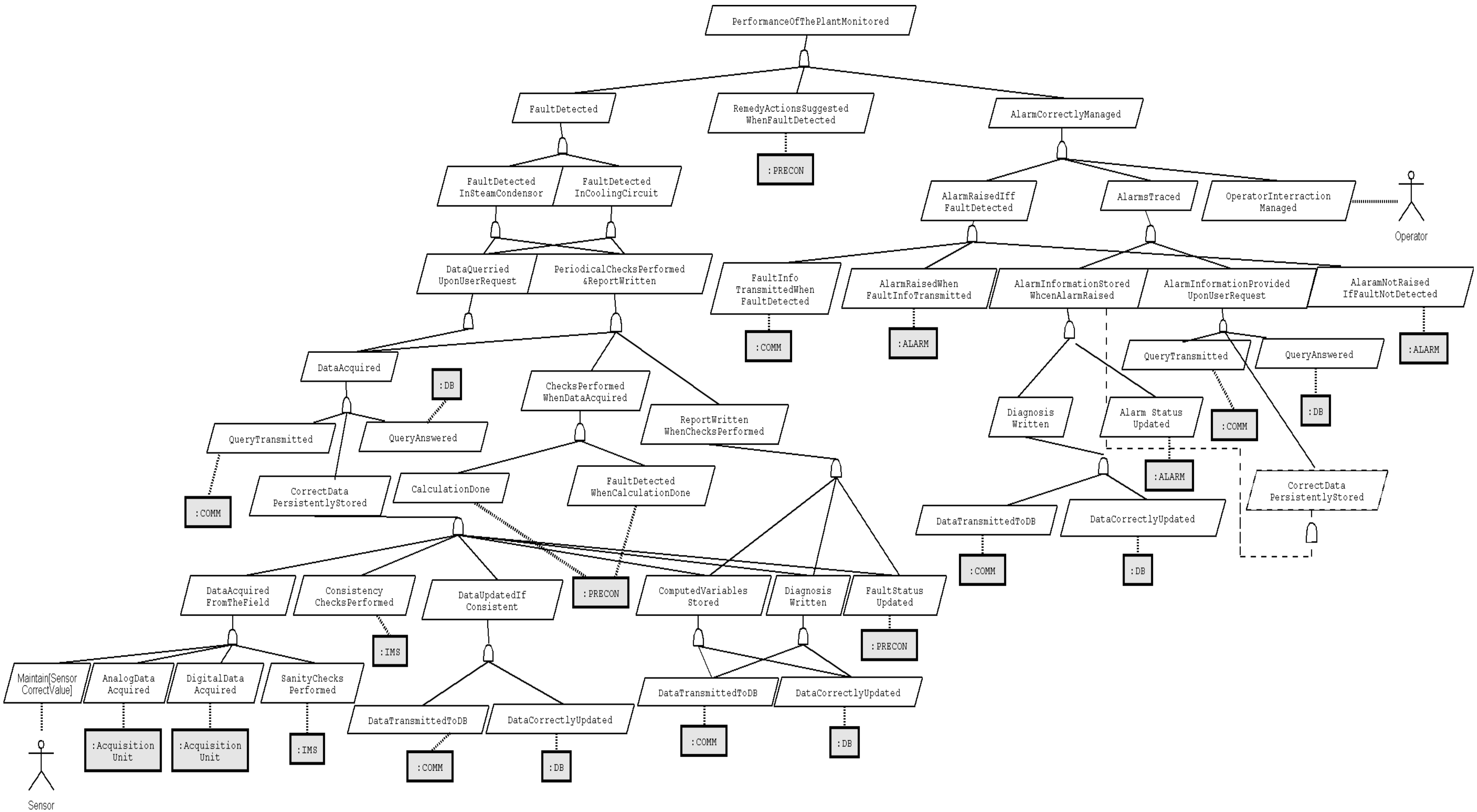
Power Plant Specification

- **Object Model**
  - Entities derived from TRIO spec
  - Attributes to characterize them
    - ✓ Some from spec
    - ✓ Most from underlying domain
    - ✓ Some added for a more complete model
  - 3 main objects: sensor, fault and alarm
- **Agent Model**
  - Each leaf in goal model assigned an agent
  - From TRIO Spec: precon, alarm, comm, db and sensor
  - Added: management unit, checks sensors for working properly
  - Differentiate: part of software to be & part of environment
    - ✓ Precon in former; sensor in latter
- **Operation Model**
  - Relies on precise definition of goals
  - Operation: pre-, trigger- and post-conditions
  - Operationalization patterns:
    - ✓ Bounded achieve
    - ✓ Immediate achieve

Power Plant Specification – Object Model

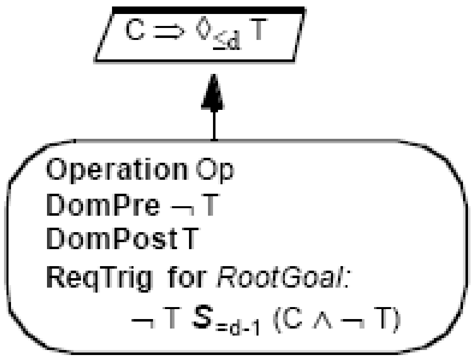


Power Plant Specification – Agent Model

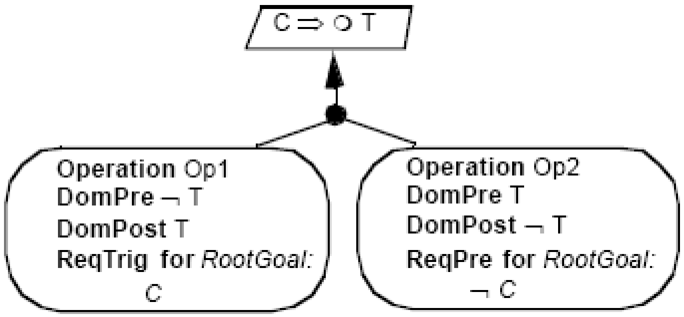


Power Plant Specification – Object Model

- Bounded achieve



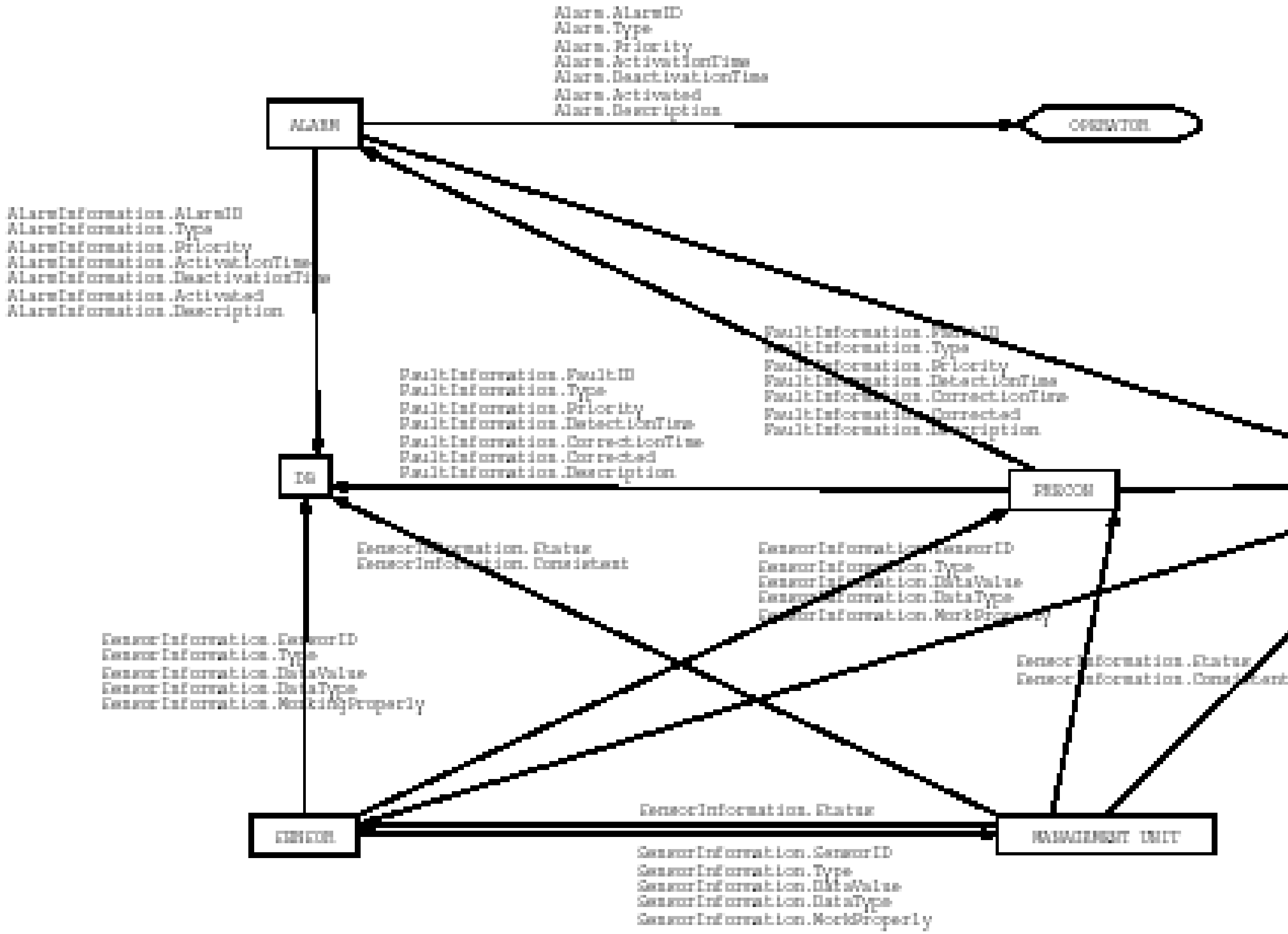
- Immediate achieve



Van Lamsweerde Method

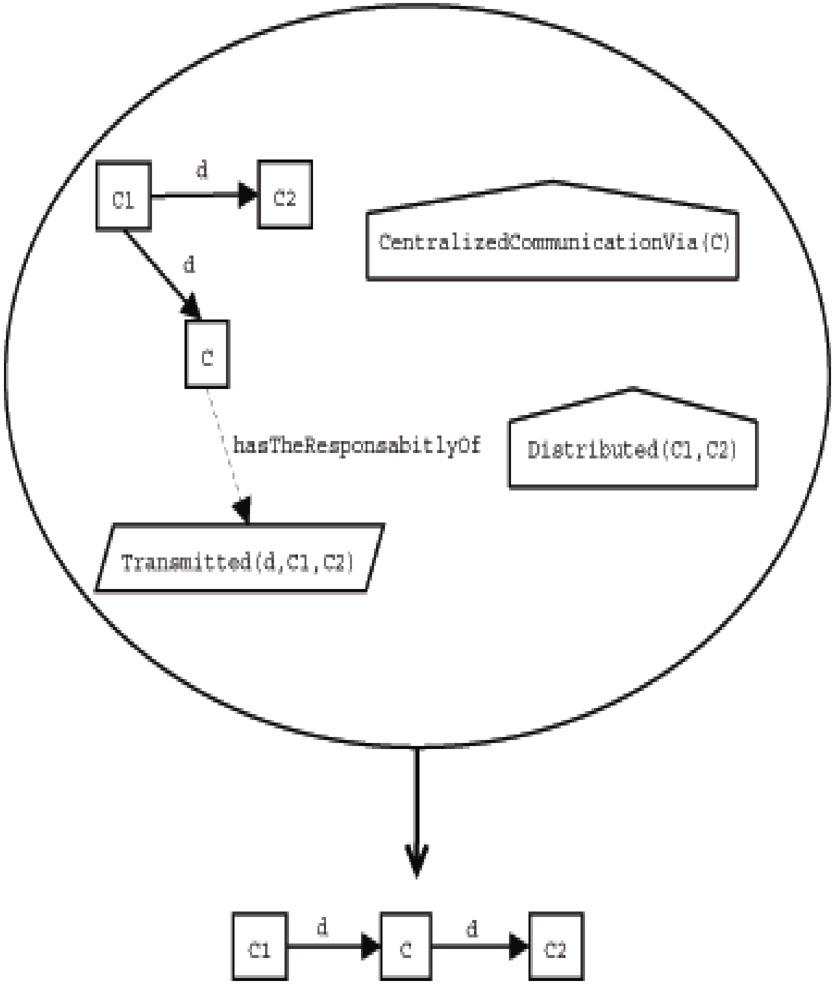
- 3 steps: Requirements to Architecture Description
  - Abstract a dataflow architecture
  - Drive and refine the data flow using styles to meet architectural constraints
  - Refine using design patterns to achieve non-functional requirements
- Step 1: Data Flow Architecture
  - Obtained from data dependencies between agents
  - Two sub-steps:
    - Agents become software components
    - Data dependencies modeled via dataflow connectors
  - Problem:
    - Dataflow connector between PRECON and ALARM
    - But really goes though COMM and DB

Van Lamsweerde Method: DF Architecture

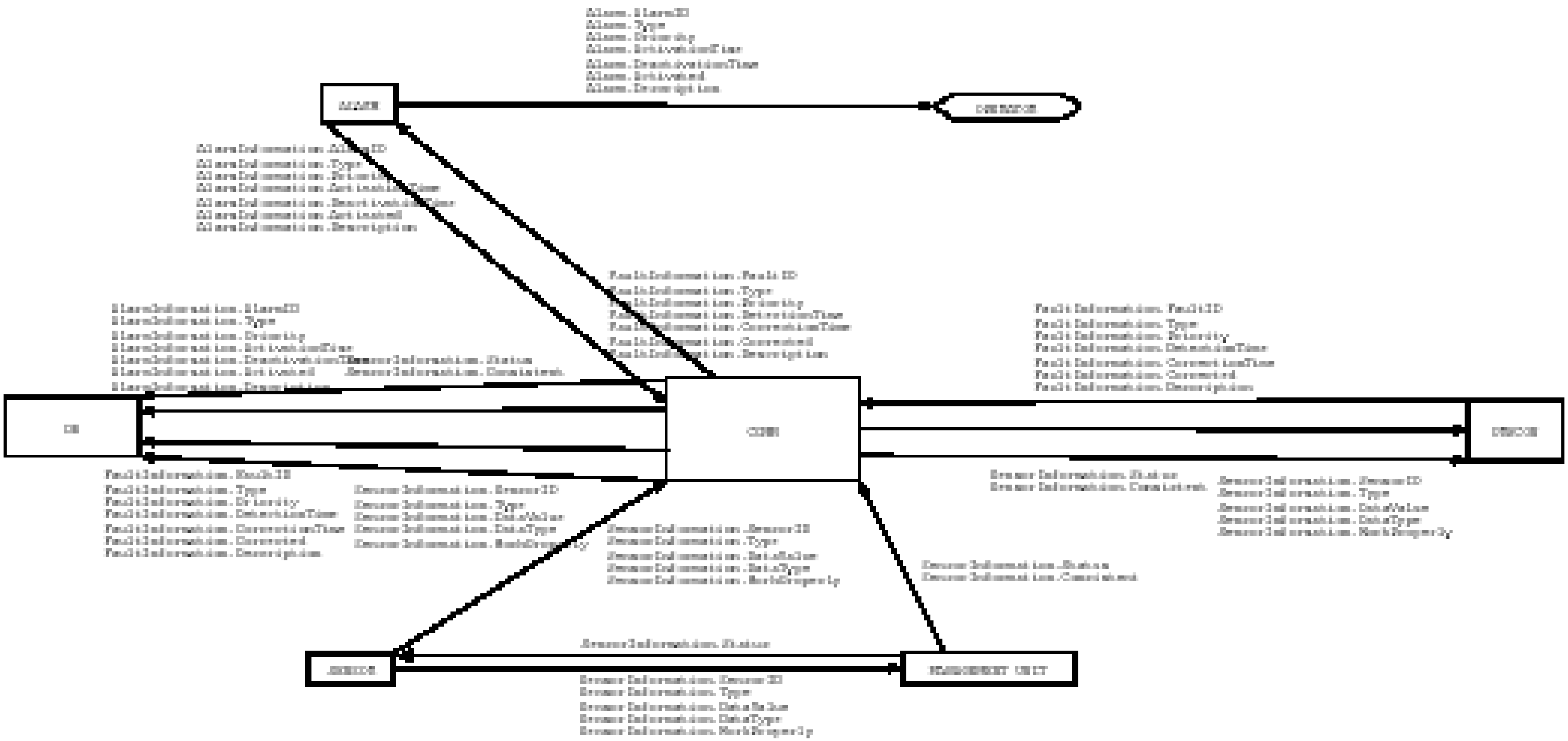


Van Lamsweerde Method

- Step 2: Style Based Refinement
  - Results of step one refined with a suitable style
  - Main architectural constraints:
    - Distributed components
    - Centralized communication
  - No appropriate style transformation rule; created one



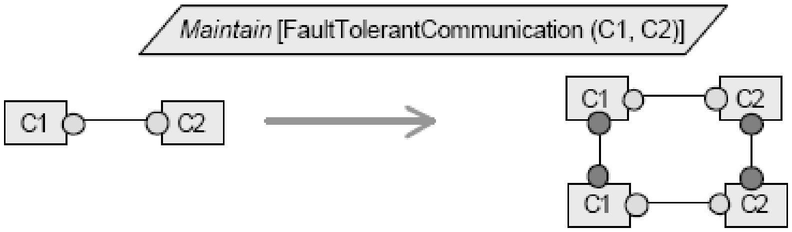
Van Lamsweerde Method: Style-Based R'ment



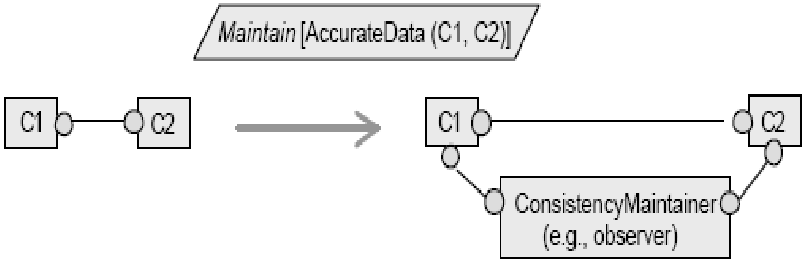
Van Lamsweerde Method

- Step 3: Pattern Based Refinement
  - Refine to achieve non-functional goals
    - Quality of service goals
    - Development goals
  - QOS
    - Security
    - Accuracy
    - Usability
    - Etc
  - Development goals
    - Minimal coupling
    - Maximum cohesion
    - Reusability
    - Etc

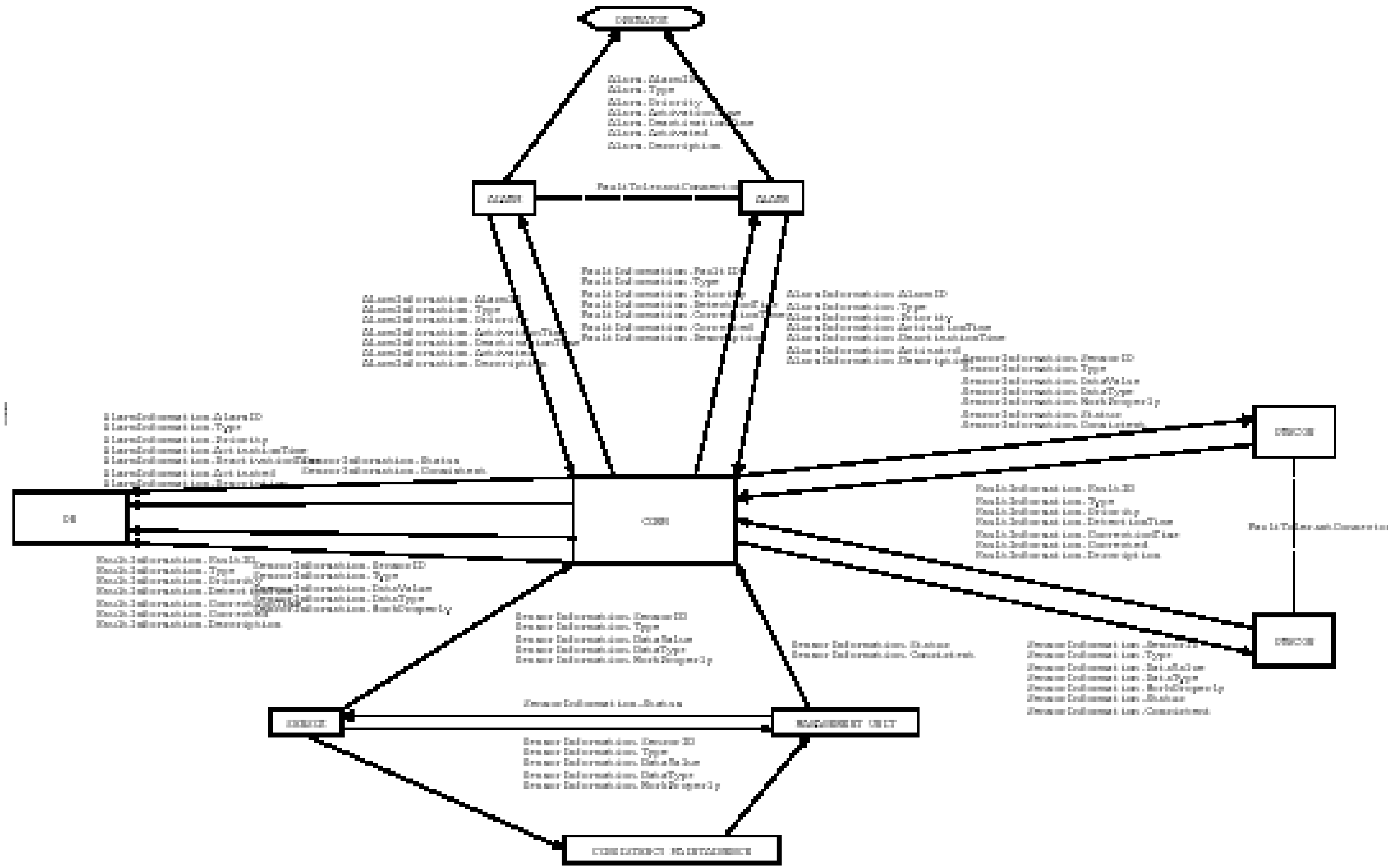
- Fault tolerant refinement



- Consistency maintainer refinement



Van Lamsweerde Method: Pattern Based R'ment



Perry Method

- Prescriptions based on Perry/Wolf Model
  - Element types: process, data & connector
- Maps KAOS entities to architecture entities
  - Agent → process or connector
  - Event → [connector]
  - Entity → data
  - Relationship → data
  - Goal → constraint
- 5 Steps: Requirements to Architecture Prescription
  - Step 1: Choose initial architecture component structure
  - Step 2: derive sub-components
  - Step 3: Partition system goals and assign to components
  - Step 4: Achieve non-functional goals
  - [Step 5: Create box diagram]

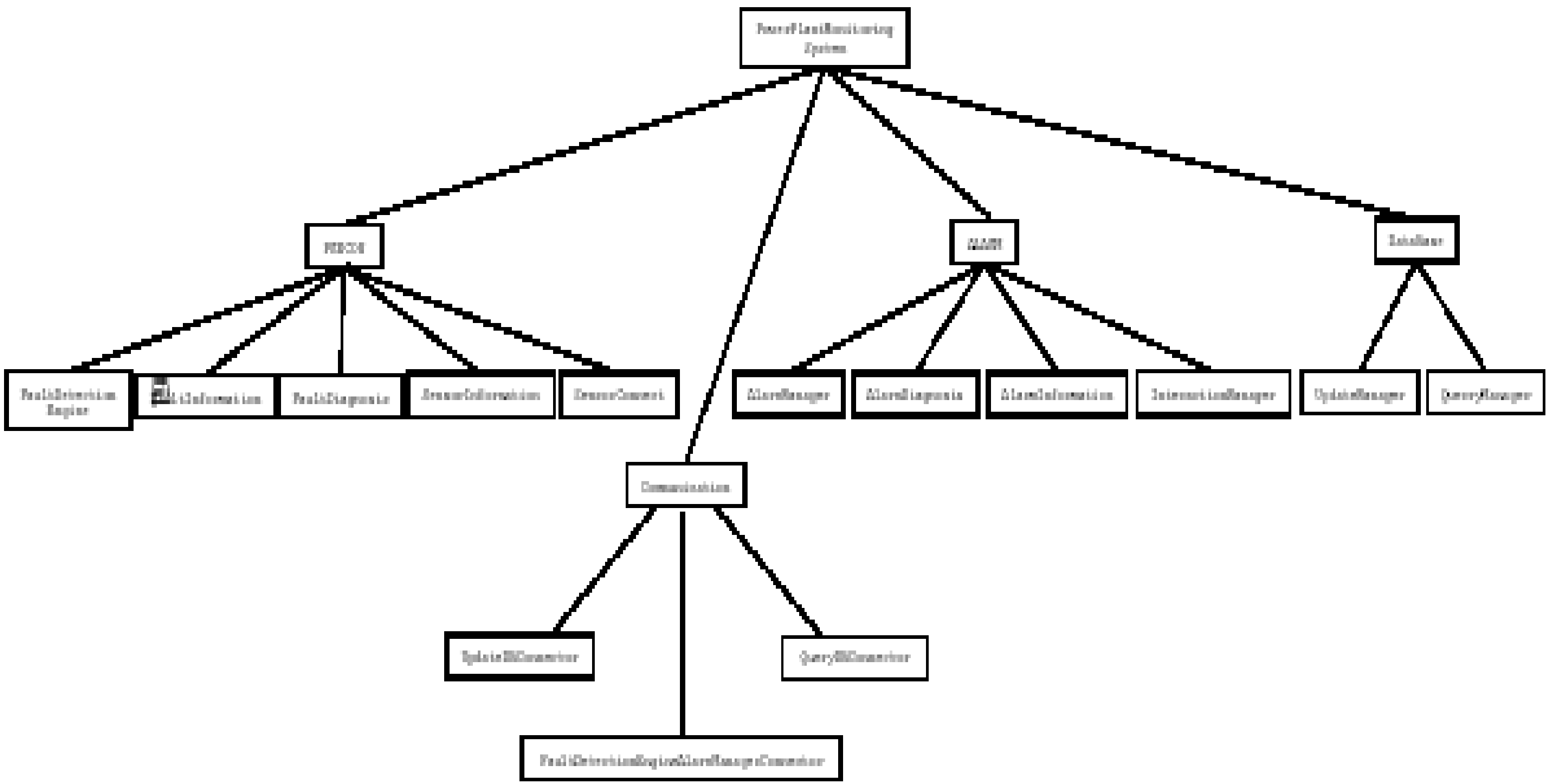
Perry Method

- Step 1: Choose initial architecture component structure
  - Using the goal refinement tree, select appropriate elements
    - Choose top goal:
      - ✓Probably too vague
    - Choose leaves
      - ✓Probably too constrained
      - ✓Architecture structure dictated by the requirements structure
    - Based on experience in the problem/solution domains
  - Problem: hard to know where to start – creative decision
  - Chose PRECON, ALARM, DB and COMM as components
- Step 2: Derive sub-components
  - Derive components from the KAOS Spec to implement these components
  - Examples: Fault (data), FaultInformation (data), SensorConnect (connector) and QUERYManager (process)
  - Continue to derive process and connector elements

Perry Method

- Step 3: Partition system goals and assign to components
  - Assign goals and sub-goals to the defined components
  - Depends on how the architect intends to realize the system
    - Again, a creative decision rather than a methodical one
  - All KAOS goals and/or sub-goals must be accounted for
  - Elements with no constraints are discarded
    - Eg, fault was discarded since it was not needed for any goal
  - COMM handled all communication – too broad
    - UpdateDBConnect
      - ✓ Secure and 2s response time
    - FaultDetectionEngineAlarmManagerConnect
      - ✓ 5s response time
    - QueryDBConnect
      - ✓ Fault tolerant, secure and 1s response time

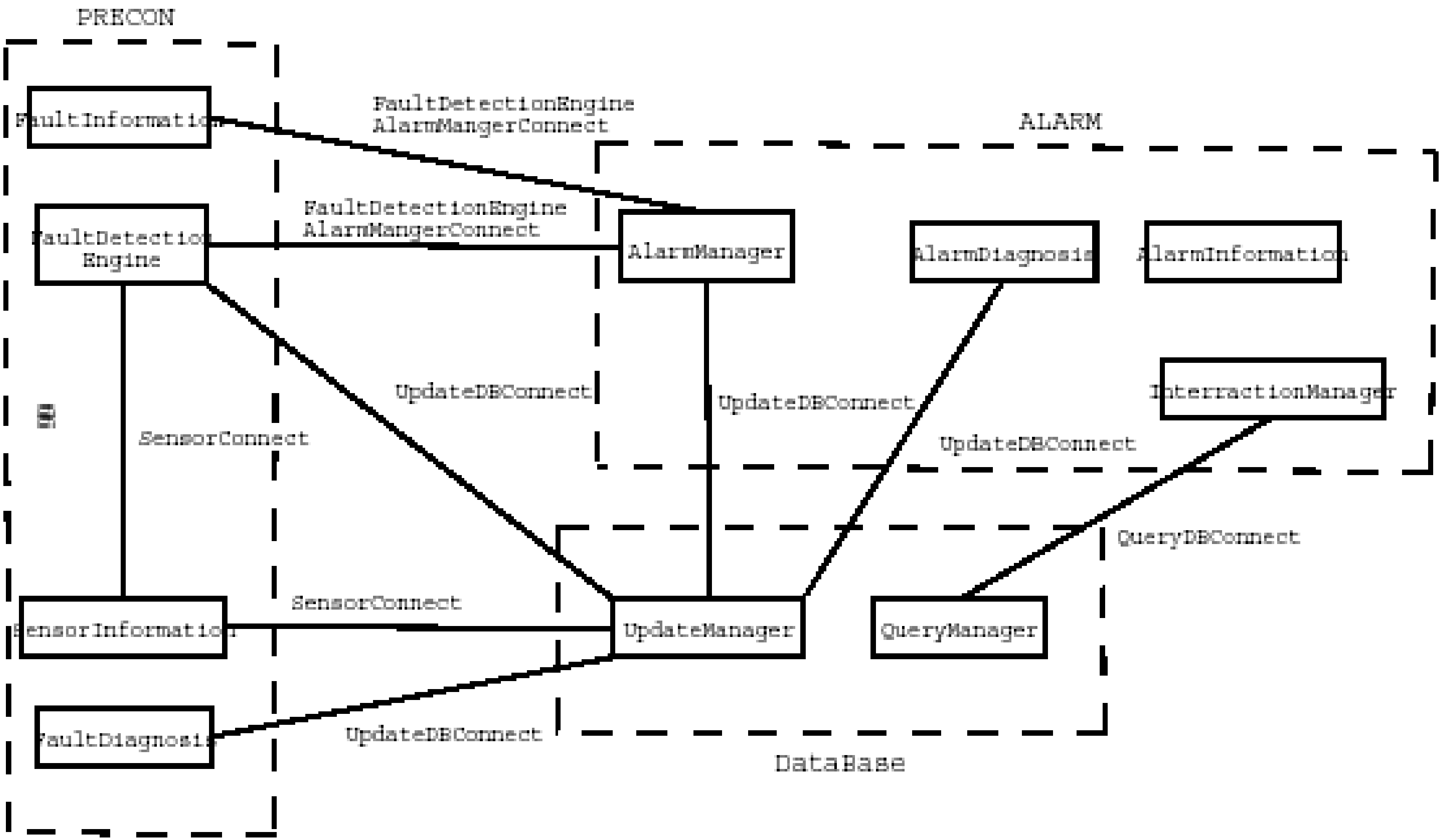
Perry Method: Component Refinement Tree



Perry Method

- Step 4: Achieve non-functional goals
  - Refine and transform the prescription
  - Goals such as reliability, reusability, etc
  - Introduced additional components and constraints
    - Connector between ALARM and PRECON
    - Redundant DBs for fault tolerance
      - ✓ Further constraints on connector and elements
    - Redundant PRECON and ALARM
- [Step 5: Create box diagram]
  - Needed to provide a graphical view of the system

Perry Method: Box Diagram



Evaluation

- Common
  - Neither has the means of addressing as architectural constraints: reliability, fault tolerance, etc
    - Architectures are derived only from goals
    - Non-functional requirements may arise for architectural reasons
  - Incomplete requirements
    - Eg, nothing about performance
- Van Lamsweerde method
  - Easy to get started, harder to finish
    - Step 1 proceeded well
    - But few styles to use in step 2
    - Step 3 had pattern application problems
      - ✓ Limited choice of patterns
      - ✓ Some cases required multiple patterns – difficult to decide how to do it
  - Problem when introducing new components
    - New components, no operations defined
    - New connectors without complete definitions

Evaluation

- Van Lamsweerde method (continuted)
  - Problem in insuring consistency in redundant components
    - Method of communication between redundant components
    - Affect on the connector used to the components
  - Communication as a component was a problem
    - Communication among different components had different consistency, performance and reliability constraints
- Perry method
  - First hurdle was step 1 – a large degree of freedom
    - Lacked sufficient guidance
      - ✓ May be appropriate for an experienced architect
      - ✓ Difficult for a novice
    - Examples of goal trees and initial architecture would have helped
  - How much leeway to allow in each step
    - How free in distributing and allocating goals and subgoals
  - Component refinement tree indicates hierarchy
    - But box diagram makes it clear the architecture is a network
  - Need to add data as a constraint on connectors - critical

Comparison

- Level of design - most significant difference
  - Van Lamsweerde (vL) method produces a much lower level architecture - descriptive
    - Components + operations creates a much more rigid design
  - Perry (P) higher level - prescriptive rather than descriptive
    - Emphasis on constraints
- Basic view of architecture
  - vL produces a more 'network like' view
  - P appeared more hierarchical
    - For P, box diagram made network structure clearer
- Process
  - Getting Started
    - vL more systematic at beginning; less so later
    - P hard to get past the first step
  - Continuing and finishing
    - vL got more confusing
    - P became more manageable given the initial structure

Comparison

- Connectors
  - vL – focus on data but not constraints
  - P – focus on constraints but not data
- Non-functional requirments
  - vL - applied appropriate patterns
  - P – added constraints
- Overall
  - Both methods provided useful but different views of the system
- Subsequent work
  - Jani's MS Thesis:
    - added patterns for non-functional properties
    - Extended connector prescriptions
  - Vanderveken's MS Thesis (Co-supervised by AvL and DEP)
    - Added behavior view to architecture descriptions
    - Precise definitions and applications of transformation patterns

After Thoughts

- vL
  - RE driven approach
  - Initial structure dependent on RE structure
  - Transformations afterwards
- P
  - Architect driven approach
  - Creative integration of requirements drives initial structure
  - May integrate transformations into initial structure
  - Architecture constraints include requirements goals