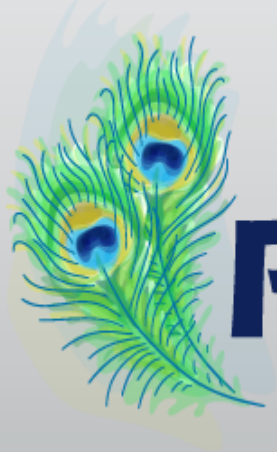


PANOPTESEC

Software Architecture

20/05/2015

Content prepared by: Matteo Merialdo

Software Architecture applied to PANOPTESEC System

Course Subjects and Objectives

- What the term *software architecture* means?
- The role and importance of software architecture on a software project
- Possible methodologies for building software architectures
- Principles of Requirements Engineering
- Modeling software architectures
- Application of theory to a real case: PANOPTESEC System
- Practical exercises

PANOPTESEC

Software Architecture

'The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.'

- Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice, Third Edition.* Boston, MA: Addison-Wesley, 2012)

PANOPTESEC

Software Architecture

A software architecture is an abstraction of a system. The architecture:

- defines elements and how they relate to one another
- suppresses details of what the elements do internally and purely local information about them; internal details are not architectural
  - The internal details of the elements do not affect how the elements are used or how they relate to or interact with other elements.

PANOPTESEC

Software Architecture

Software architecture defines properties of the System, those assumptions that one element can make about another element such as

- which services it provides
- how it performs
- how it handles faults
- how it uses shared resources
- How elements interact with each other via interfaces that partition details into public and private parts. Architecture is concerned with the public side of this division.

PANOPTESEC

Software Architecture

Every system has an architecture.

- Every system is composed of elements, and there are relationships among them.
- In the simplest case, a system is composed of a single element, related only to itself. Just having an architecture is different from having an architecture that is known to everyone:
  - Is the “real” architecture the same as the specification?
  - What is the rationale for architectural decisions? If you don’t explicitly develop an architecture, you will get one anyway—and you might not like what you get!
- A good architecture is one that allows a system to meet its functional, quality attribute, and lifecycle requirements

Software Architecture

Architecture is important for three primary reasons:

- It provides a vehicle for communication among stakeholders.
- It is the manifestation of the earliest design decisions about a system.
- It is a transferable, reusable abstraction of a system

PANOPTESEC

Software Architecture

Architecture dictates the structure of the organization.

- Architecture represents the highest level decomposition of a system and is used as a basis for
  - partitioning and assigning the work to be performed
  - formulating plans, schedules, and budgets
  - establishing communication channels among teams
  - establishing plans, procedures, and artifacts for configuration management, testing, integration, deployment, and maintenance
- For managerial and business reasons, once established, an architecture becomes very difficult to change.

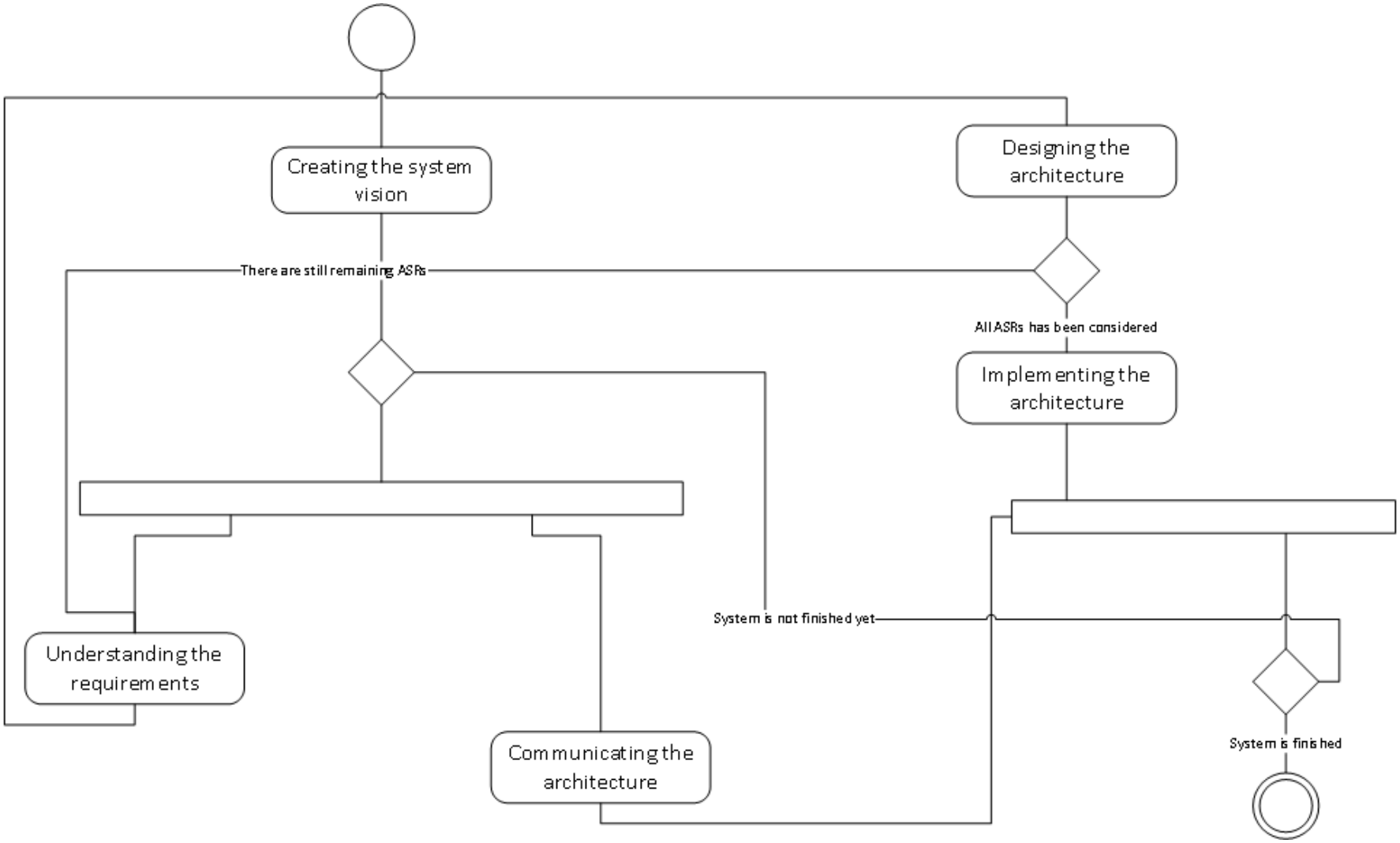
Software Architecture

Architecture enables more accurate cost and schedule estimates, project planning, and tracking

- The more knowledge we have about the scope and structure of a system, the better our estimates will be
- Teams assigned to individual architectural elements can provide more accurate estimates
- Project managers can roll up estimates and resolve dependencies and conflicts.

PANOPTESEC

Building a software architecture



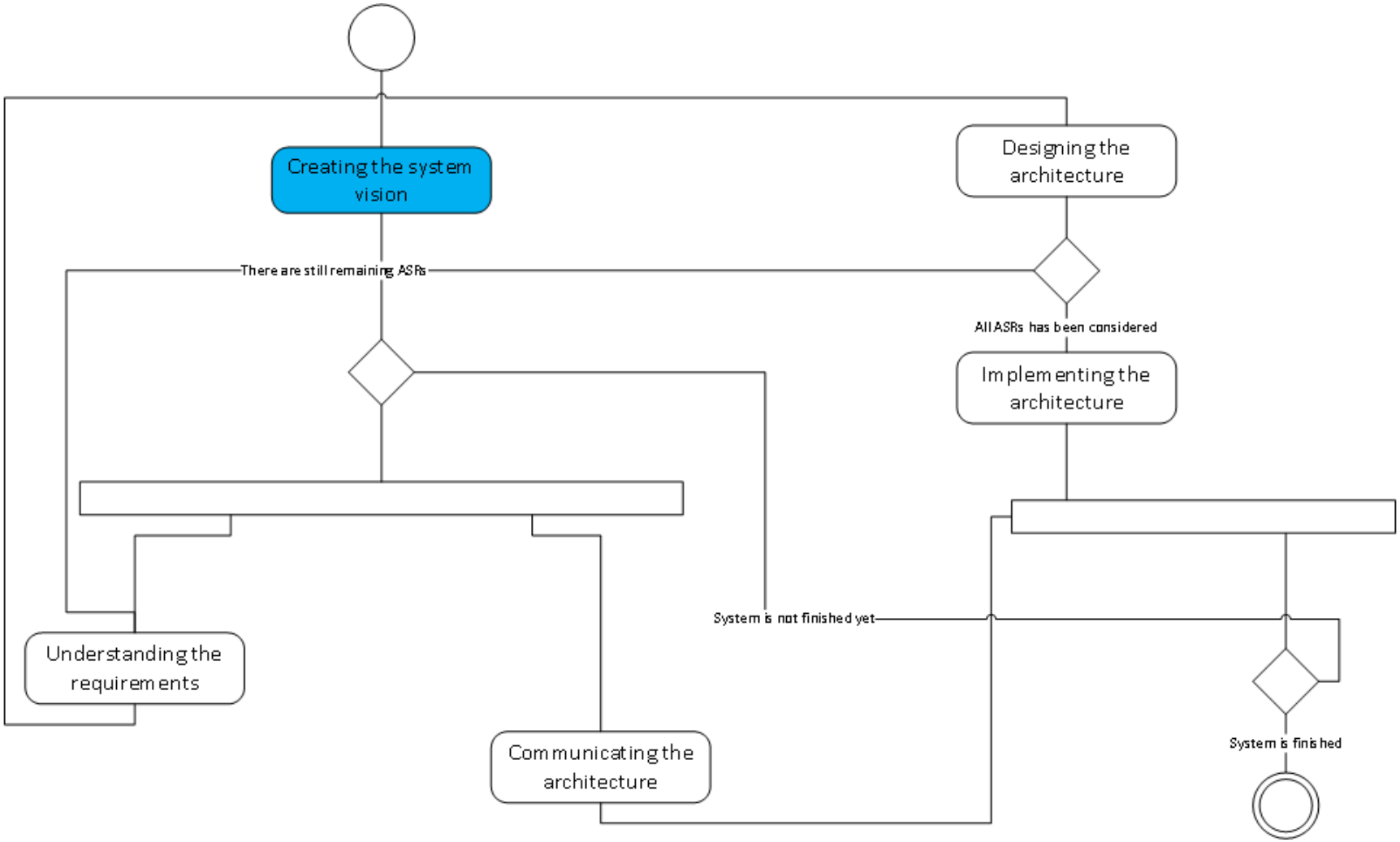
PANOPTESEC

Creating the System Vision

Where Do Architectures Come From?

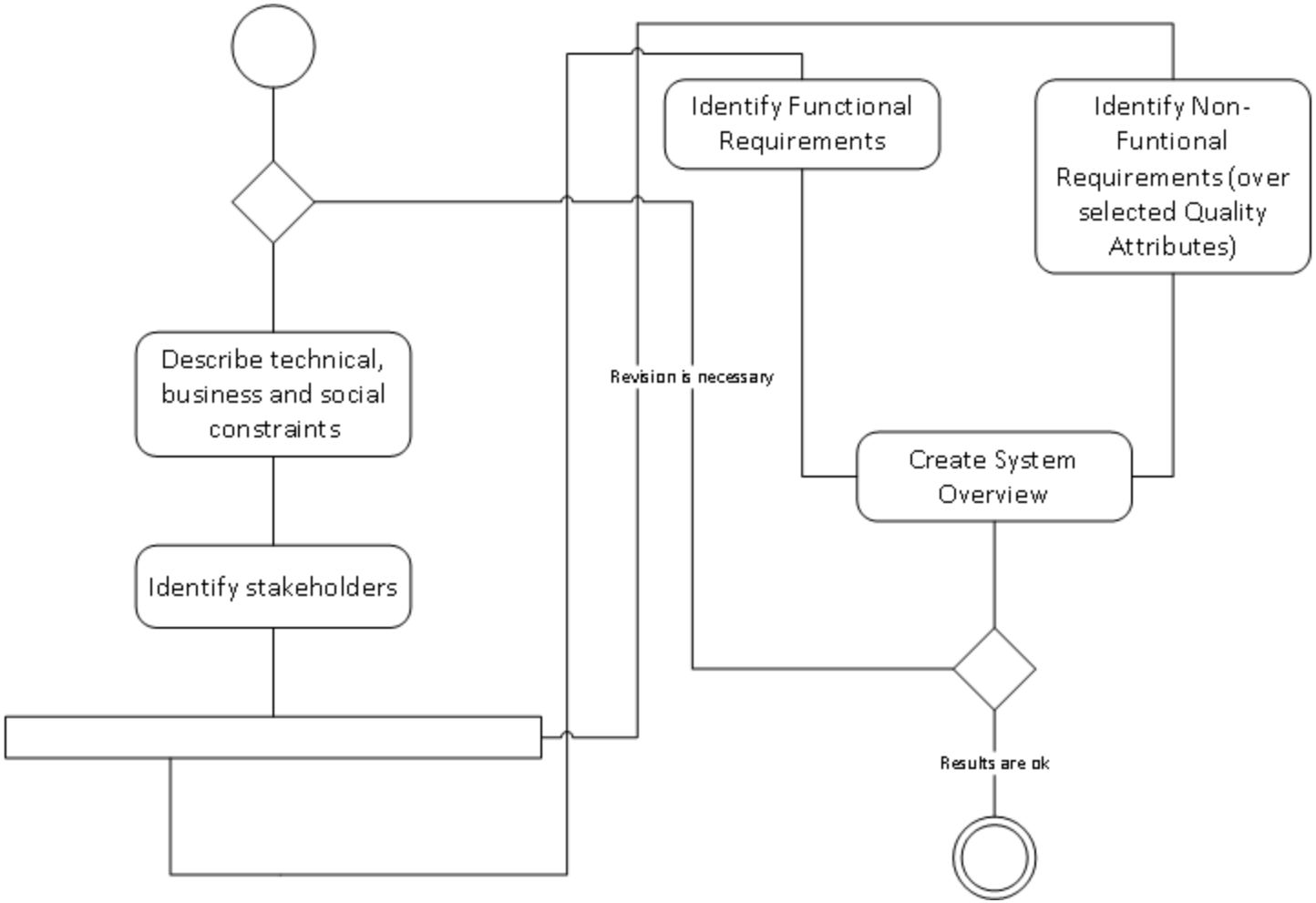
- Software architecture is based on much more than requirements specifications.
- It is the result of many different technical, business, and social influences.
- Its existence, in turn, influences the technical, business, and social environments that subsequently affect future architectures.
- Architects need to know and understand the nature, source, and priority of these influences as early in the process as possible.

Creating the System Vision



PANOPTESEC

Creating the System Vision



PANOPTESEC

Creating the System Vision

Factors Influencing Architectures

- system stakeholders
- the development organization's business environment
- the technical environment
- the architect's professional background and experience

Creating the System Vision

Influence of System Stakeholders

- customers (low cost, timely delivery)
- users (behaviour, performance, security, reliability, usability)
- developers (modifiability, languages, technology)
- project managers (low cost, keeping people employed, timely delivery)
- marketers (next features, short time to market, low cost)
- maintainers (modifiability)

An architect must find a trade-off between these different needs!

Creating the System Vision

The organizational goals and system properties required by the business/mission are rarely understood, let alone fully articulated. Customers' quality attribute requirements are seldom documented, which results in

- goals not being achieved
- inevitable conflict between stakeholders. Architects must identify and actively engage stakeholders early in the lifecycle to
- understand the real constraints of the system
- manage the stakeholders' expectations
- negotiate the system's priorities
- make tradeoffs

Creating the System Vision

Influence of Development Organization's Business Environment

Software architecture can be shaped by business/mission concerns, including

- time to market
- rollout schedule
- use of legacy systems
- available expertise
- support for existing products
- targeted markets
- political interests
- existing architectures
- plans for long-term infrastructure
- organizational structure
- projected lifetime of the system
- workforce utilization
- cost
- investment in existing assets

PANOPTESEC

Creating the System Vision

Influence of Technical Environment on Architectures

The technical environment that is current when an architecture is designed will influence that architecture.

- Multicore
- Cloud computing
- HTML5
- Mobile Web
- Web Services
- Integration Frameworks
- Service-Oriented Architectures
- …

Creating the System Vision

Influence of Architect's Professional Background on Architectures

Architects make choices based on their past experiences:

- Good experiences will lead to the replication of those prior designs that worked well.
- The methods, techniques, and/or technology that led to bad experiences will be avoided in new designs, even if those methods or techniques might work better in subsequent designs.
- An architect's choices might be influenced by education and training

Quality Attributes

Quality attributes are properties of work products or goods by which stakeholders judge their quality. It is necessary to identify a set of Quality Attributes in order to define Non-Functional Requirements.

Some examples of quality attributes by which stakeholders judge the quality of software systems are:

- Performance
- Security
- Modifiability
- Dependability
- Usability
- Compatibility
- Portability

PANOPTESEC

Stakeholders and Quality Attributes

Stakeholders influence quality attributes requirements

Stakeholder Concerns	Quality Attribute Requirements
"Increase market share"	Modifiability, Usability
"Maintain a quality reputation"	Performance, Usability, Availability
"Introduce new capabilities seamlessly"	Performance, Availability, Modifiability
"Provide a programmer-friendly framework"	Modifiability
"Integrate with other systems easily"	Interoperability, Portability, Modifiability

PANOPTESEC

Quality Attributes

Quality Attributes and Architecture

- Quality attribute requirements for software systems have a significant influence on the architecture of those systems.
- The degree to which a software system meets its quality attribute requirements depends on its architecture.
- Architectural decisions are made to promote various quality attributes.
- A change in an architecture to promote one quality attribute often affects other quality attributes.
- Architecture provides the foundation for achieving quality attributes but is useless if not adhered to in the implementation.

Quality Attributes

Describing Quality Attributes

Quality attribute names by themselves are not enough.

- Quality attribute requirements are often non-functional. – For example, it is meaningless to say that the system shall be “modifiable.” Every system is modifiable with respect to some set of changes and not modifiable with respect to some other set of changes.
- Heated debates often revolve around the quality attribute to which a particular system behavior belongs. – For example, system failure is an aspect of availability, security, and usability.
- The vocabulary describing quality attributes varies widely

Functional Attributes

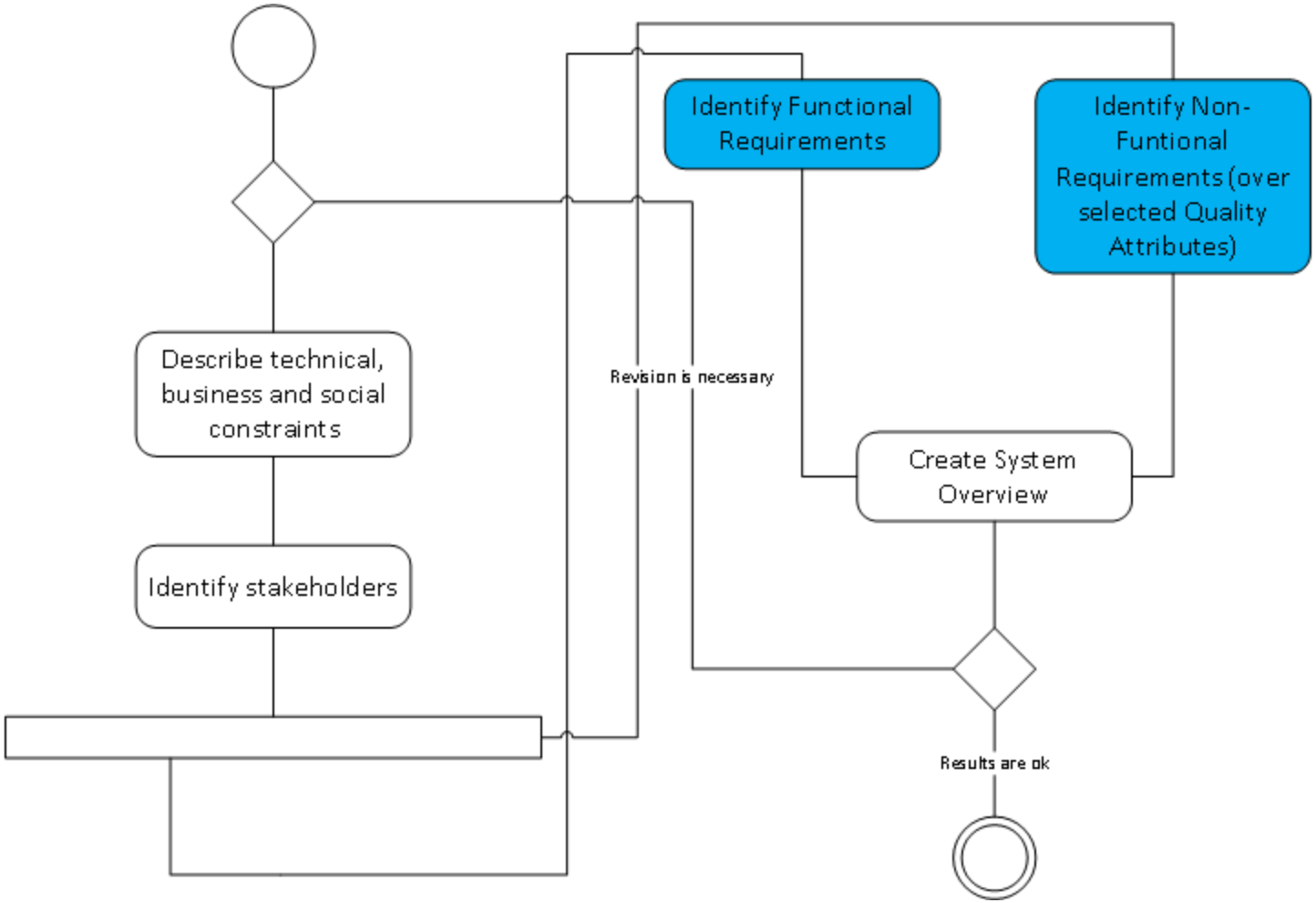
Functionality and Architecture

Identifying the functional requirements means defining the functionality the system is to provide.

- We can achieve functional requirements and yet fail to meet quality attribute requirements: the system, while functional, may be unsuccessful
- Functionality can be achieved using many different architectures, while
- achieving quality attribute requirements can only be achieved through judicious choice of architectures..

PANOPTESEC

Creating System Vision: Identify Requirements



PANOPTESEC

Requirements Engineering

- Quality Attributes and Functional Attributes play a major role on the definition of System's Requirements
- System's Functional and Non-Functional Requirements are defined on a set of processes defined by Requirements Engineering discipline.
- Requirements Engineering is the process of establishing the services that the stakeholders require from a system and the constraints under which it operates and is developed

Requirements Engineering

- Requirements gathering
  - (a.k.a. “requirements elicitation”) helps the stakeholder to define what is required: what is to be accomplished, how the system will fit into the needs of the business, and how the system will be used on a day-to-day basis
- Requirements analysis
  - refining and modifying the gathered requirements
- Requirements specification
  - documenting the system requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness

Requirements Engineering

In principle requirements should be both complete and consistent

Complete

They should include descriptions of all facilities required

Consistent

There should be no conflicts or contradictions in the descriptions of the system facilities

In practice, it is very difficult or impossible to produce a complete and consistent requirements document

Requirements Engineering

In most cases, it could be useful to describe business and stakeholders needs using Use Case Scenarios, short “stories” identifying a relevant use case:

- Title:
- Primary Actor
- Goal in Context
- Scope
- Stakeholders and Interests
- Precondition
- Minimal Guarantees
- Success Guarantees
- Main Success Scenario

Identify Functional Requirements

- Describe functionality or System services, gathered from stakeholders, description of works, project proposals
- Depend on the type of software, expected users and the type of system where the software is used
- Directly related to Functional Attributes

Identify Non-Functional Requirements

- NFRs define system properties
  - reliability
  - response time and storage requirements
- NFRs Constraints
  - I/O device capability
  - system representations, etc.
- NFRs Are directly related to identified Quality Attributes
- Non-functional requirements are more critical than functional requirements. If they are not met, the system may be useless. Modern Software Architectural methods are mainly based on the analysis of Non-Functional Requirements

PANOPTESEC

Identify Non-Functional Requirements

- Questions?

PANOPTESEC Project

- Problem
  - Lack of security-driven continuous and dynamic Management for large and complex ICT systems
  - Security relies on “know-how” and “expertise”
- Relevance
  - Modern Systems are growing in size and complexity
  - Attacks are more frequent, and new vulnerabilities are reported on daily basis
  - Attacks are getting more accurate, sophisticated
  - Traditional (Paper-based) Risk Management methods are obsolete

PANOPTESEC

Challenges

- Scripted Multi-step attacks are difficult to assess
- Priority targets are difficult to identify
- Mission/System dependency is difficult to determine
- System interdependency is difficult to assess
- Attacks are complex and non-obvious
- Optimal mitigations are difficult to determine

PANOPTESEC Project

PANOPTESEC System

Operational use of Dynamic Risk Approaches (DRA) for Automated Cyber Defense

- Proactive
  - Focus on potential attack paths to high priority systems
  - Used to prioritize vulnerability remediation activities
  - Interim mitigations may leverage system reconfiguration
    - Shutting down services, blocking ports, etc
- Reactive
  - Focus on blocking or preventing spread of ongoing attacks
  - Must leverage rapid reconfiguration of systems
    - Shutting down services, blocking ports, etc.

PANOPTESEC

PANOPTESEC Project – Description of Work

The PANOPTESEC consortium will deliver a beyond-state-of-the-art prototype of a cyber defence decision support system, demonstrating a risk based approach to automated cyber defence that accounts for the dynamic nature of information and communications technologies (ICT) and the constantly evolving capabilities of cyber attackers. “Panoptes” is an ancient Greek term meaning ‘all eyes’ or ‘all seeing’. This term has incorporated into the project name to represent the PANOPTESEC consortium because the overall goal of the PANOPTESEC project is to deliver a continuous cyber security monitoring and response capability.

The PANOPTESEC prototype will address these challenges by proactively and reactively evaluating system weaknesses, identifying potential attack paths, providing a list of prioritized response actions, and delivering a means to execute these responses; all supported by automated analysis engines. The resulting PANOPTESEC prototype will provide a continuous monitoring and response capability to prevent, detect, manage and react to cyber incidents in real-time. The near market-ready system will support breach notifications and improve situation awareness while supporting the decision-making process required by security personnel. PANOPTESEC will deliver this capability through an integrated and modular, standards-based integration of technologies that will collectively deliver the required capabilities.

PANOPTESEC

- Summary of main innovations
  1. Improved multi-sensor data integration and correlation based on Semantic Web (i.e., Description Logic) technologies;
  2. Automation of attack awareness and risk assessment via:
     - Attack modelling taking into account the dynamic aspects of systems and services, threat, system vulnerabilities, and mission priorities;
     - Automated risk quantification;
     - Decomposition of risk into proactive and reactive assessment chains.
  3. Automation of response assistance to provide cyber defence operators with prioritized courses of action recommendations for review and activation;
  4. Advanced mechanisms to capture and model mission/business process relationships to services and systems based on Semantic Web technologies; and
  5. Advanced visualization techniques for mission/business process risk display including representation of risk levels derived from risks in supporting services and systems.

PANOPTESEC Project

SME - Experienced technology company
- Corporate experience in both FP programmes and cyber defence

Large telecom equipment provider
- Security research dept. Bell Labs France focus on cyber security and cloud security

SME – Knowledge based systems for industrial applications
- Specialized in description logics and semantic web

Sapienza Research Centre on Cyber Intelligence and Information Security
- Research focus on distributed systems and information visualization

Institute for Software Systems and Institute of Security in Distributed Systems
- Research focus on knowledge based systems and description logics

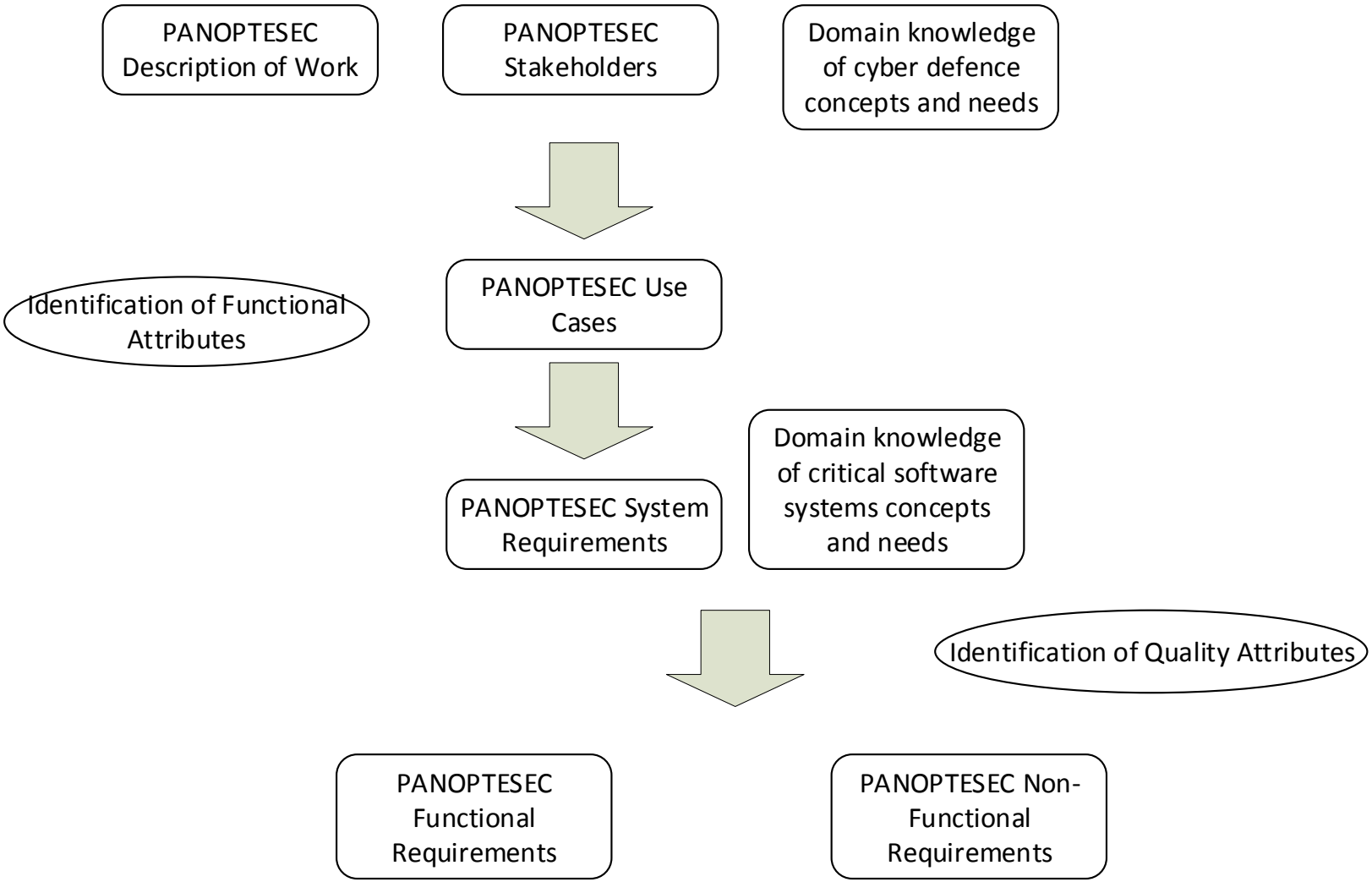
Industrial development of ICT through the science and technology
- Distributed Services, Architectures, Modelling, Validation and Network Administration

Important Italian public utility which focuses on energy and water
- Operational environment for experimentation and demonstration

The “CIDre” team (*Confidentialité, Integrité, Disponibilité, et repartition*)
- Research focus on intrusion detection, correlation and visualization

PANOPTESEC

PANOPTESEC Project Requirements Gathering



PANOPTESEC

PANOPTESEC Project - Stakeholders

Client Stakeholders

- European Commission

Market Stakeholders

- ACEA
- European Defense Agency
- French Information Systems Security Agency

Partner Stakeholders

- Technical Project Manager
- Software Architects
- Software Developers
- Domain expert

PANOPTESEC

PANOPTESEC Project – Use Cases

Various use cases have been described for the PANOPTESEC System. Among them:

- Attacks to Command and Control networks and systems of a Critical Infrastructure
- Attacks to the SCADA (*Supervisory Control And Data Acquisition*) equipment/devices of a Critical Infrastructure, performed through its SCADA Command and Control network, that have a relevant social impact
- Attack to one or more nodes that operate some ICT services underlying a business process.

PANOPTESEC

PANOPTESEC Project

PANOPTESEC System Requirements
The PANOPTESEC System MUST proactively calculate risk, identify and actuate courses of action based on evaluated operational impact due to updated knowledge of cyber vulnerabilities and changes in network and computer system configuration.
The PANOPTESEC System MUST provide multi-sensor data integration and correlation of infrastructure, security and operation data through the use of advanced ontology and inference technologies.
The PANOPTESEC System MUST reactively calculate risk, identify and enable Security Operators to actuate courses of action, based on evaluated operational impacts due to detected cyber incidents.
The PANOPTESEC System MUST recommend to Security Operators reactive courses of action optimized to maximally deny attacker capabilities while minimizing operational impact.
The PANOPTESEC System MUST provide the option to actuate reactive courses of action in a fully automatic mode (i.e., without operator intervention).

PANOPTESEC Project

PANOPTESEC System Requirements
The PANOPTESEC System MUST provide optimized notification of detected cyber incidents through the use of advanced technologies for multi-sensor data integration and correlation.
The PANOPTESEC System SHOULD provide an advanced visual analytics engine to improve Security Operator's ability to access and analyze large complex data sets.
The PANOPTESEC System MUST provide proactive and reactive risk and response models resulting in optimized course of action recommendations to Security Operators.
The PANOPTESEC project SHOULD deliver an integrated system of beyond-state-of-the-art security algorithms and software modules providing continuous monitoring and automated response for cyber defense.
The PANOPTESEC project MUST conduct demonstrations on an operational segment of user agency (ACEA) for comparison of PANOPTESEC-delivered capabilities against the status quo.

PANOPTESEC

PANOPTESEC Project – Quality Attributes

It makes no sense to try to fulfil dozens of Quality Attributes: most of them are often in contradiction (Performance and Security, for example, needs a trade-off analysis if both are required). It is better to select a restricted list, after the first gathering of the System Requirements:

- Performance
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

PANOPTESEC Project – Non-Functional Requirements

id	Description	Goal	MainPurpose	Importance	Reachability	Version
PRF001	The PANOPTESEC System SHOULD improve Security Operator course of action determination time from days to minutes in response to awareness of new cyber vulnerabilities and changes in network and computer system configuration.	Ensure that the PANOPTESEC system really improves the situational awareness of security operators.	Understanding precisely the state of the monitored system, the events that happen and what is to be done on it is a very difficult task as soon as the system is made of more than a few machines. The PANOPTESEC system intends to improve this situation.	2	3	1
PRF002	The PANOPTESEC system SHOULD reduce the total number of cyber incident alerts generated from multiple alert sensors (e.g., IDS, Firewalls, Syslog servers) by up to 25% through alert correlation functions.	Ensure reduce the number of false positive alerts that are provided to Security Operators and provide semantically rich real positive alerts.	Security Operators are often overwhelmed by false positive and/or semantically poor alerts. The PANOPTESEC system benefit from its correlation system to reduce false positive and improve the semantic value of the alerts it raises.	2	3	1
PRF003	The PANOPTESEC system SHOULD reduce the false negative cyber incident detection rate of a single Intrusion Detection System (IDS) by at least 25%, through alert correlation of multiple cyber incident detection sources.	Ensuring that the PANOPTESEC system detects more real malicious actions that a single Intrusion detection system.	The PANOPTESEC system will benefit from its multiple information sources and its correlation systems to improve the detection rate on real malicious action compared to a single intrusion detection system.	2	3	1
PRF004	The PANOPTESEC project SHOULD analyze system scalability using analytical methods to derive expected PANOPTESEC system response times in environments ranging from 10 to 10,000 nodes, in various typical network topologies (e.g., interconnected star and limited mesh) wherein nodes have a range from 1 to 10 applications installed and 10%, 20% and 30% of nodes each have a range of 1 to 3 vulnerabilities present.	Ensuring that the PANOPTESEC system can scale to monitor medium to large systems.	Systems made of hundreds of nodes are common and systems made of thousands of nodes can be found. The PANOPTESEC system SHOULD be designed to scale to networks of this size.	2	3	1

PANOPTESEC Project – Non-Functional Requirements

id	Description	Goal	MainPurpose	Importance	Reachability	
CMP001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve flexibility (in organization and configuration), ability to develop new function combinations rapidly and component re-use	Ensure that each component can be modified/replaces with little impact on the other components.	The PANOPTESEC is made of various components that interact. While these components are closely related, it is important that modifications can occur in each of them with minimal impact on the design and implementation of the others.	3	2	1
CMP002	Communications between the various functional components of the PANOPTESEC System MUST be handled in a transparent way by a middleware in order to reduce complexity for developers.	Ensure that developers will not spend too much time handling network communications.	Network communications design, implementation and debugging can rapidly become complicated. Developers of the PANOPTESEC project MUST be able to focus on components functionalities and should not waste time on network communications.	3	2	1

PANOPTESEC Project – Non-Functional Requirements

id	Description	Goal	MainPurpose	Importance	Reachability	Version
MNT001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve maintainability.	Ensure that the PANOPTESEC system can be easily maintained even when its various components evolve and new features are implemented.	Maintaining a project comprising multiple technologies provided by multiple partners can be difficult. A modular architecture helps maintainability.	3	2	1
MNT002	The PANOPTESEC data model SHOULD be designed to support integration with new data sources in the monitored system(s)	Ensure that new data sources having different formats can be added to the PANOPTESEC system.	While a set of data sources have already been identified for the PANOPTESEC system, new types of data sources will certainly appear as they provide valuable information. It SHOULD be possible to support integration of new data sources in the PANOPTESEC project.	2	2	1
MNT003	The PANOPTESEC System SHOULD provide a unique configuration environment to ease the maintenance of the deployed components.	Ensure that the PANOPTESEC system can be easily deployed, managed and configured by system administrators.	Systems that are too complex to deploy, manage and configure are rarely adopted. The PANOPTESEC system SHOULD be easy to deploy, manage and configure.	2	2	1

PANOPTESEC Project – Functional Attributes

After the analysis of the needed functionalities, a set of Functional Attributes can be described for the PANOPTESEC System:

- The system must be able to collect data from the monitored network
- The system must be able to correlate these data
- The system must be able to proactively compute the level of risk (based on the correlated data)
- The system must be able to reactively compute the level of risk (based on the correlated data)
- The system must be able to offer a complete a customizable visualization for the previous activities

PANOPTESEC

PANOPTESEC Project – Functional Requirements

id	Description	Goal	MainPurpose	Importance	Reachability	Version
DSC001	The PANOPTESEC system MUST provide a data collection system	Ensuring that the PANOPTESEC system uses data that are representative of the current state of the system and of threats.	In order to perform proactive and reactive response, data about the system and threats are necessary.	3	1	1
DSC002	The data collection system MUST provide a standard based interface for data collection from data sources forming part of the monitored system(s).	Ensuring an as automated as possible mechanism to collect data to reduce the burden of collecting data.	Collecting data manually is an error-prone human resource-consuming task. Automated collection of data through standard based interfaces both reduces human resources impact and limits errors.	3	2	1
DSC003	The data collection system MUST provide a standard based interface for data collection from the data sources not forming part of the monitored system(s) (e.g., vulnerability advisory sources).	Ensuring that data that are not specific to the monitored systems (vulnerability advisories for instance) are also automatically collected.	The PANOPTESEC system uses not only information from the monitored system (i.e. IDS/IPS/FWS logs), but also public vulnerability advisory databases (CVE). As standard based-interfaces exist, it would be inefficient and error prone not to benefit from them.	3	2	1

PANOPTESEC Project – Functional Requirements

id	Description	Goal	MainPurpose	Importance	Reachability	Version
ICA001	The PANOPTESEC system MUST contain an information correlation engine.	Ensuring that the PANOPTESEC system can correlate automatically the various collected and computed pieces of information.	Since the PANOPTESEC system collects very different types of information from many sources, it is fundamental to provide an engine that is capable of correlating the various pieces of information automatically.	3	1	1
ICA002	The information correlation engine SHOULD translate multisource information received by the data collection system into a semantic (logic-based) common information representation.	Ensuring that the internal representation of the information in the PANOPTESEC system is consistent whatever the format (syntax) of the data when it was acquired.	The PANOPTESEC system will collect pieces of information from various sources. For a given kind of input (IDS alerts for instance), different sources may provide data in different formats. For efficient correlation, a common semantic representation must be defined to which pieces of data collected in different formats can be translated.	2	3	1
ICA003	The information correlation engine MUST identify common information elements across the multi-source information received by the data collection system.	Ensuring that the PANOPTESEC system has a common internal data model to handle consistently the pieces of information coming from various sources.	The various data sources used by the PANOPTESEC system may have different representations and syntaxes to describe what is actually the very same concept (semantically equivalent). The PANOPTESEC system internal data model MUST be able to define when two syntactically different pieces of data refer to the same concepts from a semantic perspective.	3	3	1

PANOPTESEC Project

- Questions?

Requirements Modeling

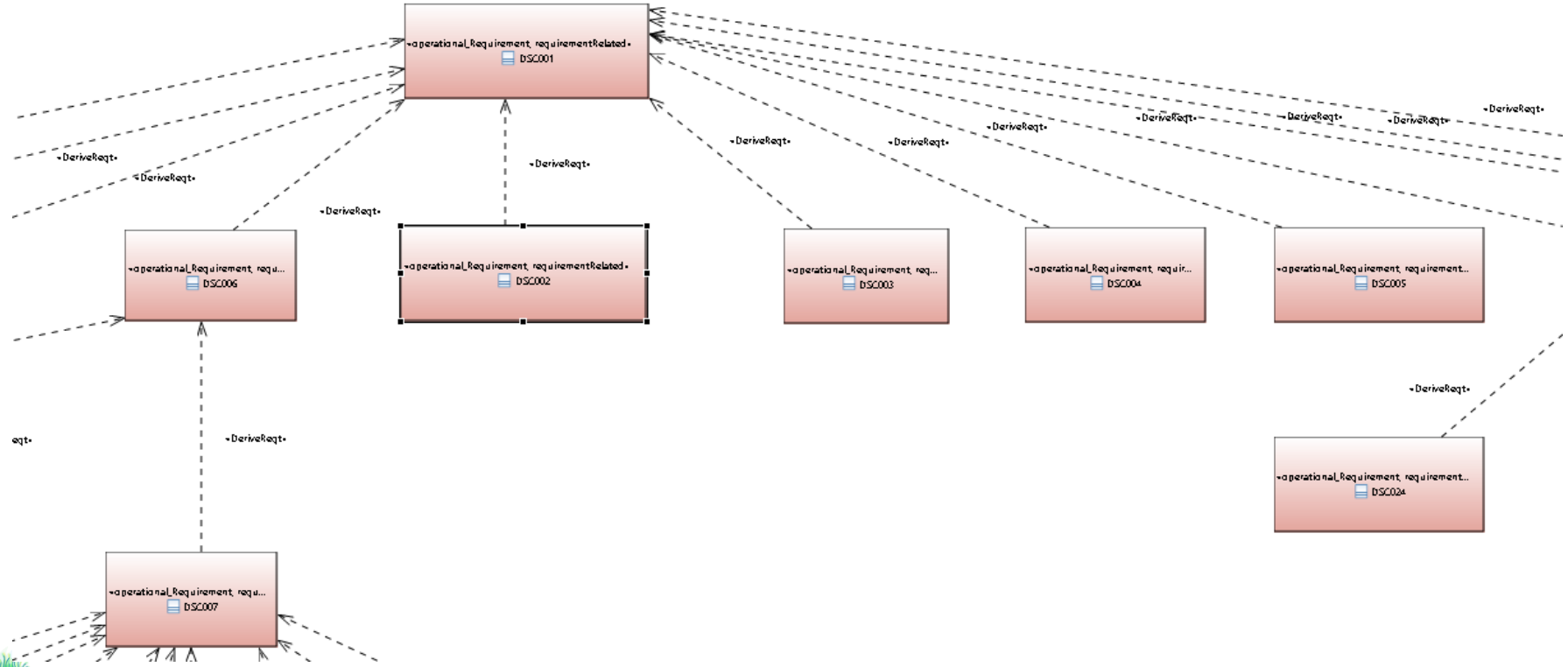
Many modeling languages offer techniques for modeling Requirements:

- A Requirements model is useful in order to depict relationships between Requirements and so possible relationships between System's elements
- Requirements modeling enhances traceability
- UML and SysML can be used for Requirements Modeling

PANOPTESEC

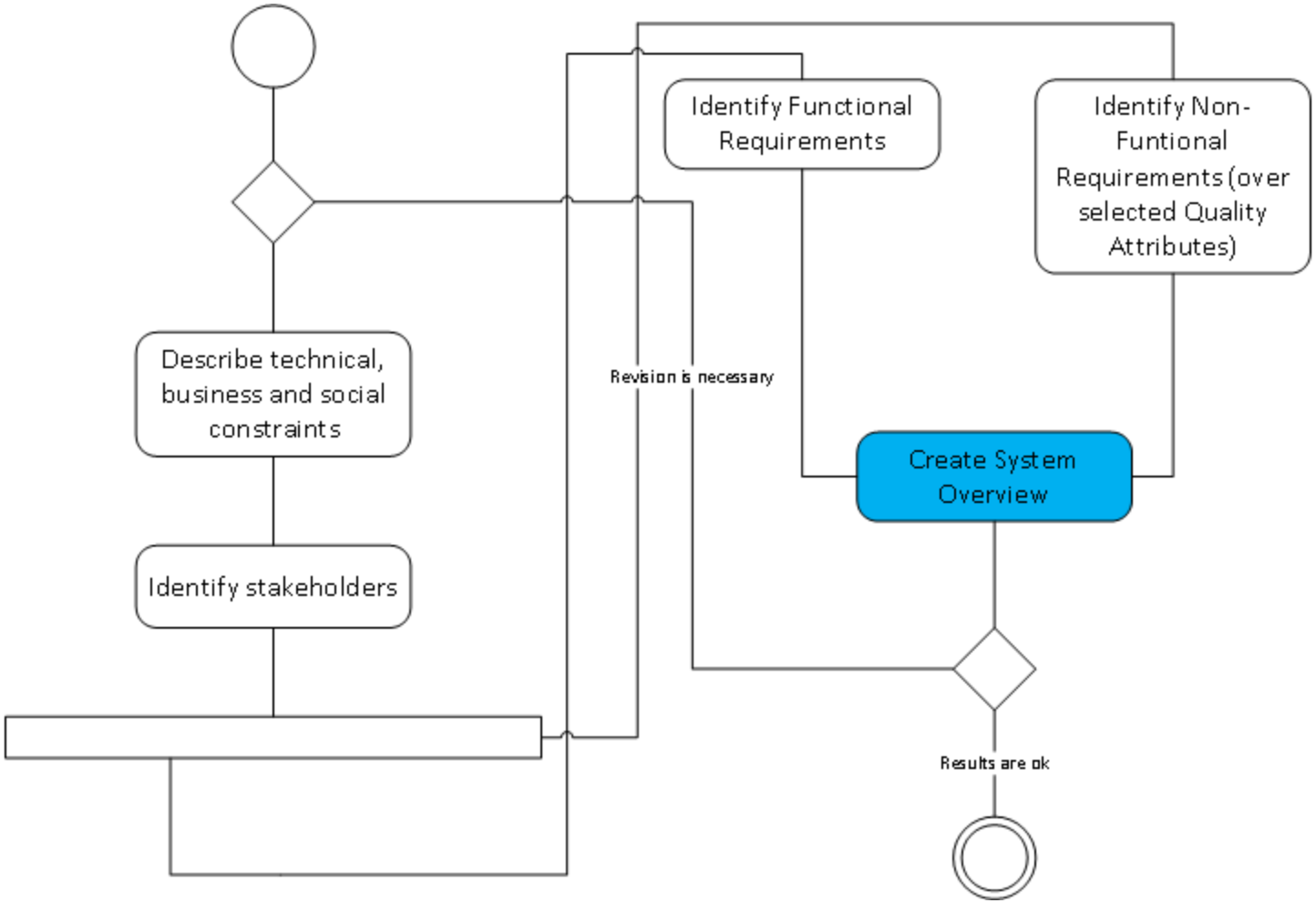
PANOPTESEC - Requirements Modeling

Every Requirements in PANOPTESEC has been modeled using SysML Requirements Diagram (Eclipse/Papyrus tool)



PANOPTESEC

Creating the System Vision



PANOPTESEC

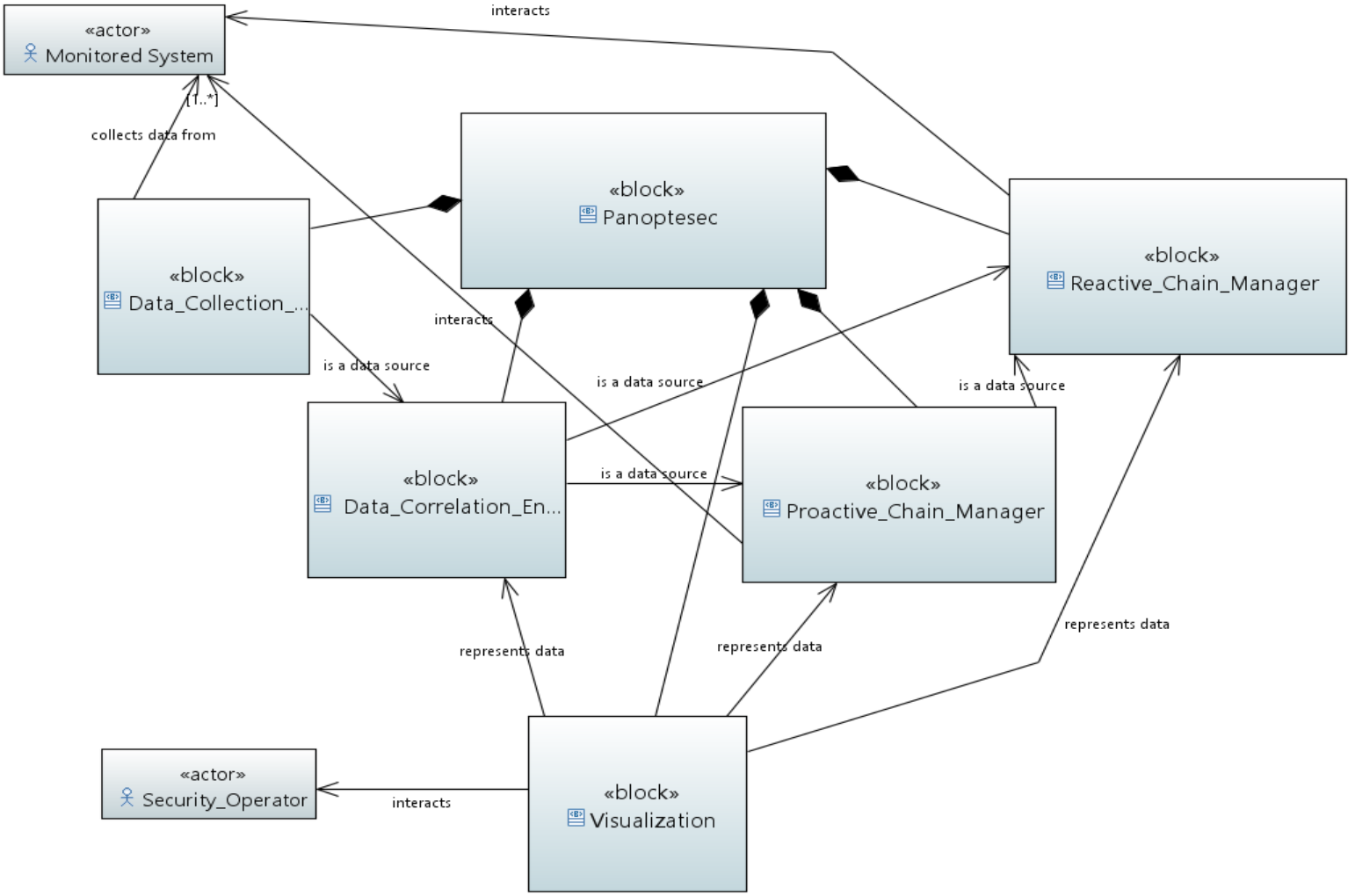
Create System Overview

After the first iteration of the methodology, a System Overview can be created:

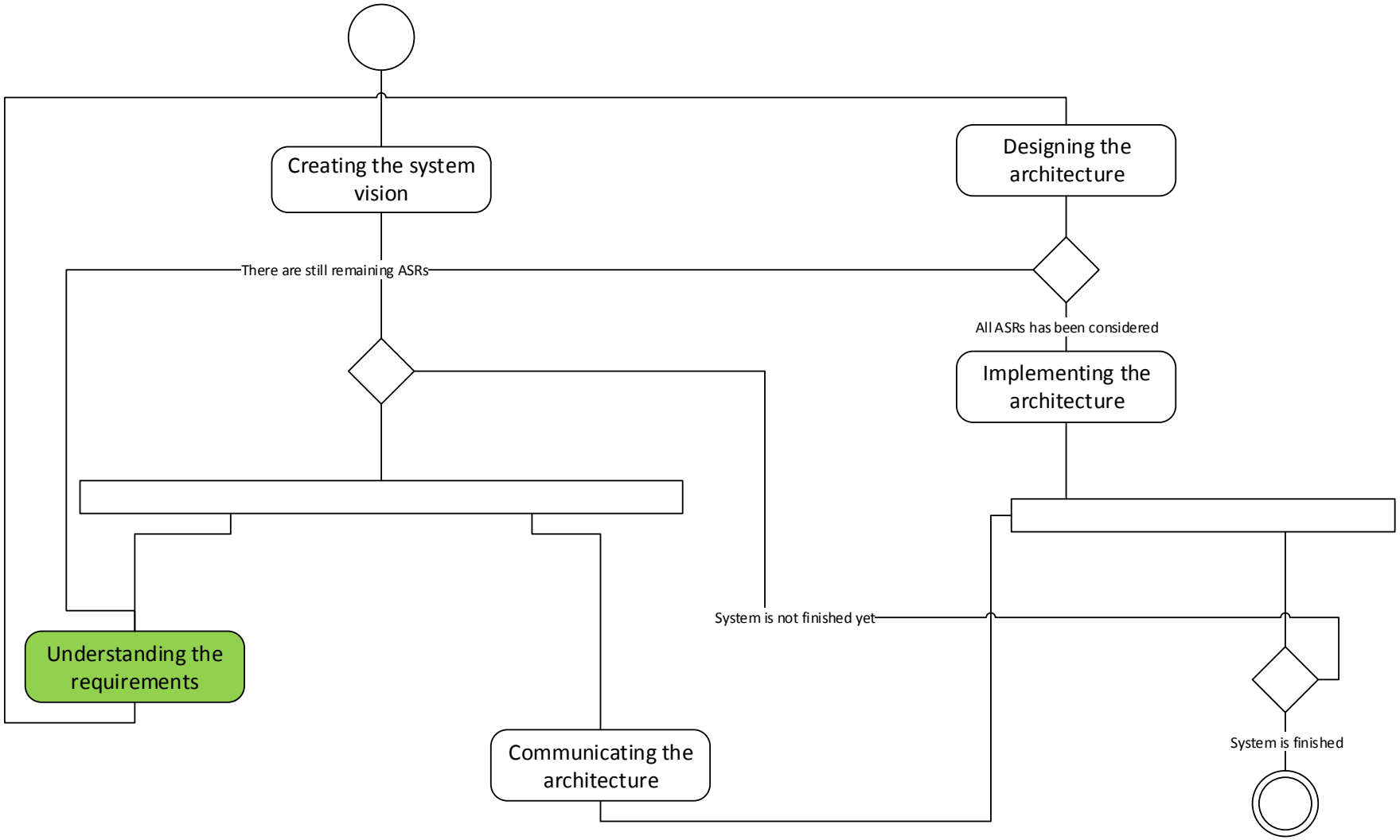
- Human and system actors are identified
- First functional decomposition
- Main relationships between components identified
- Main Non-functional constraints considered
- Main technical, business and social constraints considered

PANOPTESEC

Create System Overview - PANOPTESEC

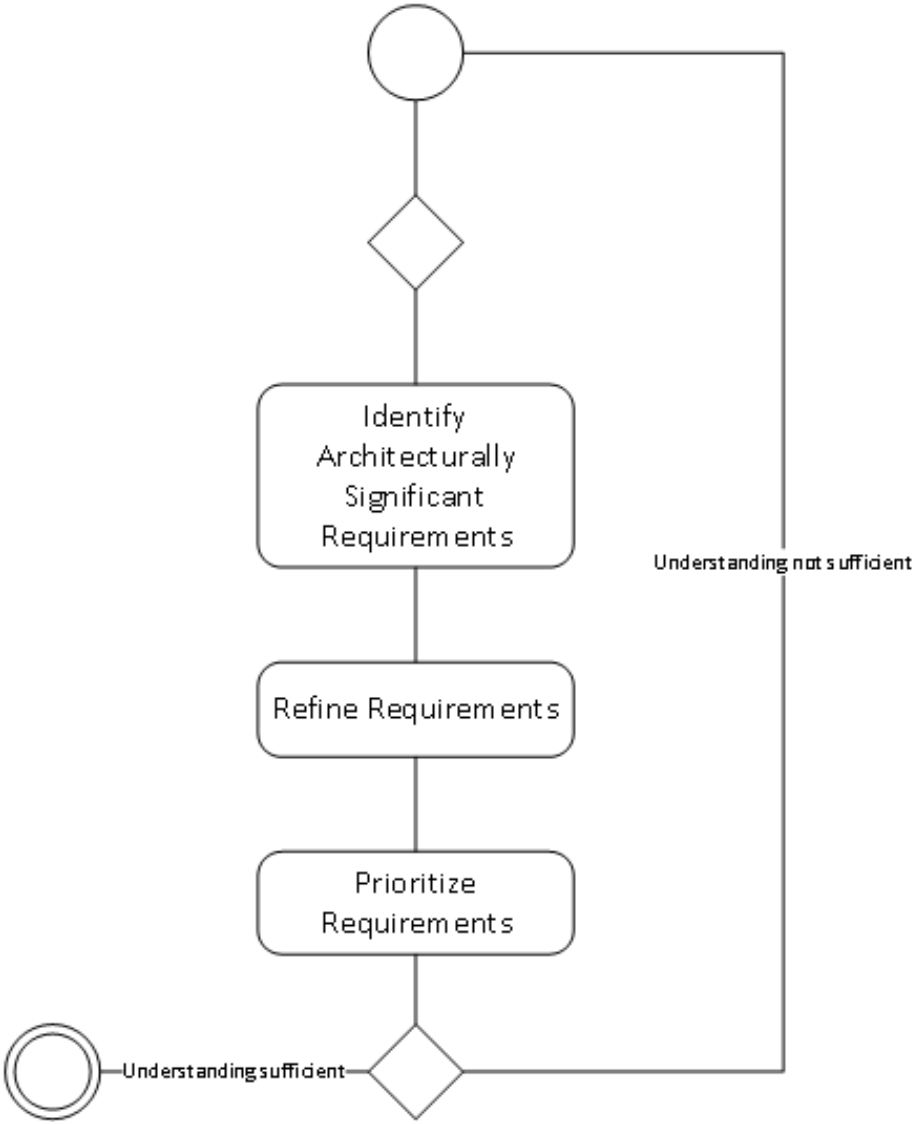


Understanding the Requirements



PANOPTESEC

Understanding the Requirements



PANOPTESEC

Identify Architecturally Significant Requirements

- Some Requirements have a far more profound impact on the architecture, and are commonly defined as Architecturally Significant Requirements, or Architecture Drivers
- ASRs are derived from Functional and Non-Functional Requirements
- Functional and Non-functional requirements have been evaluated by the Stakeholders for Importance and Reachability

Identify Architecturally Significant Requirements

- In essence, the style of the architecture is determined by the Non-Functional Architectural Requirements while the functional instances of the elements types defined by that style are determined by the Functional Architectural Requirements
- ASRs can be identified by evaluating the Importance of the identified System's Requirements, assuming that these Requirements already had been prioritized by the Stakeholders
- In addition, the Architectural Impact is evaluated on a scale between 1 (Low Impact – little effect on the architecture) and 3 (High Impact – profoundly affect the architecture)
- Requirements with both High Architectural Impact and Importance have to be considered among the ASRs.

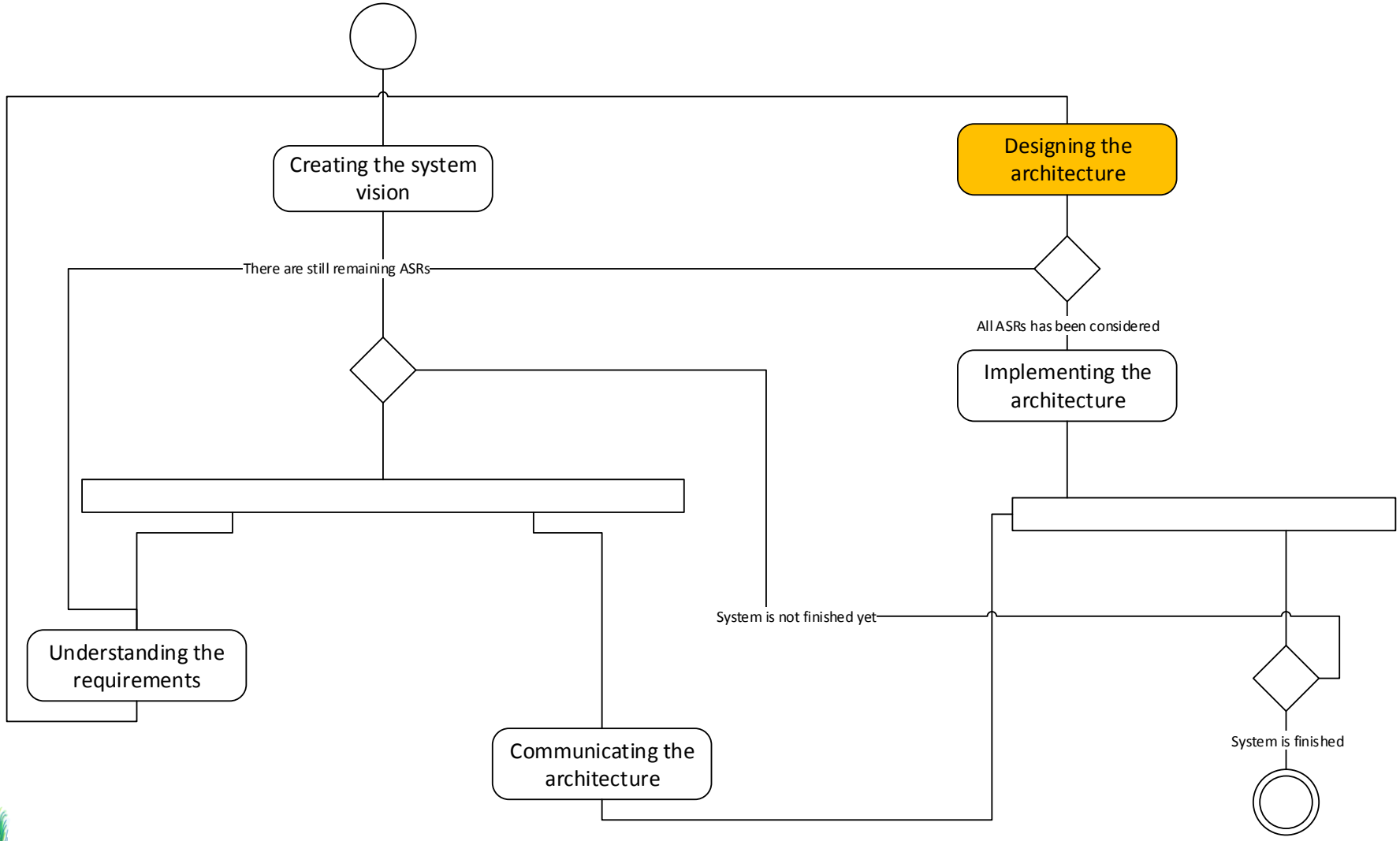
PANOPTESEC – Functional ASRs

Id	Description	Importance	Architectural Impact
DSC001	The PANOPTESEC system MUST provide a data collection system	3	3
DSC006	The data collection system MUST collect system configuration information.	3	3
DSC029	The data collection system MUST collect network and system events (e.g., cyber security alerts from intrusion detection systems).	3	3
ICA001	The PANOPTESEC system MUST contain an information correlation engine.	3	3
ICA006	The information correlation engine MUST store all information correlation results in the PANOPTESEC system.	3	3
ICA010	The PANOPTESEC system MUST include a mission impact model.	3	3
PRS001	The PANOPTESEC system MUST provide a proactive response system.	3	3
PRS022	The proactive response system MUST deploy the selected mitigation actions in the monitored system.	3	3
RRS001	The PANOPTESEC system MUST provide a reactive response system	3	3
RSS024	The reactive response system MUST deploy the selected mitigation actions in the monitored system.	3	3
VIZ001	The PANOPTESEC MUST provide a visualization system that displays cyber defense situational awareness in real-time.	3	3

PANOPTESEC – Non-Functional ASRs

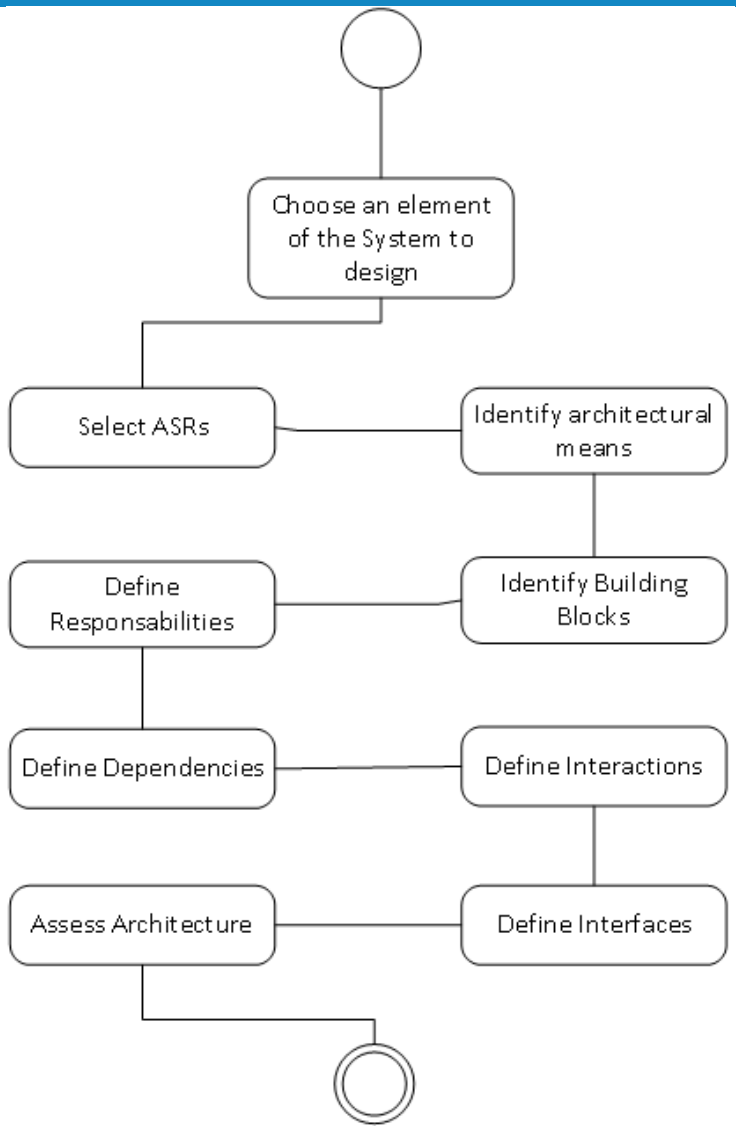
Id	Description	Importance	Architectural Impact
CMP001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve flexibility (in organization and configuration), ability to develop new function combinations rapidly and component re-use	3	3
CMP002	Communications between the various functional components of the PANOPTESEC System MUST be handled in a transparent way by a middleware in order to reduce complexity for developers.	3	3
MNT001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve maintainability.	3	3
PRT002	JAVA compliant interfaces MUST be provided by PANOPTESEC Components not written in JAVA.	3	3
PRT004	PANOPTESEC Components design MUST follow a component-oriented model in order to enhance reuse and separation of concerns.	3	3
RLB001	Unit test cases MUST be created for each PANOPTESEC component.	3	3
RLB011	Each instance of a computation in the proactive and reactive response chains MUST be performed on the most recently completed data collection and correlations results from the data collection and correlation components representing the most up to date state of the monitored system(s). For example, the most recently computed values for the Network Inventory, Vulnerability Inventory, Reachability Matrix and Low Level Correlation.	3	3
PRT005	PANOPTESEC Component designs MUST offer a clear and stable set of interfaces for use by other internal or external components.	3	3

Designing the Architecture



PANOPTESEC

Designing the Architecture



PANOPTESEC

Select Architecturally Significant Requirements

- A widely used design methodology, called Attribute Driven Design (ADD), is quite efficient during the design phase. ADD is an iterative method that, at each iteration, consists of:
  - Choosing a part of the system to design;
  - Evaluating and considering a set Architecturally Significant Requirement (ASR) related to that part, after their prioritization;
  - Creating a design for that part and evaluate it over the identified ASRs (Architectural Means are used in order to define building blocks: for each identified building block, Responsibilities, Dependencies and Interfaces must be identified).

- Each iteration of the design methodology considers the highest remaining priorities among the ASRs. At the end of the iterations, all ASRs are evaluated and considered. At that stage, all remaining Requirements are taken into account in order to finalize the architecture

Identify Architectural Means

- The selection of Architectural Means is a key procedure in every software project.
- Common Principles, Patterns, Tactics give the architect powerful drivers for building the architecture with respect of the Non-Functional ASRs
- Architectural means are no silver bullet: there is always a trade-off that must be considered with respect of the priorities of the ASRs

PANOPTESEC

Architectural Principles

Architecture Principles are general principles that should be considered when designing architectures, according to the top priority Non-Functional ASRs. Among the most important Principles:

- **Loose Coupling**: this Principle states that the coupling between building blocks should be kept as low as possible. The less strongly a building block is coupled with other building blocks, the easier it is to understand, design and maintain the building block in order to improve modifiability and compatibility of the system. Loosely coupled architecture can increase complexity or resource consumption.
- **Design for Change**: this Principle is tightly coupled with the Loose Coupling Principle. Design for Change makes the architect design the system in order to be able to manage probable changes. The Principle of Loose Coupling allows the system to be ready for change: updated in functionalities can be concentrated only over the identified building blocks.
- **Separation of Concerns**: one of the most important uses of this Principle (tightly coupled with the Loose Coupling Principle) is to support modularization. This primarily means identifying parts of a software system responsible for specific concerns, aspects or tasks and encapsulating them as separate building blocks.

Architectural Principles

- **Information Hiding**: this Principle is a fundamental concept for structuring and understanding complex systems. This Principle (when applied to software architecture) states that a building block should show to client building blocks ONLY that part of information that is really necessary for the client's tasks. This brings to the definition of well-designed interfaces between components. This Principle is strictly coupled with the Loose Coupling Principle.
- **Consistency**: this Principle states that architecture should follow a standard set of rules from beginning to end: naming convention, communication of the system building blocks, structure of the interfaces, etc.
- **Explicit Interfaces**: this sub-principle states that a system building block should clearly show which other building blocks it communicates with.
- **Separation of interfaces and Implementation**: Interfaces should be described separately from the implementation so that the interface client can rely on the interface without knowing the implementation details.
- **Language support for abstractions**: this sub-principle states that both the design and the programming languages should support the definition of the architectural abstractions (interfaces and components, for example).

Architectural Patterns

- While the Architectural Principles describe wide guidelines that help the architect developing system's design, Tactics and Patterns offer more practical solutions to architectural issues raised by the Non-Functional ASRs that drive the architecture. Chosen Patterns and Tactics have to respect the Architectural Principles previously identified.
- An *Architectural Pattern* is a package of design decisions that is found repeatedly in practice, has known properties that permit reuse and describes a class of architectures. Architectural Patterns are commonly used in order to fulfil a set of Non-Functional ASRs related to different but related Quality Attributes.
- For a complex system, it is not uncommon to rely on multiple architectural patterns: in order to maintain cohesion and consistency, the architect will always keep in mind selected Architectural Principles. Each pattern is meant to solve a set of architectural problems and to enhance a specific set of quality attributes. Each pattern brings solution to the identified needs along with a possible set of weaknesses that must be evaluated by the architect on behalf of the Non-Functional ASRs and the relative quality attributes.

Architectural Patterns

Pipe-and-Filter Pattern

Context:

Many systems are required to transform streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts.

Addressed Problem:

Such systems need to be divided into reusable, loosely coupled component with simple, generic interaction mechanism. In this way they can be flexibly combined with each other. The components, being generic and loosely coupled, are easily reused. The component, being independent, can execute in parallel.

Solution:

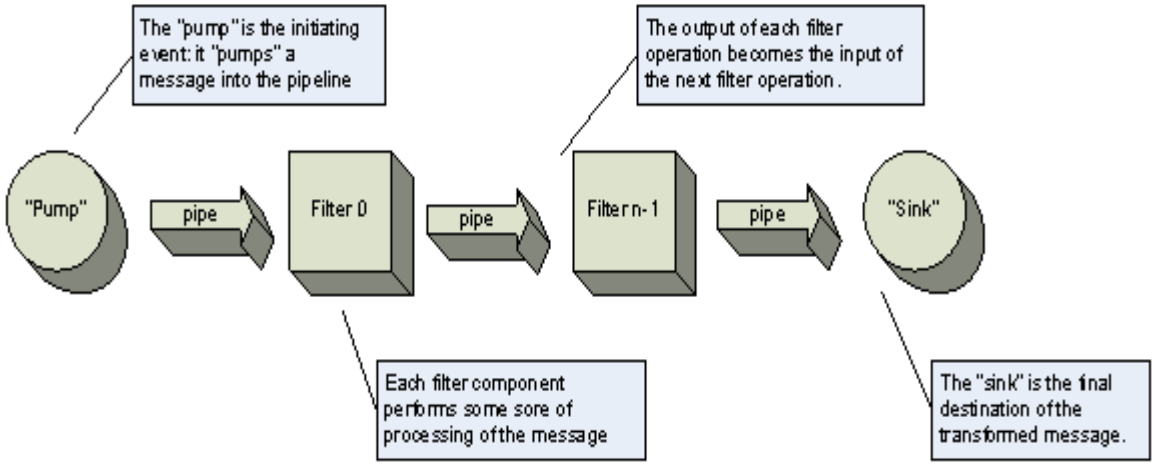
The pattern of interaction in this pattern is characterized by successive transformations of sets of data. Data arrives at a filter's input port(s), is transformed and then is passed via its output port(s) through a pipe to the next filter. A single filter can consume data from, or produce data to one or more ports.

Advantages:

The advantage of a pipes and filters architecture is that it is very flexible with regard to the combination of pipes and filters. It is possible to easily reuse filter components. It is easy to let pipes and filters work in parallel, for example, in separate processes or threads, since they are very independent of one another. This can increase the efficiency of the entire system.

Weaknesses:

This pattern is not typically a good choice for an interactive system. The forwarding of the status between filters can involve a high effort and resource consumption. If data has to be transformed to be placed in the pipe, then unnecessary bidirectional transformations may occur. Debugging or the behaviour in the case of errors can be more difficult than with other architectures because errors or debugging information has to be sent through the pipes.



Architectural Patterns

Broker Architecture

Context:

Focus is on distributed, modular object system. This means that objects are to be made available in a server process (the broker) and they are to be accessed by distributed clients via the network.

Addressed Problem:

A distributed system presents many challenges that do not occur in a local system that runs in a single process. An important challenge in this area is the communication via non-permitted networks—in contrast to local calls, a network can fail without the client or server failing. Furthermore, heterogeneous components must be brought into a coherent architecture and the distributed resources must be used efficiently. If developers from each of the distributed components had to master all of these challenges, an additional level of complexity would be added to their efforts.

Solution:

The outsourcing of all communication tasks to a broker separates the communication tasks of a distributed system from its application logic. The broker hides and controls the communication between the objects or components of the distributed system. On the client side, the broker establishes the distributed calls and then forwards them to the server. On the server side, it receives the request and establishes a call from it, which it then executes on a server object. In the same way the broker returns the response to the client. The broker takes over all details of the distributed communication, such as establishing the connection, marshalling the message (converting the data sent into the message format), etc., and hides these details from the client and the distributed object as far as possible.

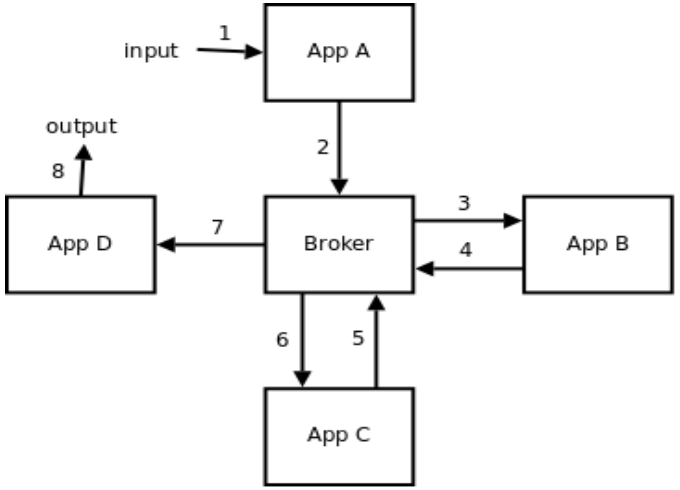
Advantages:

A broker has the advantage that it abstracts from and simplifies distributed communication. The broker infrastructure can be reused by different distributed applications.

Since the broker is responsible for finding the server or the distributed object via a symbolic name or an ID, it allows transparency of the location at which the distributed object is really located in the network.

Weaknesses:

Broker architecture typically has slightly worse performance and consumes more resources than a distributed architecture in which static, distributed objects are connected directly to the network. A broker has a certain complexity that must be understood. For very simple applications, for example, in the area of embedded systems, more simple architectures may bring the same benefits as Broker Architecture but are easier to maintain and to understand. For most other distributed systems, such as in the enterprise domain, the use of a broker is recommended.



Architectural Patterns

Publish-Subscribe Pattern

Context:

There are a number of independent producers and consumers of data that must interact. The precise number and nature of the data producers and consumers are not predetermined or fixed, nor is the data that they share.

Addressed Problem:

How it is possible to create integration mechanisms that support the ability to transmit messages among the producers and consumers in such a way that they are unaware of each other identity, or potentially even their existence?

Solution:

In this pattern, components interact via announced messages, or events. Components may subscribe to a set of events. It is the job of the publish-subscribe runtime infrastructure to make sure that each published event is delivered to all subscribers of that event.

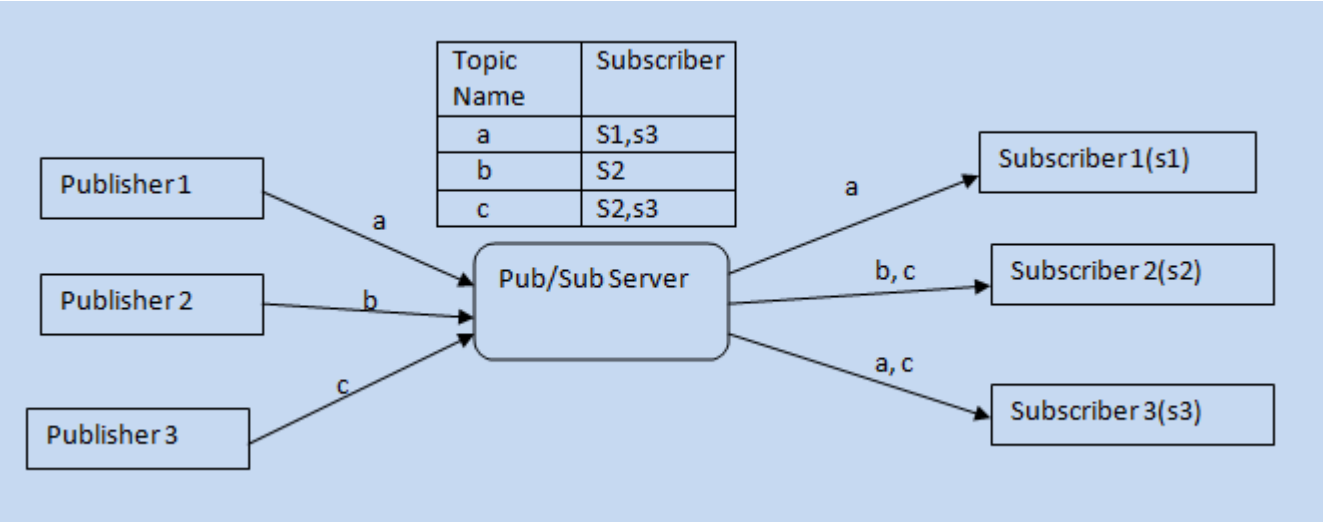
Advantages:

Publishers are loosely coupled to subscribers, and need not even know of their existence. With the topic being the focus, publishers and subscribers are allowed to remain ignorant of system topology. Each can continue to operate normally regardless of the other. In the traditional tightly coupled client–server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running.

Publish-subscribe provides the opportunity for better scalability than traditional client–server, through parallel operation, message caching, tree-based or network-based routing, etc.

Weaknesses:

This pattern typically increases latency and had a negative effect on scalability and predictability of message delivery time. Less control over ordering of messages, and delivery of messages is not guaranteed.



Architectural Patterns

Service Oriented Architecture

Context:

A number of services are offered (and described) by service providers and consumed by service consumers. Service consumers need to be able to understand and use these services without any detailed knowledge of the implementation.

Addressed Problem:

How it is possible to support interoperability of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across a network? How it is possible to locate services and combine (and dynamically recombine) them into meaningful coalitions while achieving reasonable performance, security and availability?

Solution:

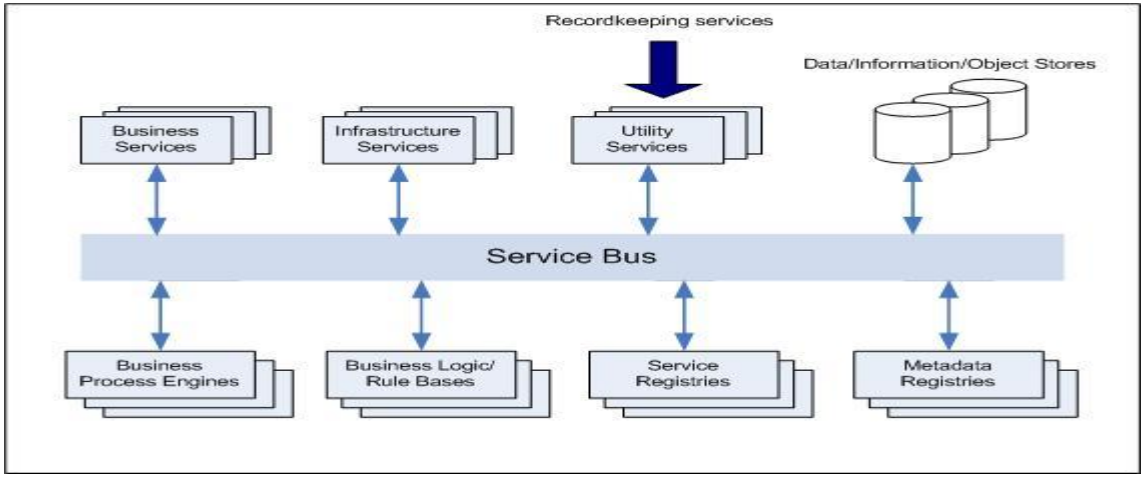
The Service Oriented Architecture Pattern describes a collection of distributed components that provide and/or consume services. Services are largely standalone: service providers and service consumers are usually deployed independently and often belong to different systems or even different organizations. Components have interfaces that describe the service they request from other components and the service they provide. Orchestration mechanisms are possible in order to describe the interactions among various service consumers and providers.

Advantages:

Service Oriented Architecture Pattern allows developing a complex system by integrating different components produced by different organizations, independent of the platform and technology. Thus, it helps to manage complexity involved. Interoperability, reliability, reusability and maintainability are enhanced.

Weaknesses:

Service Oriented Architecture systems are complex to build. It can be difficult to control the evolution of the different services/components. Orchestration mechanisms, while effective, can be complex to implement and to maintain if a change is induced in the system. Service Oriented Architectures add a performance overhead associated to the middleware and to the common messaging connector used (SOAP, REST). Complexity and performance may be affected if new services are added to the system



PANOPTESEC

Architectural Patterns

Shared-Data Pattern

Context:

Various computational components need to share and manipulate large amount of data, this data do not belong solely to any one of those components.

Addressed Problem:

How can systems store and manipulate persistent data that is accessed by multiple independent components?

Solution:

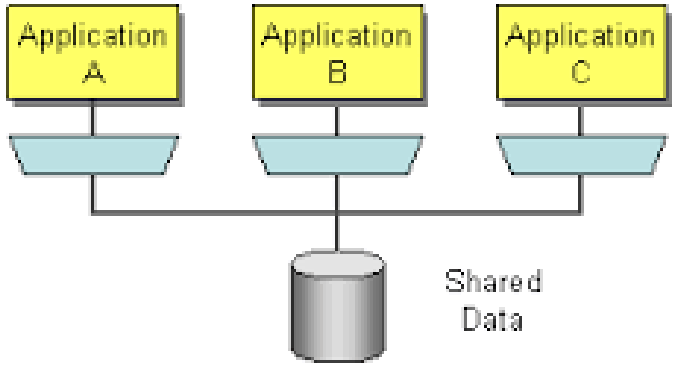
In the Shared-Data pattern, interaction is dominated by the exchange of persistent data between multiple data accessors and at least one shared-data store. The general computational model associated with shared-data systems is that data accessors perform operations that require data from the data store and write results to one or more data stores. That data can be viewed and acted on by other data accessors.

Advantages:

This Pattern is useful whenever various data items are persistent and have multiple accessors. Use of this pattern has the effect of decoupling the producer of the data from the consumers of the data while maintaining a common data persistency between them.

Weaknesses:

The shared-data store may be a performance bottleneck. The shared-data store may be a single point of failure. Producers and consumers of data may be tightly coupled, through their knowledge of the structure of the shared data.



PANOPTESEC

Architectural Patterns

Client-Server Pattern

Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests

Context:

There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service

Addressed Problem:

By managing a set of shared resources and services, we can promote modifiability and reuse, by factoring out common services and having to modify these in a single location, or small number of location. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers

Solution:

Clients interact by requesting services of serves, which provide a set of services. Some components may act as both clients and servers. There may be one central server or multiple distributed ones

Advantages:

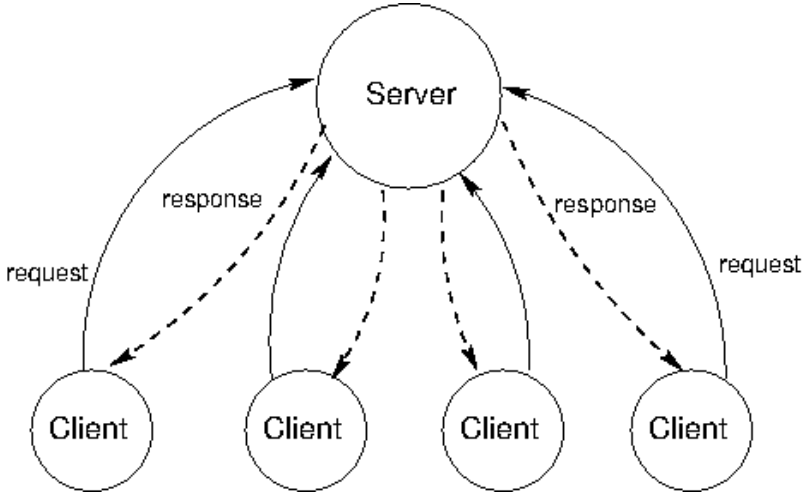
This pattern simplifies systems by factoring out common services, which are reusable. Because servers can be accessed by any number of clients, it is easy to add new clients to a system. Similarly, servers may be replicated to support scalability or availability

Weaknesses:

Server can be a performance bottleneck.

Servers can be a single point of failure

Decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after the system has been built



Architectural Patterns

Many other patterns are described in literature:

- Multi-Tier Pattern
- Peer-to-Peer Pattern
- Layered Pattern
- …

Architectural Patterns

Patterns are useful in order to define an architecture, depending on the identified ASRs. It is possible to identify which Pattern works better with respect of a particular Quality Attribute

Pattern Name	Maintenability	Portability	Compatibility	Performance	Security	Usability	Reliability
Pipe-and-Filter	Strengths	Strengths	Strengths	Weaknesses	Weaknesses	Weaknesses	Strengths and Weaknesses
Broker Architecture	Strengths	Strengths	Strengths	Weaknesses	Strengths	Strengths	Neutral
Publish-Subscribe	Strengths	Strengths	Strengths	Weaknesses	Neutral	Strengths	Weaknesses
Service Oriented Architecture	Strengths	Strengths	Strengths	Weaknesses	Strengths	Neutral	Strengths and Weaknesses
Shared-Data	Strengths and Weaknesses	Strengths and Weaknesses	Strengths and Weaknesses	Weaknesses	Strengths	Strengths	Strengths and Weaknesses
Client-Server	Weaknesses	Strengths and Weaknesses	Strengths	Strengths and Weaknesses	Strengths	Strengths	Strengths and Weaknesses

PANOPTESEC - Architectural Patterns

From the Non-Functional ASRs analysis a particular emphasis on some specific Quality Attributes emerges including:

- *Compatibility;*
- *Maintainability;*
- *Portability;*
- *Reliability*

Id	Description	Possible Fulfilling Patterns
CMP001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve flexibility (in organization and configuration), ability to develop new function combinations rapidly and component re-use	Pipe-and-Filter, Publish-Subscribe, Broker Architecture, Service Oriented Architecture
CMP002	Communications between the various functional components of the PANOPTESEC System MUST be handled in a transparent way by a middleware in order to reduce complexity for developers.	Broker Architecture, Publish-Subscribe, Service Oriented Architecture
MNT001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve maintainability.	Pipe-and-Filter, Service Oriented Architecture, Publish-Subscribe, Broker Architecture
PRT002	JAVA compliant interfaces MUST be provided by PANOPTESEC Components not written in JAVA.	Service-Oriented Architecture, Broker Architecture
PRT004	PANOPTESEC Components design MUST follow a component-oriented model in order to enhance reuse and separation of concerns.	Pipe-and-Filter, Service Oriented Architecture, Broker Architecture, Publish-Subscribe
RLB001	Unit test cases MUST be created for each PANOPTESEC component.	Pipe-and-Filter, Broker Architecture, Service Oriented Architecture
RLB011	Each instance of a computation in the proactive and reactive response chains MUST be performed on the most recently completed data collection and correlations results from the data collection and correlation components representing the most up to date state of the monitored system(s). For example, the most recently computed values for the Network Inventory, Vulnerability Inventory, Reachability Matrix and Low Level Correlation.	Pipe-and-Filter, Shared-Data, Service Oriented Architecture
PRT005	PANOPTESEC Component designs MUST offer a clear and stable set of interfaces for use by other internal or external components.	Pipe-and-Filter, Broker Architecture, Service Oriented Architecture

PANOPTESEC - Architectural Patterns

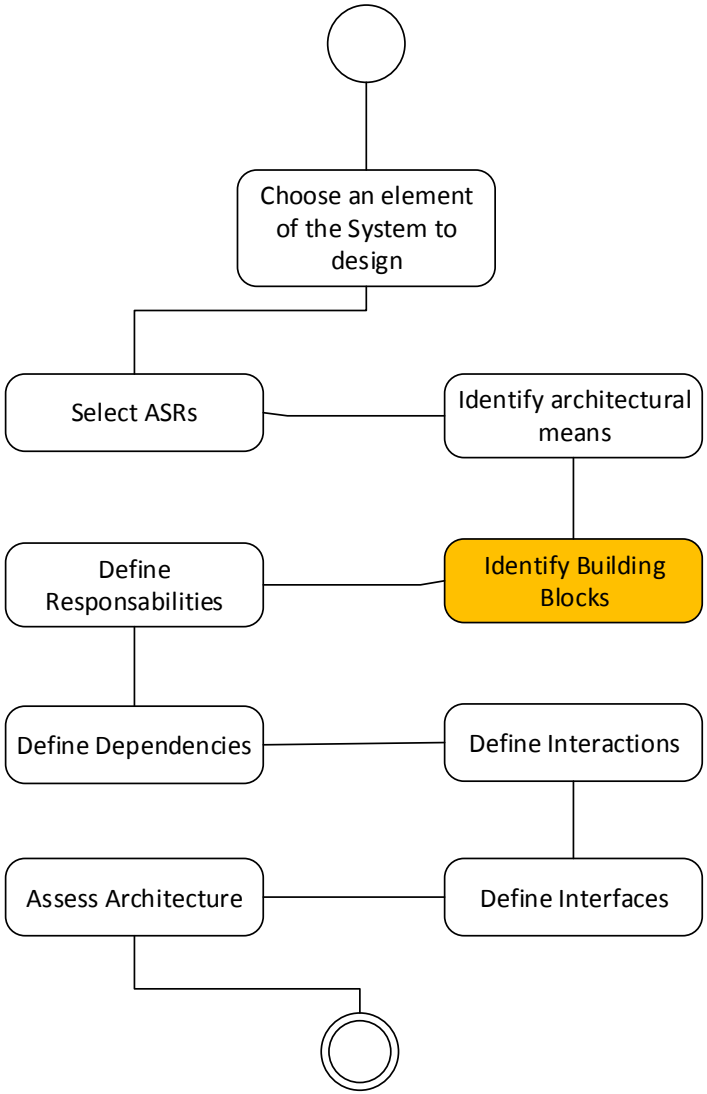
One of the main consequences related to the evaluation of the top-priority Non-Functional ASRs and the choice of the Architectural Patterns is the need for the PANOPTESEC System to be able to implement the chosen **Broker Architecture**, to have the capability of building the **Pipe-and-Filter** chains (possibly by using orchestration mechanisms within the technology used in order to implement the Broker) and to use **Service Oriented Principles** if needed.

This issue can be solved by building an Integration Framework. Integration Frameworks can be used to integrate different technologies, applications and products without needing to write a lot of glue code. Connectors, implicit type converters, domain specific languages and Enterprise Integration Patterns are already implemented in the framework.

Some other benefits of an Integration Framework include:

- More lightweight than an Enterprise Service Bus (that are usually based on Service Oriented Architectures);
- Can be attached as simple libraries (usually, Java based and so perfectly interoperable among different platforms and different Operating Systems) into a project (for example, in Eclipse);
- Are open source and usually perfectly integrated with Java applications;
- Feature great flexibility.
- Integration Frameworks are also beneficial because they reduce complexity in intricate integrations. They support architects and developers by realizing routing, connectivity, error handling and testability.

Identify Building Blocks



PANOPTESEC

Identify Building Blocks

- The functional building blocks are concerned with the first System Overview and the associated ASRs Functional Requirements
- Functional building blocks are designed by identifying functional needs of the System as described in Functional Requirements
- A building block identified in this way also takes care of a significant functional concern.

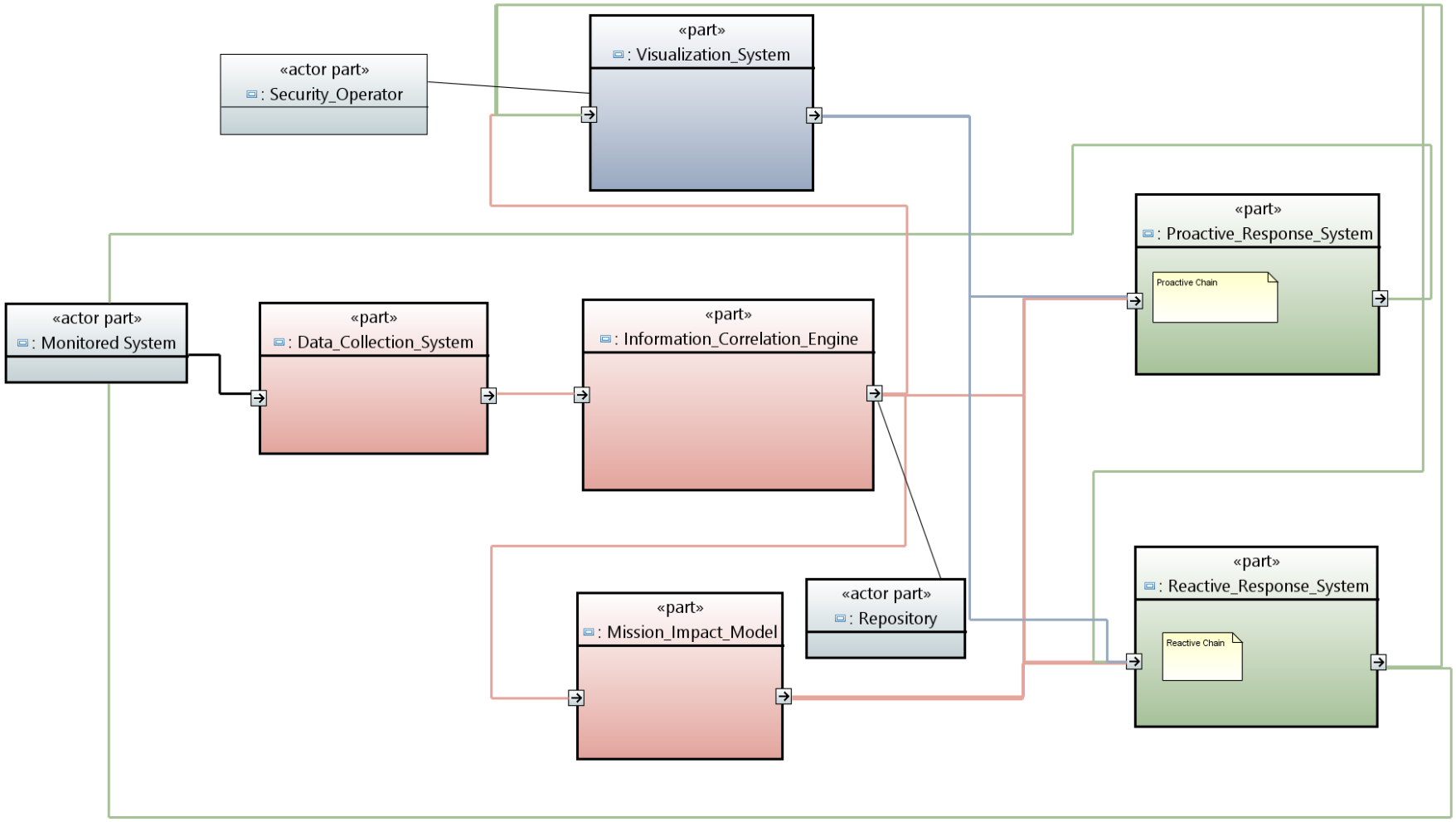
PANOPTESEC

PANOPTESEC – Functional ASRs

Considering the top Functional ASRs (Requirements with Importance 3 and Architectural Impact 3) for the PANOPTESEC System it is possible to derive the first functional architecture (in SysML).

Id	Description	Importance	Architectural Impact
DSC001	The PANOPTESEC system MUST provide a data collection system	3	3
DSC006	The data collection system MUST collect system configuration information.	3	3
DSC029	The data collection system MUST collect network and system events (e.g., cyber security alerts from intrusion detection systems).	3	3
ICA001	The PANOPTESEC system MUST contain an information correlation engine.	3	3
ICA006	The information correlation engine MUST store all information correlation results in the PANOPTESEC system.	3	3
ICA010	The PANOPTESEC system MUST include a mission impact model.	3	3
PRS001	The PANOPTESEC system MUST provide a proactive response system.	3	3
PRS022	The proactive response system MUST deploy the selected mitigation actions in the monitored system.	3	3
RRS001	The PANOPTESEC system MUST provide a reactive response system	3	3
RSS024	The reactive response system MUST deploy the selected mitigation actions in the monitored system.	3	3
VIZ001	The PANOPTESEC MUST provide a visualization system that displays cyber defense situational awareness in real-time.	3	3

PANOPTESEC – First Functional Architecture



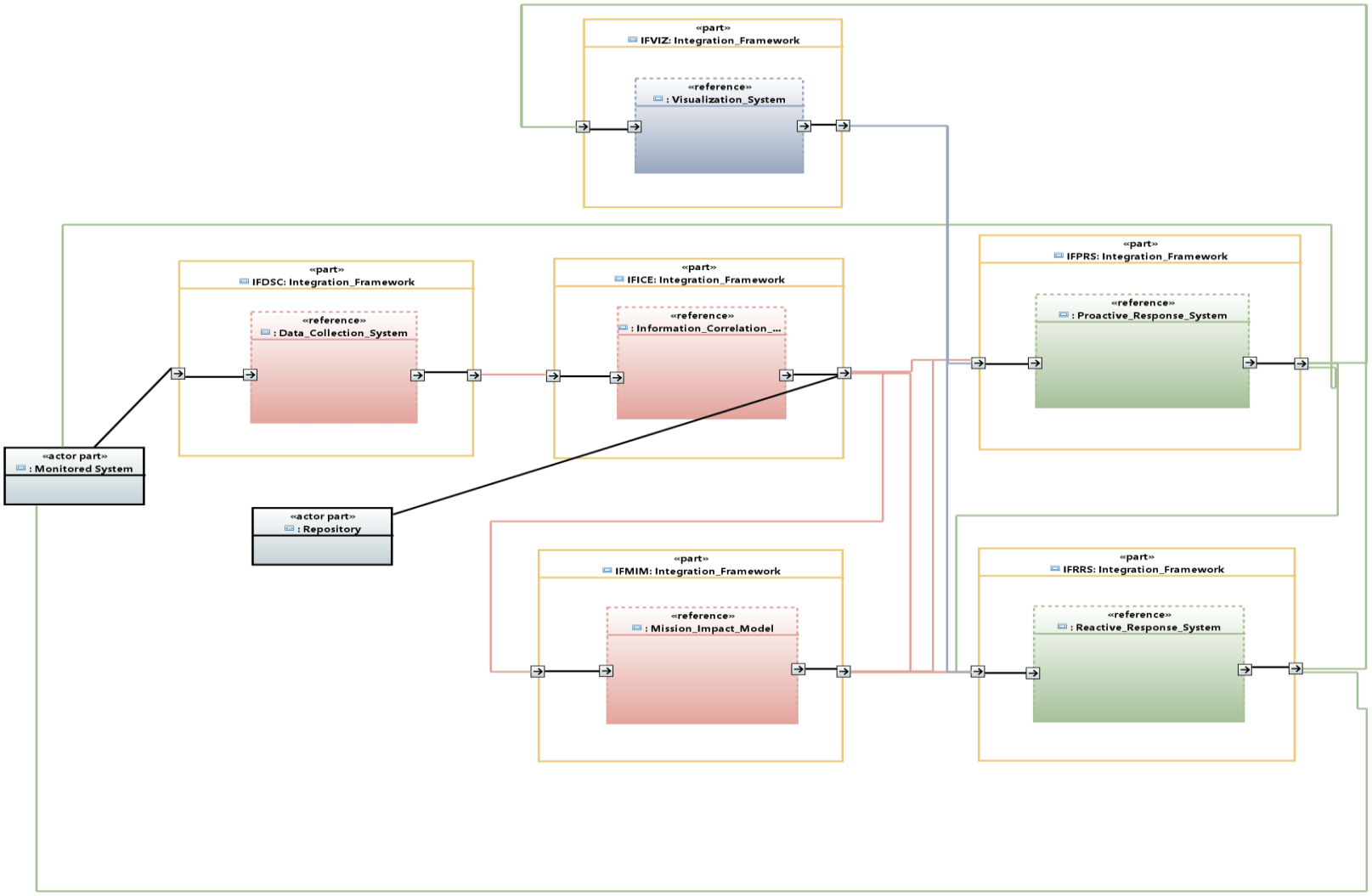
PANOPTESEC

PANOPTESEC – Non-Functional ASRs

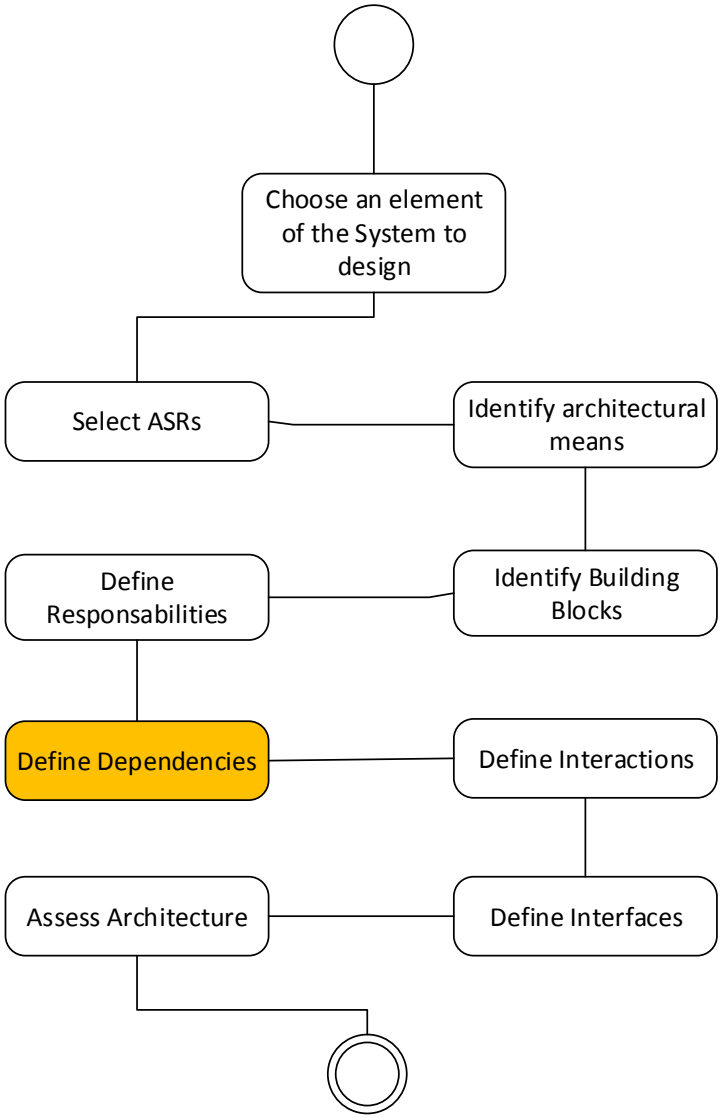
Considering the top Non-Functional ASRs (Requirements with Importance 3 and Architectural Impact 3) for the PANOPTESEC System and the first functional architecture, it is possible to derive the first architecture (in SysML), after the application of architectural means.

Id	Description	Importance	Architectural Impact
CMP001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve flexibility (in organization and configuration), ability to develop new function combinations rapidly and component re-use	3	3
CMP002	Communications between the various functional components of the PANOPTESEC System MUST be handled in a transparent way by a middleware in order to reduce complexity for developers.	3	3
MNT001	The PANOPTESEC System MUST be modular and decomposed in different components in order to improve maintainability.	3	3
PRT002	JAVA compliant interfaces MUST be provided by PANOPTESEC Components not written in JAVA.	3	3
PRT004	PANOPTESEC Components design MUST follow a component-oriented model in order to enhance reuse and separation of concerns.	3	3
RLB001	Unit test cases MUST be created for each PANOPTESEC component.	3	3
RLB011	Each instance of a computation in the proactive and reactive response chains MUST be performed on the most recently completed data collection and correlations results from the data collection and correlation components representing the most up to date state of the monitored system(s). For example, the most recently computed values for the Network Inventory, Vulnerability Inventory, Reachability Matrix and Low Level Correlation.	3	3
PRT005	PANOPTESEC Component designs MUST offer a clear and stable set of interfaces for use by other internal or external components.	3	3

PANOPTESEC – First Architecture Draft



Define Dependencies

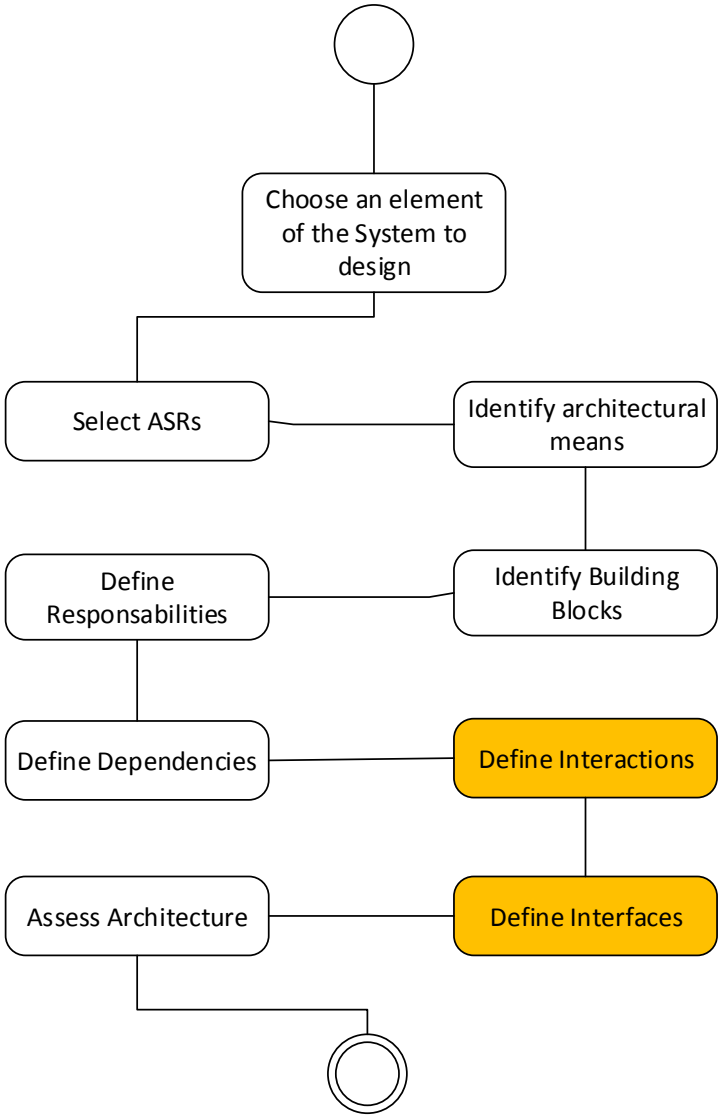


PANOPTESEC

Define Dependencies

- Once the building blocks are identified and their responsibilities documented, it is possible to define their dependencies.
- The dependencies result on one hand from the functional conditions and on the other from the non-functional conditions.
- It is possible to derive the functional dependencies from the relationships between the Functional ASRs (a SysML Requirements Diagram is usually very useful) or from the functional key abstractions identified in the System Overview.
- Non-functional dependencies result from the architecture means selected for designing the architecture

Define Interactions and Interfaces



PANOPTESEC

Define Interactions and Interfaces

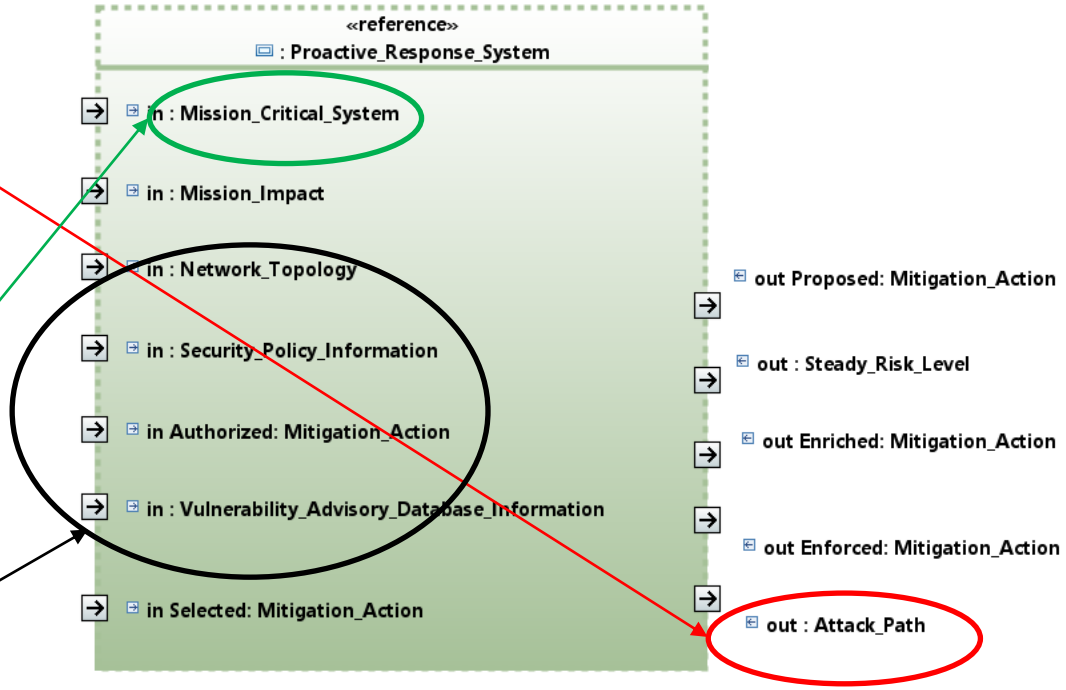
- By defining the interactions it is possible to document the dynamic relationships between the software building blocks at an architectural level.
- ASRs are analyzed in order to determine which building blocks communicate with each other and how, in order to realize the functionality to be delivered.
- This action is closely related to the definition of the interfaces, since the interactions are expressed by the service calls and the transfer of data to interfaces.
- By illustrating the significant features of the interactions, it is possible to determine the strengths and weaknesses of an architecture: this is particularly important for the assessment of the architecture.

Define Interactions and Interfaces

- UML communication and sequence diagrams are frequently used to model the interactions
- Building blocks communicate with one another via interfaces.
- Blocks offer their services via interfaces and use functionality from other building blocks.
- Interfaces are abstractions of concrete building block implementations. Therefore, implementation-specific aspects should not be visible in an interface.
- By identifying interfaces within building blocks it is possible to define the common data model of the System (at the level of the High Level Data Model)

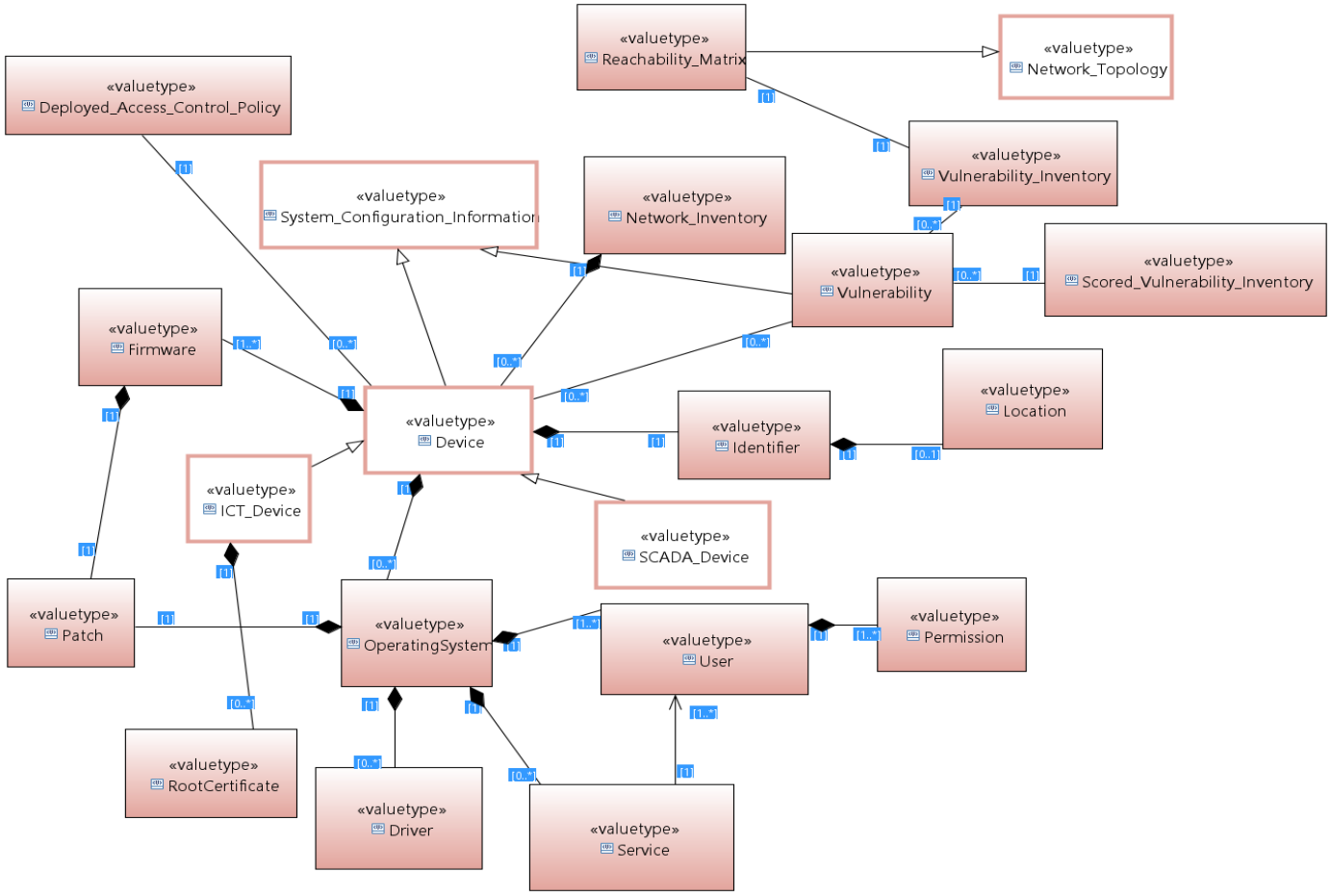
PANOPTESEC - Define Interactions and Interfaces

PRS002	The proactive response system MUST evaluate the potential cyber security attack paths present in the system.
PRS003	The proactive response system MUST evaluate the potential cyber security attack paths from hypothetical attack sources to identified mission critical systems.
PRS004	The evaluation of potential cyber security attack paths by the proactive response system MUST take into account system configuration information.



PANOPTESEC - Define Interactions and Interfaces

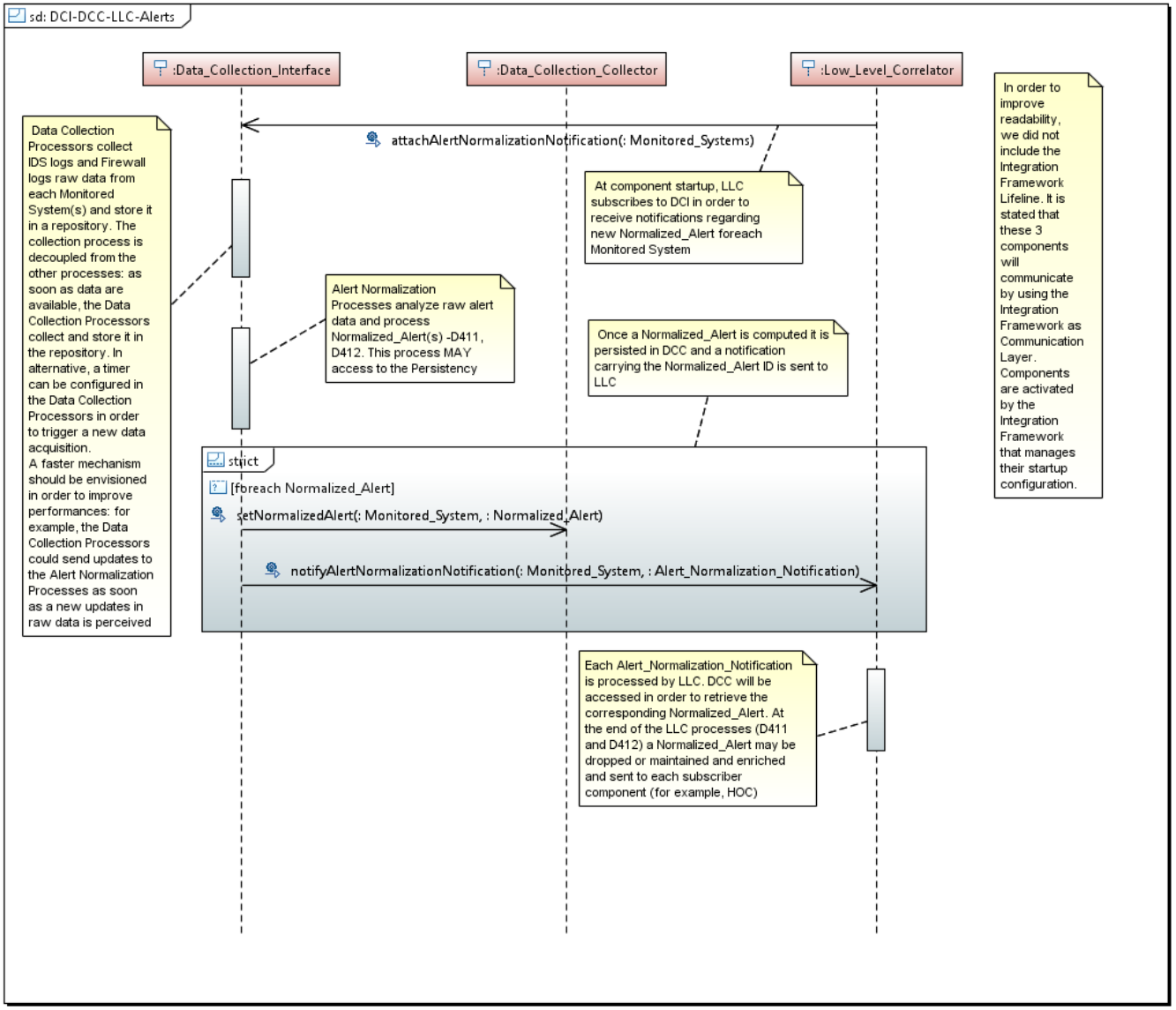
High Level Data Model



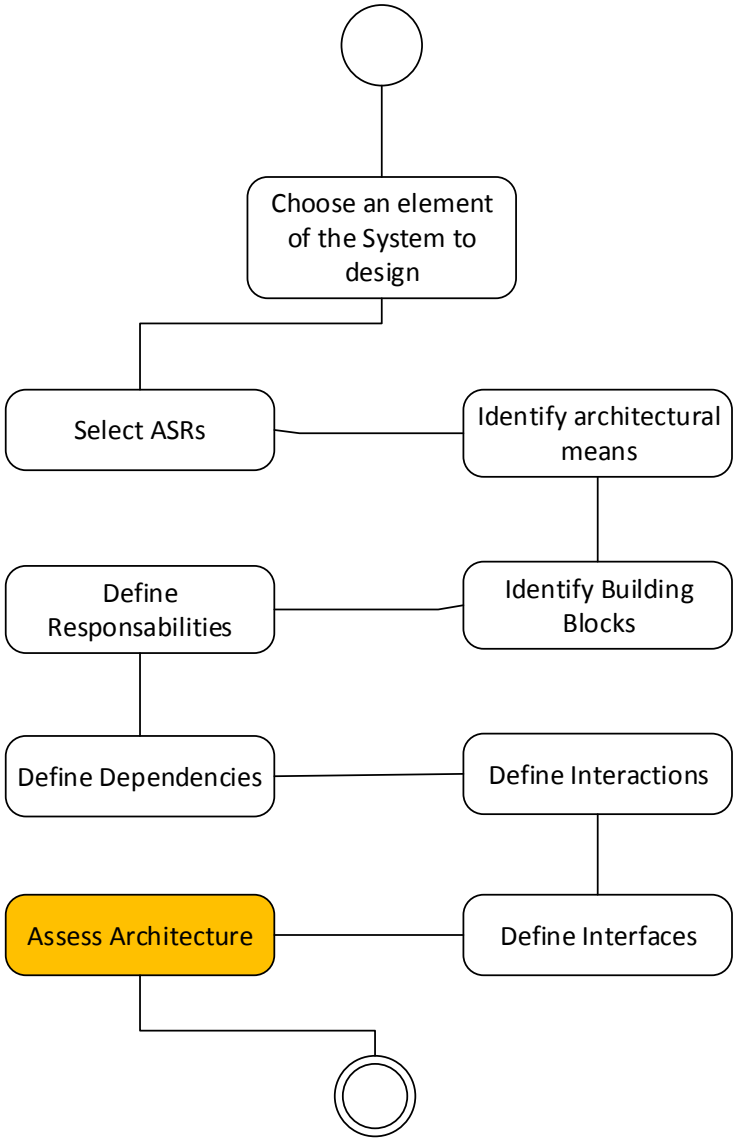
PANOPTESEC - Define Interactions and Interfaces

Sequence Diagrams

DSC029	**The data collection system MUST collect network and system events (e.g., cyber security alerts from intrusion detection systems).**



Assess architecture



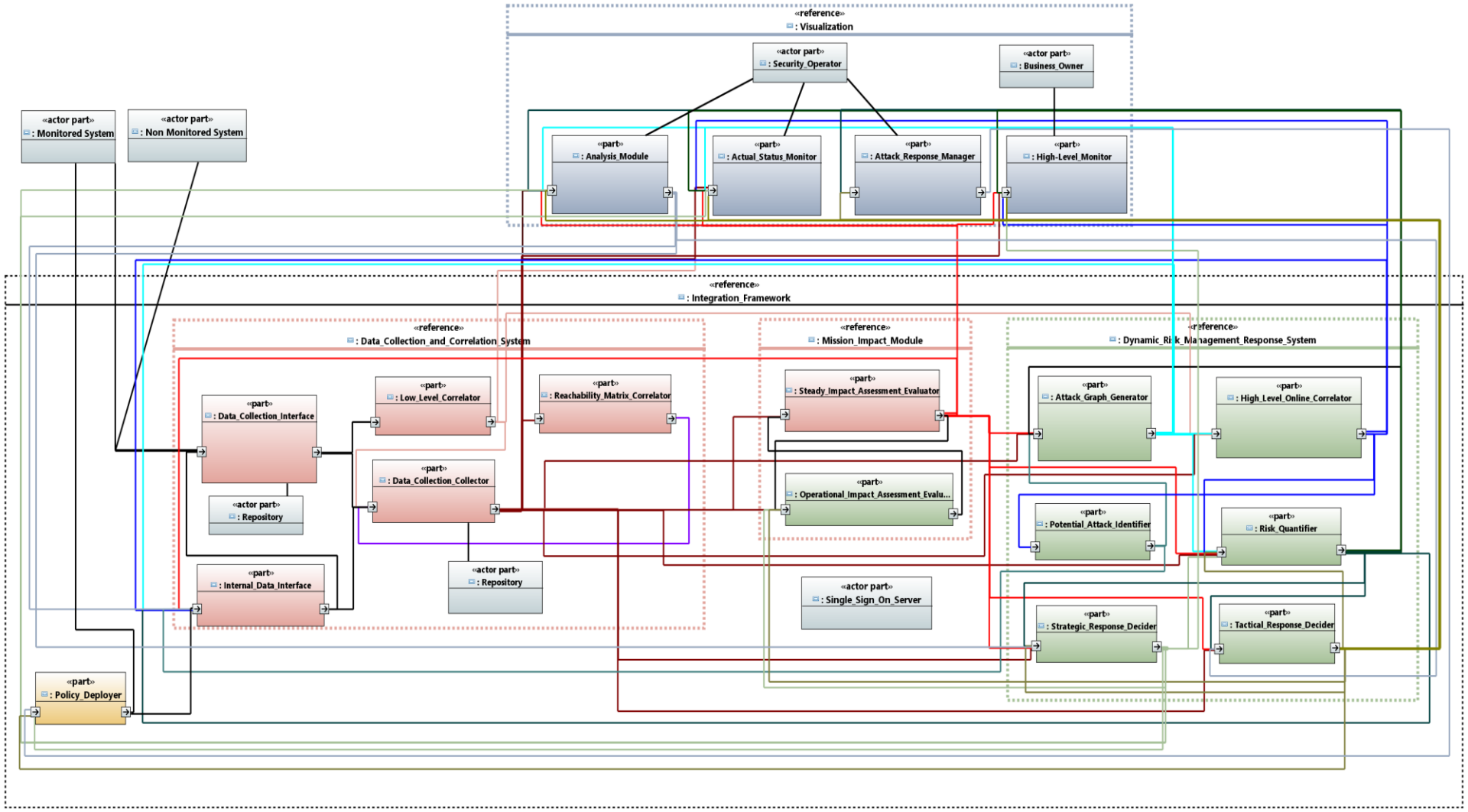
PANOPTESEC

Assess architecture

At the end of each design iteration, it is important to assess if the proposed architecture fulfil the selected ASRs. Various methods are possible:

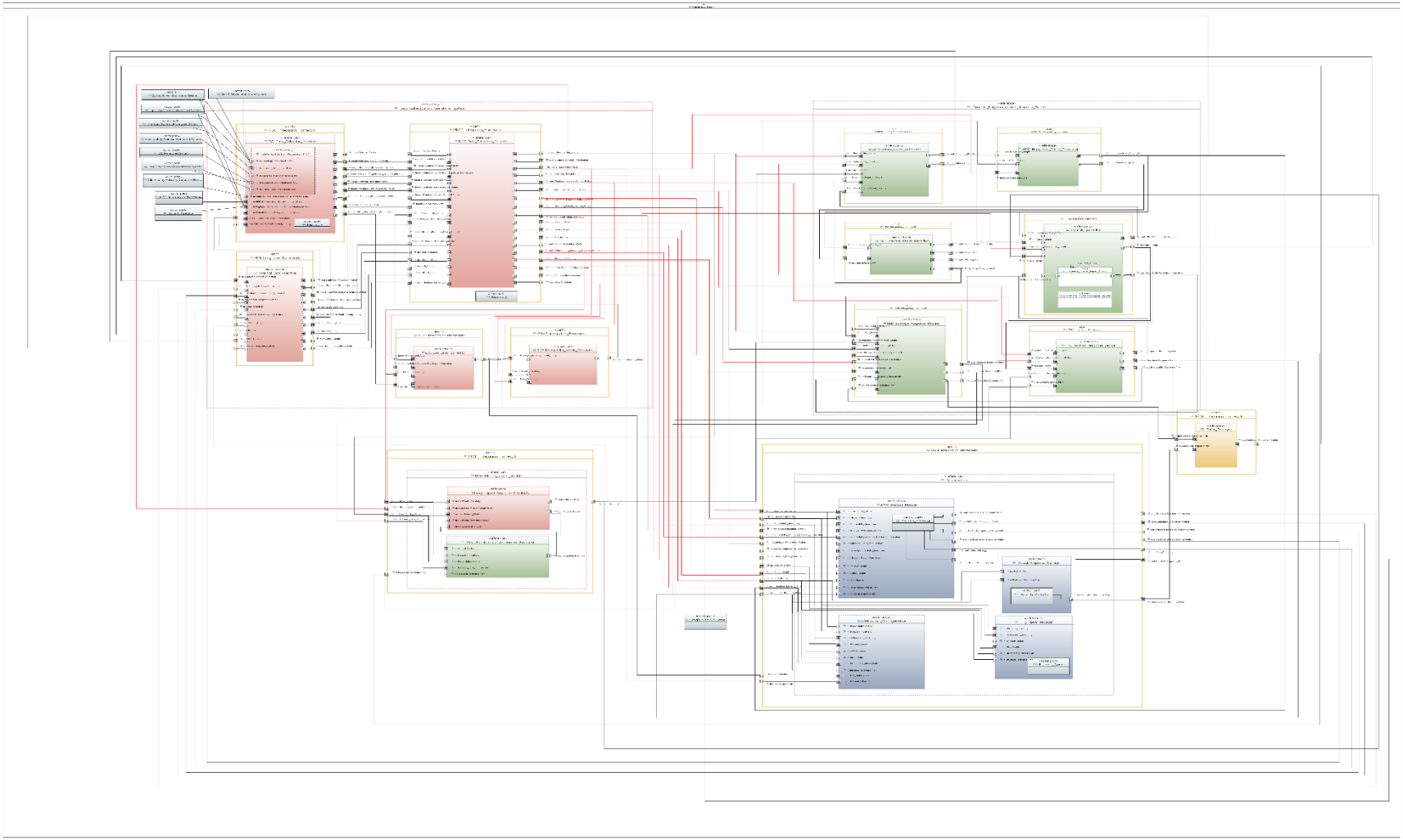
- Presentations to stakeholders
- Formal reviews with stakeholders
- Walkthroughs (with the stakeholders) based on architecturally significant use case scenarios
- Test cases covering ASRs (part of Verification & Validation processes): for each ASR (or a small group of ASRs) test cases verifying the requirement are designed and traced to design and requirement
- Checklists

PANOPTESEC High Level Design



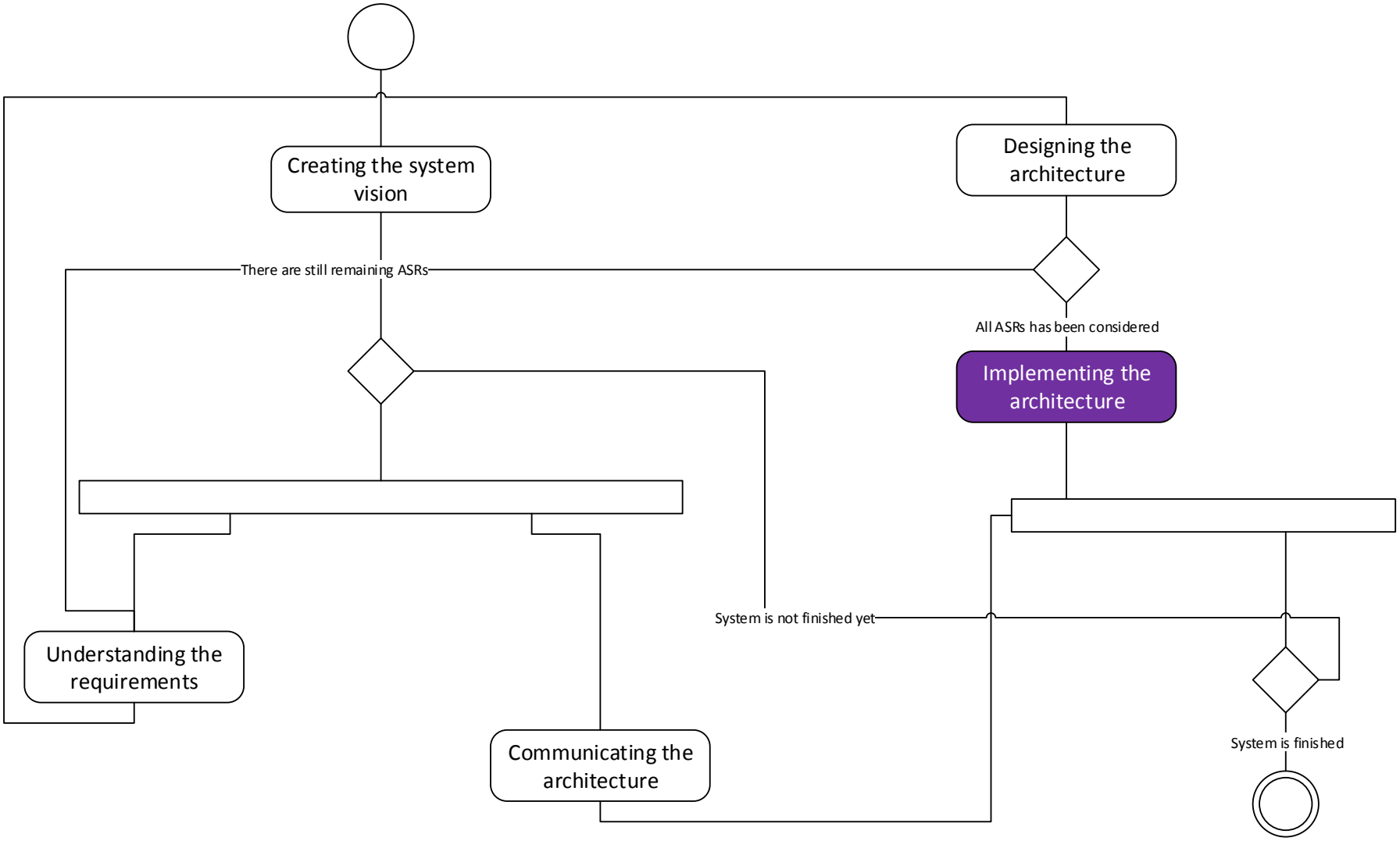
PANOPTESEC

PANOPTESEC High Level Design (detailed)



PANOPTESEC

Implementing the architecture



PANOPTESEC

Implementing the architecture

During the implementation phase, **the architect is still deeply involved** in the project. He works as the orchestrator of the activities related to the development. Among the tasks:

- Definition of the Low Level Design for some (if not all) building blocks: while the architecture usually refers to the High Level Design, it is possible to describe in detail the elements of the System (usually with component/class diagrams in UML). Low Level Design should be directly traced to the High Level Design
- Change management (using traceability): the architect may be involved in any process of change management. If a complete traceability of LLD, HLD and Requirements had been implemented, it is possible to directly understand the impact of a change of a design element or a requirement
- Architecture conformity verification: after the implementation it will be necessary to state if the System still conform to the design (and so, to the requirements)

Implementing the architecture

The architect is a driver for developers. Among his tasks:

- Definition of naming conventions
- Definition of the implementation infrastracture:
  - Programming environments
  - Modeling environments
  - Build environments
  - Configuration management environments
  - Test environments
- Management of interactions between developers and project managers
- Produces most of the requested documentation
- Implementation of the technical basis (frameworks, libraries, e.g.)