Quality Criteria and an Analysis Framework for Self-Healing Systems

Prepared by Faisal Sibai for CS 895

Original paper "Neti, S. and Muller, H. A. 2007. Quality Criteria and an Analysis Framework for Self-Healing Systems. In *Proceedings of the 2007 international Workshop on Software Engineering For Adaptive and Self-Managing Systems* (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 6. DOI= http://dx.doi.org/10.1109/SEAMS.2007.15 "

Presentation Agenda

- Introduction
- Quality criteria for self-healing systems
- Architectural styles for self-healing systems
- Attribute-based architectural styles (ABAS)
- ABASs use in analysis
- Conclusion and future work

Introduction

- Main objective: "Evaluate and document software architecture of self-managing applications with respect to specific quality attributes."
- Identify key attributes and their changes and adaptation over long periods of time
- Technique used is based on specific individual quality attributes
- Self-healing is the main concentration of this paper
- Takes into consideration changes in their designed operating mode, accumulated component and resources, external environments…etc.
- Analysis and reasoning framework for the architecture Attributed – Based Architectural Style (ABAS).

Quality criteria for self-healing systems

- Main concentration is quality in this paper
- Both traditional qualities (reliability, modifiability or availability) and Autonomic-specific criteria's(dynamic adaptation support, dynamic upgrade support, diagnostics support…etc.) are explored.
- Used for evaluation of existing and new self-healing systems.
- Traditional quality attributes:
  - Reliability and maintainability are key traditional properties in any self healing systems
  - **Reliability** includes fault tolerance and robustness
  - **Maintainability** Includes modifiability and extensibility.
- Autonomic-specific quality attributes:
  - Self-healing systems have to predict and take action to prevent failures from having detrimental impact on applications
  - Challenge: Determination of the cause of the failure for specific element without a restart to the entire system!
  - Can be predicted by characterizing and detecting anomalous behavior of the system.

Quality criteria for self-healing systems

- Key autonomic-specific quality attributes:
  - ***Support for detecting anomalous system behavior***
    - Monitor, recognize, and address anomalies with respect to performance or quality of service of a subject system or its environment
    - Key points: awareness and observability
  - ***Support for failure diagnosis***
    - Ability to locate the source of misbehavior or failure, such as component failure, system degradation, or changes in system usage
    - Dependent on the complexity of the problem!
  - ***Support for simulation of expected or predicted behavior***
    - Ability to accurately model the system and thereby obtain the expected or predicted behavior of the executing system
    - Depends on the system's awareness, correctness, completeness, and consistency as well as the complexity of the model involved
  - ***Support for differencing between expected and actual behavior***
    - Ability to detect whether the actual behavior of the real system differs from its expected or predicted behavior.
    - Depends on the support for simulation of expected or predicted behavior quality
  - ***Support for testing of correct behavior***
    - Ability to test and verify that autonomic elements behave correctly

Quality criteria for self-healing systems

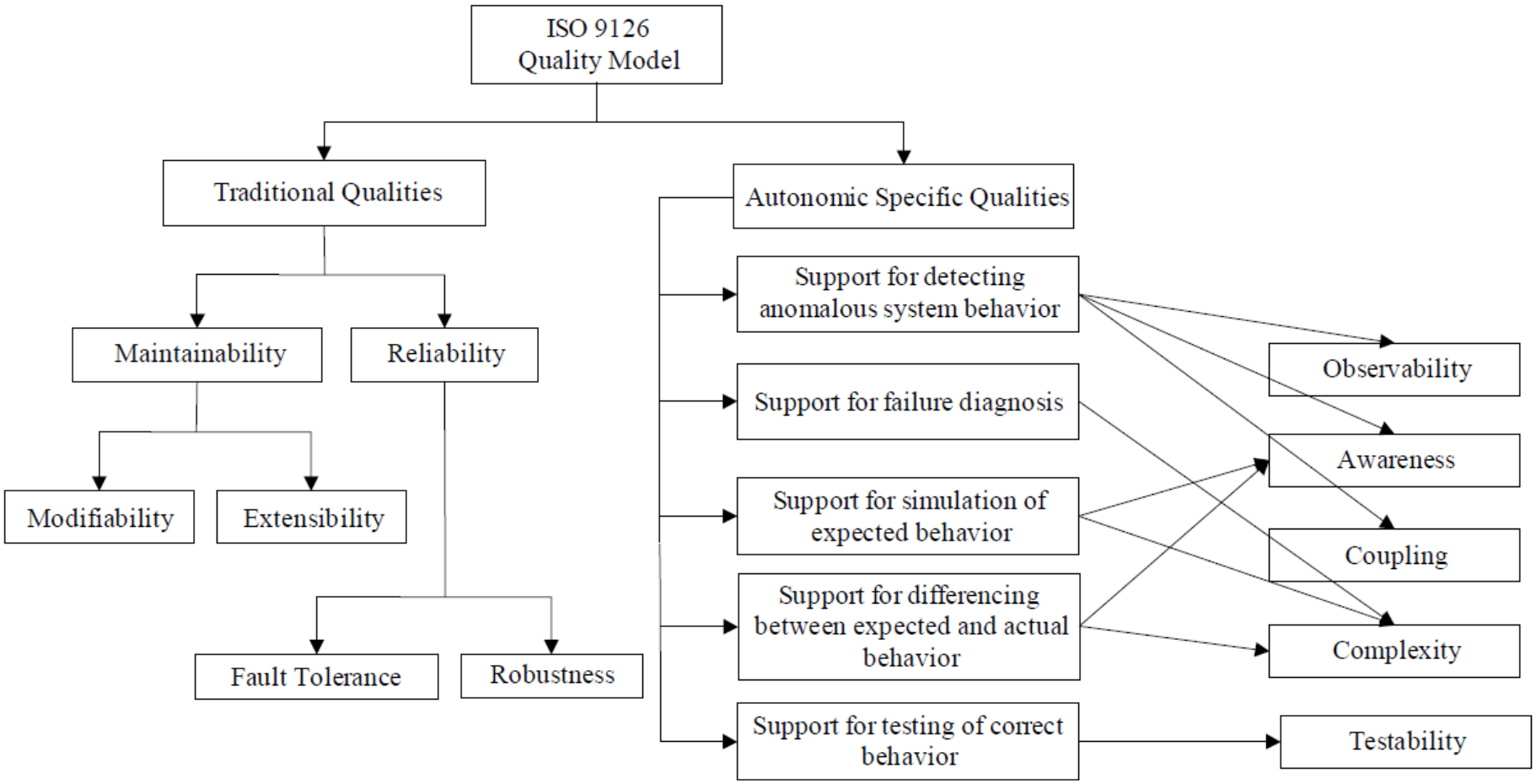

Figure 1. Quality model for self-healing systems based on ISO 9126

Architectural styles for self-healing systems

- Hawthorne and Perry: Basic requirements for the architecture of self-healing:
  - A **reflection** mechanism to detect internal or external conditions to which the system should respond to.
  - A **reasoning** mechanism to determine what actions to perform in response to input from the reflection mechanism.
  - A **configuration** mechanism to perform the necessary changes to repair or optimize the system as directed by the reasoning mechanism.

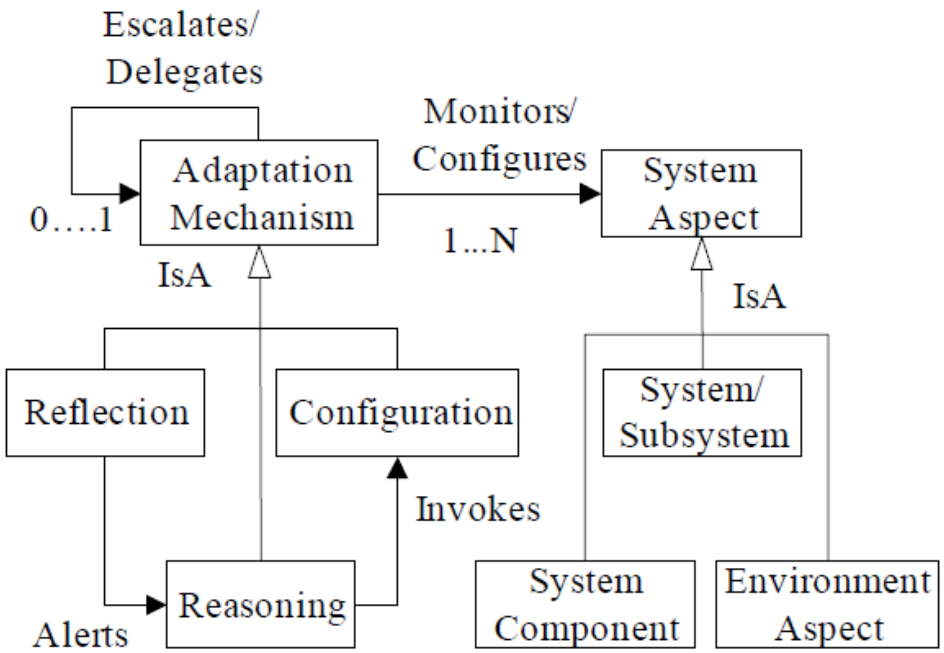

Figure 2. Conceptual architectural model

Architectural styles for self-healing systems

- Managed element constitutes a **system component or the system's environment** and the **adaptation mechanism** represents the autonomic manager.

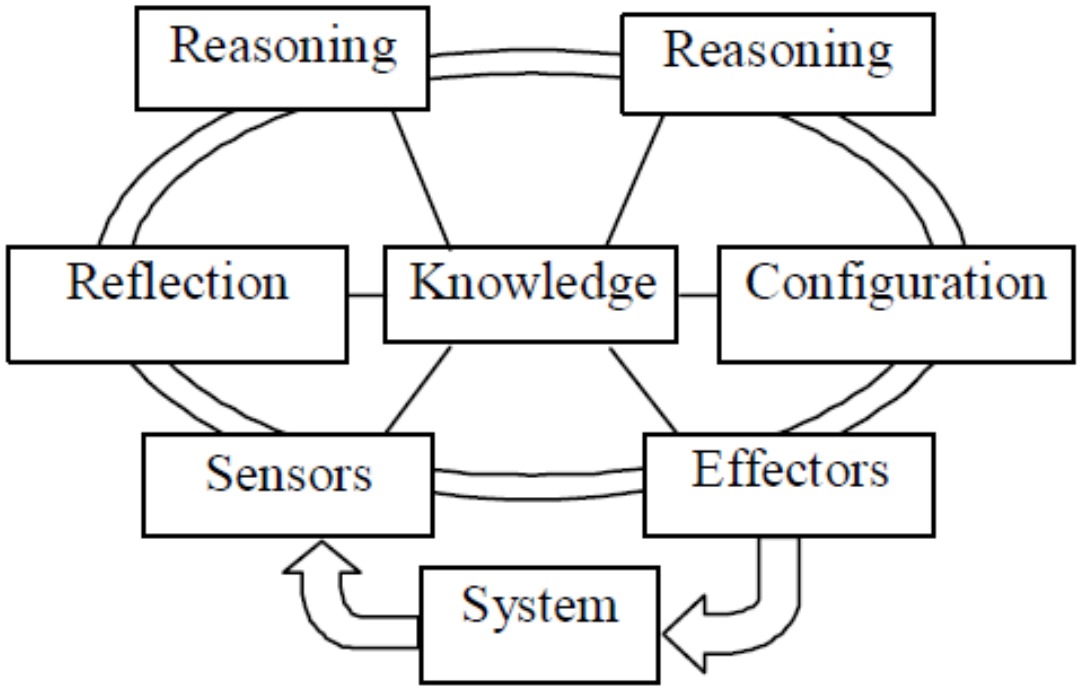

Figure 3. Conceptual model of a self-healing architecture mapped to an autonomic element

Architectural styles for self-healing systems

- Hawthorne and Perry present and discuss the following architectural styles for self-healing systems:
  - **Aspect peer-to-peer architectural style:**
    - Simple approach that consists of a monitor component to observe each aspect of the system or its environment and a peer configurator component to reconfigure the system
    - Viewed as a set of autonomic elements with one autonomic element for each component of the system
    - Each autonomic element in the set is independent and complete on its own.

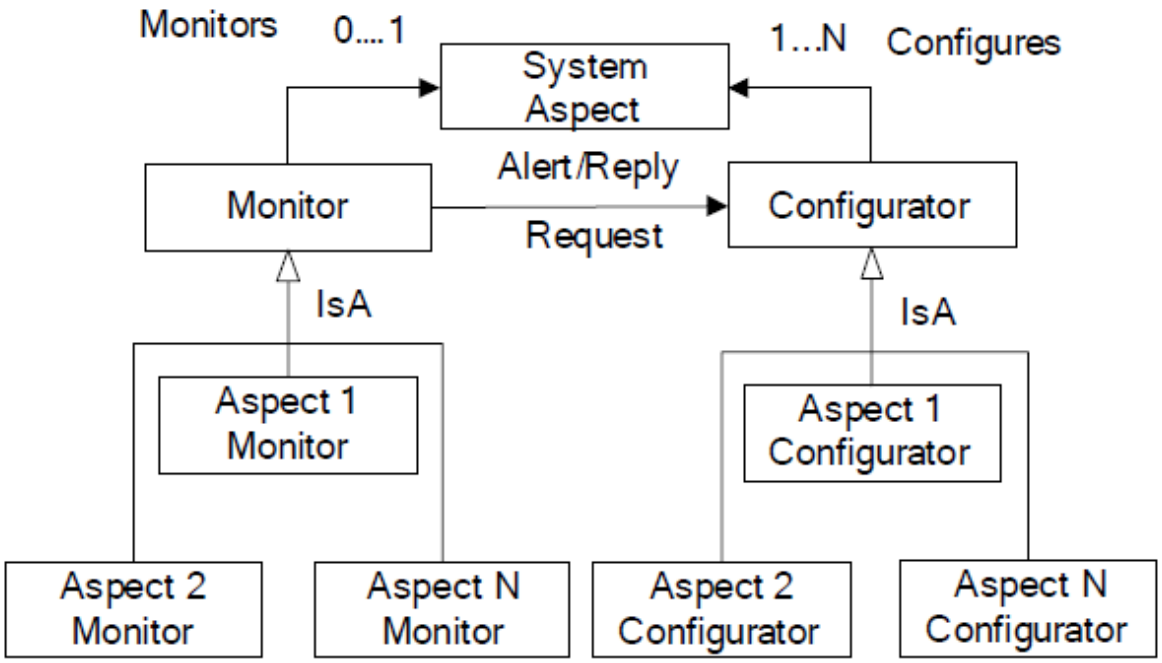

Figure 4. Aspect peer-to-peer architectural style

Architectural styles for self-healing systems

- **Aggregator-escalator-peer architectural style:**
  - Overcomes the limitations of a strict peer-to-peer approach by allowing monitors to pass their outputs to higher level aggregator monitors
  - Higher level configurator can then make better configuration decisions
  - Viewed as a set of autonomic elements where one autonomic element exists for each component of the system
  - Each element is **not** completely independent. Higher level elements need information from lower level elements to function properly

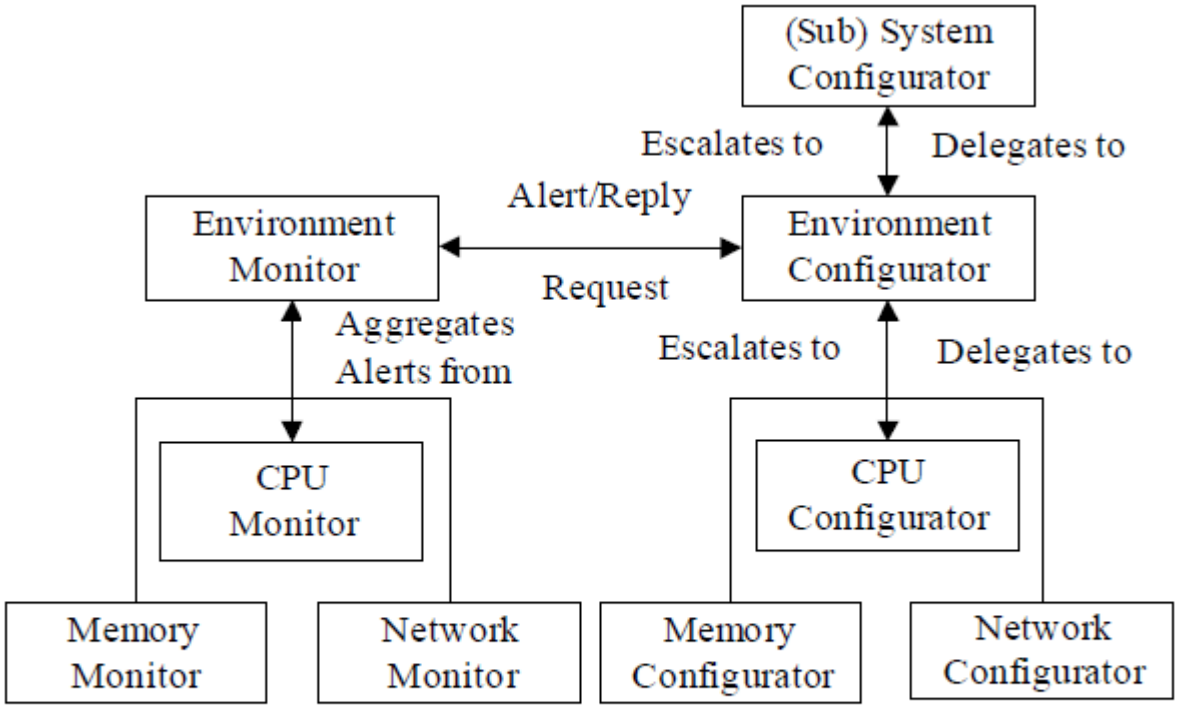

Figure 5. Aggregator-escalator-peer architectural style

Architectural styles for self-healing systems

- **Chain-of-configurators architectural style:**
  - Allows self-healing systems to try all available configuration strategies to repair a given problem
  - System can then promote successful strategies and demote less successful strategies while the system is running by just manipulating its configurator list.
  - Each configurator is separated from its corresponding autonomic element but all the configurators are chained together
  - A single monitor is allocated to the entire system
  - For each specific problem, an optimal configuration can be chosen as a solution
  - Enhances the flexibility and increases loose coupling compared

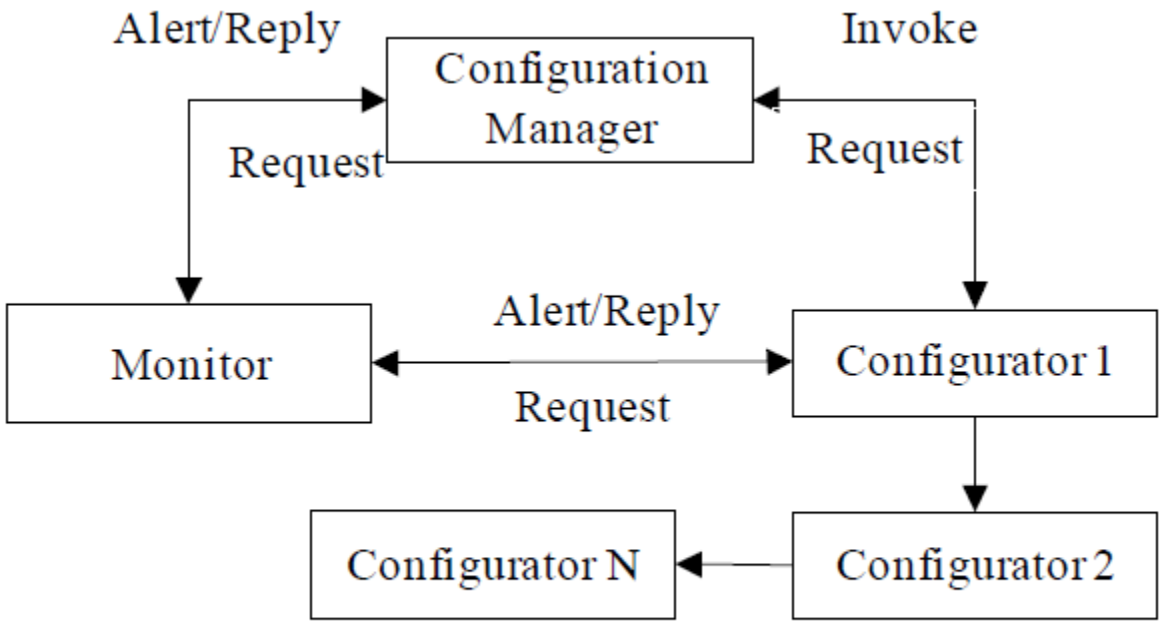

Figure 6. Chain-of-configurators architecture style

Attribute-based architectural styles (ABAS)

- **ABASs** provide a foundation for more effective reasoning of architectural design and evolution by explicitly associating a reasoning framework, qualitative, quantitative, or both, with an architectural style.

- For each attribute a separate reasoning framework is developed

- Klein define the structure or template of an ABAS to consist of four parts:
  - **Problem description:**
    - Description of the design and analysis problem that the ABAS is intended to solve, including the quality attribute of interest, the context of use, constraints, and relevant attribute-specific requirements.
  - **Stimulus/response attributes measures:**
    - A characterization of the stimuli to which the ABAS is to respond to and the quality attributes measures of the response
  - **Architectural style:**
    - A description of the architectural style in terms of component, connectors, properties of the components and connectors, as well as patterns of data and control interactions
  - **Analysis framework:**
    - A description of how the quality attribute models are formally related to the architectural style and a discussion of the architectural behavior.

Attribute-based architectural styles (ABAS)

- ABAS with respect to traditional and autonomic specific qualities:
  - **Traditional quality ABASs:**
    - Modifiability: consideration for previous architectural styles and used qualitative reasoning. (will be discussed in detail)
    - Reliability: consideration for the architectural style that belongs to a general family of reliability styles and employed quantitative reasoning

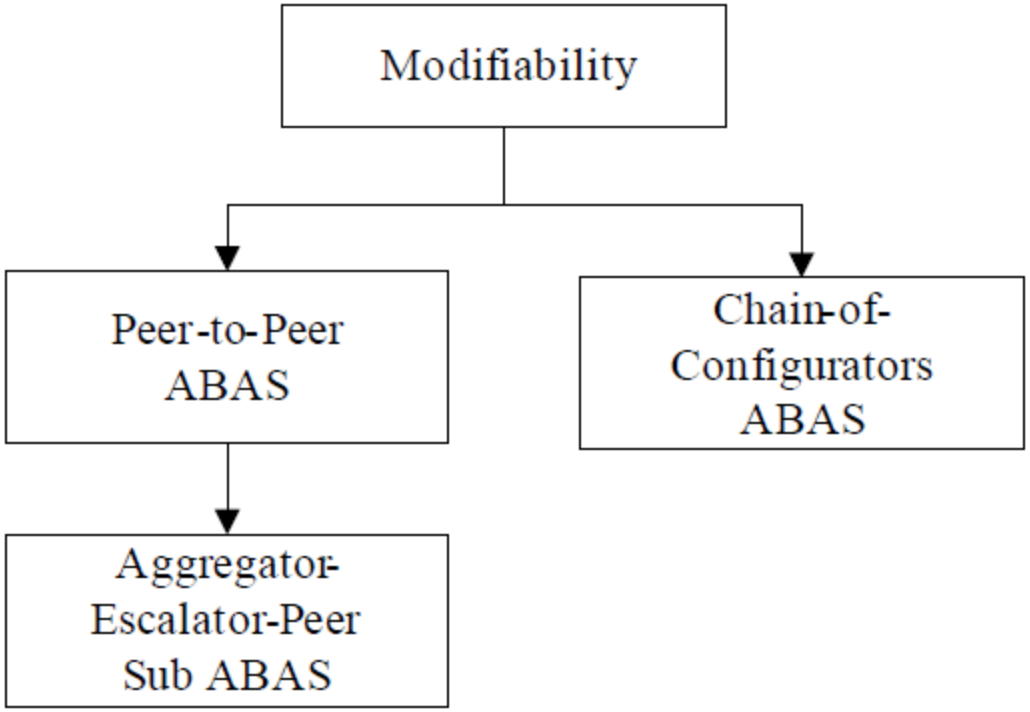

Figure 7. Characterization of the modifiability ABAS

Attribute-based architectural styles (ABAS)

- **Peer-to-Peer modifiability ABAS:**
  - **Problem description**
    - A component in this architecture has no knowledge or dependencies on components below it.
    - hides implementation details and thus provides for modifiability at different levels of abstraction
  - **Stimulus/response attribute measures**
    - A change in a peer layer and number of peers affected, number of components or connectors added, deleted, or modified.
  - **Architectural style**
    - Discussed previously in figure 4
  - **Parameters**
    - Topology and connectivity
  - **Analysis reasoning framework**
    - Changing a component at a peer level affects only the components at that peer level. (No propagation)
    - Examines how well the architecture supports the following modification scenarios:
      - *Adding a new monitor:* requires adding a new *configurator* (forms a new peer level)
      - *Adding a new configurator:* requires adding a new *monitor* (forms a new peer level)
      - *Deleting a monitor:* requires deletion of the *peer level configurator* (del. a peer level)
      - *Deleting a configurator:* requires deleting the *peer-level monitor* (del. a peer level)
  - **Design heuristics discussion**
    - Chose this ABAS for small systems with few monitors and configurators
    - Chose this ABAS if configurator does **not require output** from more than one monitor
    - Strict p2p approach decreases coupling. Does **not** support runtime modification.

Attribute-based architectural styles (ABAS)

- **Aggregator-escalator-peer modifiability sub ABAS:**
  - Discuss the reasoning framework for this ABAS
  - **Analysis reasoning framework**
    - We examine how the ABAS reacts to some of the scenarios defined in the peer-to-peer ABAS
    - Introducing a new monitor or configurator affects the components at a higher level
    - *Adding a new monitor:* necessitates change in **higher** level monitors to be able to use output from the new monitor
    - *Adding a new configurator:* Requires change in **higher** level configurators to be able to use output from the new configurator.
    - *Deleting a monitor:* Requires change in **higher** level monitors so that they do not send or receive messages from this monitor.
    - *Deleting a configurator:* Necessitates change in **higher** level configurators so that they do not send or receive messages from this configurator
  - **Design heuristics discussion**
    - Architecture provides full monitor output (better decisions and more efficient operation)
    - Architecture provides consistent output
    - Architectural style increases the coupling between components (decreasing modifiability)
    - Each layer comes at the cost of a performance penalty (traversing layers)
    - Architecture does **not** support run-time modification

Attribute-based architectural styles (ABAS)

- **Chain-of-configurators modifiability ABAS:**
  - Discuss the reasoning framework for this ABAS
  - **Analysis reasoning framework**
    - Introducing or deleting a configurator does not affect any other configurators in the chain
    - *Adding a new monitor:* Necessitates change in **other** monitors to use output from the new monitor
    - *Adding a new configurator:* This change does **not** affect any other component.
    - *Deleting a monitor:* Requires change in **other** monitors so that they do not send or receive messages from this monitor
    - *Deleting a configurator:* this change does **not** affect any other component
  - **Analysis and design heuristics discussion**
    - This ABAS can be chosen if run-time addition and removal of configurators is needed
    - This ABAS can be chosen if an optimal configuration solution from a set of existing ones is to be selected

Attribute-based architectural styles (ABAS)

- ABAS with respect to traditional and autonomic specific qualities:
- Considered the architectural styles mentioned in the previous section and used the qualitative reasoning framework
  - **Autonomic specific quality ABASs:**
    - Support for detecting anomalous system behavior ABAS is discussed in detail

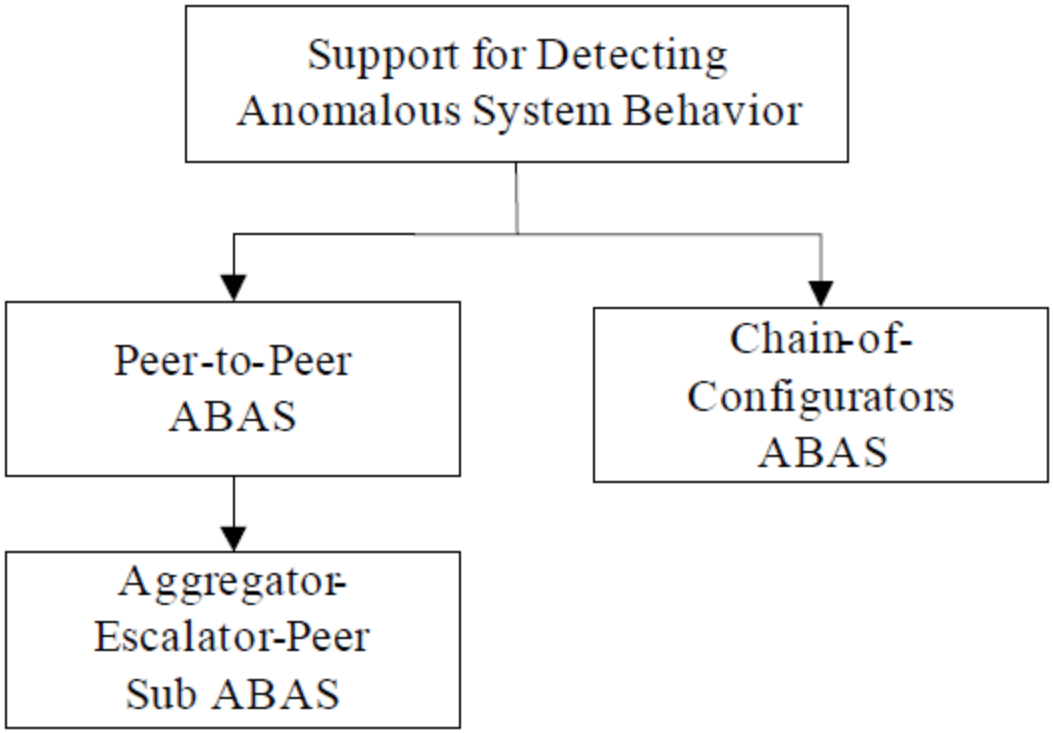

Figure 8. Characterization of support for detecting anomalous system behaviour ABAS

Attribute-based architectural styles (ABAS)

- **Peer-to-peer support for detecting anomalous system behavior ABAS:**
  - **Problem description**
    - Support this quality?
    - Criteria for choosing: 1) peer-to-peer topology is used. 2) problem has a requirement to support the detection of anomalous system behavior
  - **Stimulus/response attribute measures**
    - A fault in the system and system detecting the anomaly caused due to this particular fault.
    - Response obtained using the following measurable quantities:
      - Detection rate: needs to be good to satisfy quality requirement
      - Detection time: should be detected as early as possible to avoid failure
      - Coupling: causes dependencies among subsystems
      - Awareness: **internal** monitoring of the system.
      - Observaibility: **external** monitoring of the systems
      - Fault model: expected to identify the abnormal behavior of the systems.
  - **Architectural style**
    - Discussed previously in figure 4
  - **Analysis reasoning framework**
    - Some important questions:
      - What architectural decisions were made during the design of the system?
      - How does the system know if it is about to crash?
      - How can we assess normal behavior if the precise specification of the system is unavailable?
      - And others

Attribute-based architectural styles (ABAS)

- **Peer-to-peer support for detecting anomalous system behavior ABAS:**
  - **Analysis reasoning framework**
    - *Coupling:* there are no dependencies between peers and, hence, there is loose coupling. loosely coupled systems can easily be reconfigured
    - *Awareness:* A separate monitor is assigned to each component. But the monitors do not interact with each other and hence they have **no** knowledge of the other monitors in the system
    - *Observaibility:* Since there is loose coupling the architecture can easily be modified but this **cannot** be done at run time
    - *Fault model:* Suitable for small systems only
  - **Design heuristics discussion**
    - Some important questions:
      - What properties of the execution system should be monitored?
      - What constraints (e.g., reliability, availability) need to be evaluated?
      - And others
    - *Instrumenting software systems to gather information about the changes:* static(code) or dynamic(probes or gauges).
    - *Simulations to find the difference between the expected and actual behavior*
    - *Design for testability techniques :*Checking frequency of raised exceptions; monitoring log files...etc.
    - *Instrumentation to observe hard-wired requirements*

Attribute-based architectural styles (ABAS)

- **Aggregator-escalator-peer support for detecting anomalous system behavior sub ABAS:**
  - Only different aspects from the peer-to-peer will be discussed here
  - **Reasoning**
    - *Coupling:* There is coupling between components at the peer level as well as among different peers. Coupling makes it difficult to reconfigure
    - *Awareness:* A separate monitor is assigned to each component. Also the monitors interact with each other and, hence, they have **complete** knowledge about other monitors in the system. (more accurate decisions)
    - *Observaibility:* There is coupling among the peers the architecture cannot be easily modified. **Cannot** be done at run time
    - *Fault model:* Suitable for large scale systems
- **Chain-of-configurators support for detecting anomalous system behavior ABAS:**
  - Only different aspects from the peer-to-peer will be discussed here
  - **Reasoning**
    - *Coupling:* There is loose coupling between components. Coupling makes it difficult to reconfigure. Easier to reconfigure the system due to the flexible architecture
    - *Awareness:* There is **no** separate monitor and configurator for each component of the system. Monitor for the **whole system**. This loose coupling can affect awareness.
    - *Observaibility:* modifications **can** easily be done at **run time**.
    - *Fault model:* Suitable for all kinds of faults.

ABASs use in analysis

- Model problem to help illustrate the use of ABAs in analysis .
  - Extract the quality attributes that are most critical
  - Probe the architecture for styles that are used to satisfy requirements
  - Analytical models associated with them can be applied to understand ramifications of architectural decisions on the quality attributes.

- **Case Study: Model for Self-Managing Java Server**
  - Model for a self-managing java server
  - Self-configuration and self-healing aspects
    - Analyzers: monitors log file for any expectations
    - Polices: polices used by the analyzer (knowledge)
    - Policy manager.
    - Healing manager: gets the input from the policy manager in regards to expectations for planning. Decides which healer to call to take corrective actions for the problem
    - Healers: executes corrective actions
  - Analysis process:
    - Identify quality attributes that are important:
      - Problem: The java server logs the exception or error and **aborts** the current transaction
      - Solution: Considering modifiability, support for detecting anomalous system behavior, and support for failure diagnosis
    - Identify styles from the architecture
      - It is clear that the **chain-of-configurators** architectural style is used for this system
    - Observed ramifications:1)enhances loose coupling. 2)enhances loose coupling between components. 3)diagnosis becomes easier

Conclusions and future work

- Analysis and reasoning framework to evaluate and document the architecture of self-healing systems.
- Used ABAS analysis techniques
- Identified several key quality attributes for self-healing
- Employed the ISO 9162 quality model to customize it to the quality requirements discussed (traditional and autonomic)
- Benefit of the framework that it can be reused.
- Future work:
- Can be extended to other self-managing applications
- Propose to record the relationship between architecture and quality attributes of self-managed systems using reverse engineering handbook
- Predict that autonomic-specific quality attributes can be analyzed by directly simulating events and observing responses on the running application .