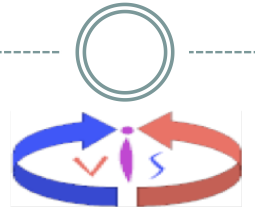
Tool Support for Tradespace Exploration and Analysis



JAKUB J. MOSKAL, MITCH M. KOKAR

Raytheon

PAUL R. WORK, THOMAS E. WOOD

OCTOBER 29, 2014

Background and Motivation

- SBIR Phase I: OSD12-ER2
  - "MOCOP": Functional Allocation Trades Between HW and SW

- Project objectives:
  - Develop a software tool **for allocating system functions to implementations of hardware or software.** The tool shall make comparative (qualitative and quantitative) assessments between allocations of the same function to hardware and software implementations.

Engineered Resilient Systems (ERS)

- OSD: a resilient system *"is trusted and effective out of the box in a wide range of contexts, and easily adapted to many others through reconfiguration and replacement"*
  - Adaptable (and thus robust) designs (based on models)
  - Faster, more efficient design iterations
  - Decisions informed by mission needs
    - More options considered deeply, broader trade space analyses
    - Interaction and iterative design in context among collaborative groups
    - Ability to simulate and experiment in synthetic operational environments

Functional Allocation Use Cases

- Considered at the beginning of the project:
  - Top-down
  - Bottom-up
  - Dynamic (reallocation)
  - Predictions-based

- The reality:
  - Models expressed in SysML
    - No “library”
    - Lack of formal semantics
  - Clean slate design for complex systems rarely happens
  - Composition of existing technology to meet the next generation challenge – a historical problem at DoD

Real-World Challenges (1/2)

- SME-centric process
  - SME's knowledge is not formally captured
  - Trade space is not fully explored
  - Rigid process is desired
- Software reuse not always possible
  - Old components might rely on OS that is no longer available
- Non-functional requirements do matter
  - Example: foreign customers allowed less capable versions only
- Cost of requirements is not well established
  - Requirements become very entangled
  - Sometimes cost is not known until its built

Real-World Challenges (2/2)

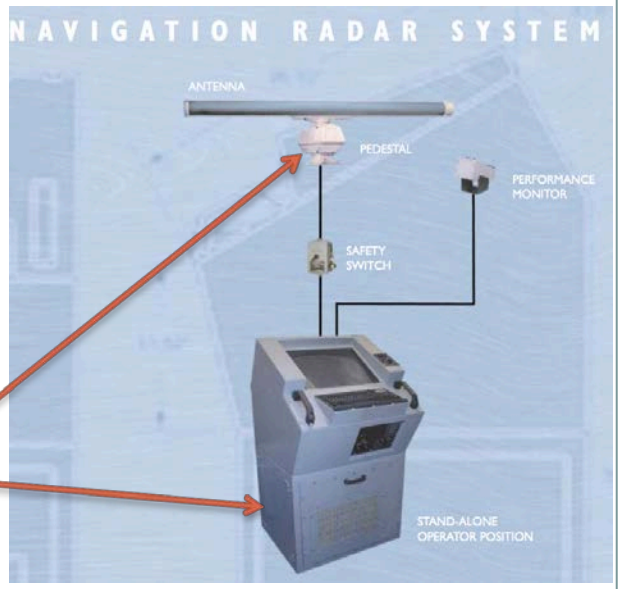
- Appropriate model library is missing
  - Not clear what each model should capture
  - *What would it take for HW component X do something else?*
- The cost function varies
  - Utility function changes from activity to activity
  - Different sources of money
- Contracting arrangements must be considered
  - Single giant model library unlikely
  - Rather: allocation across multiple contractor-specific model libraries
- The impact of reallocation ("ripple effect") must be quantified
  - Scenario 1: New commercial component
    - *Can it improve cost, performance, or functionality of the existing system?*
  - Scenario 2: Change in the customer's affordability of a system
    - *What requirements can we let go?*

Primary Use Case considered in Phase I

- Dynamic reallocation of functionality
  - Driven predominantly by cost reduction
- Use case: Ripple effect assessment
  - Impact on functionality
  - Impact on non-functional aspects:
    - Engineering cost
    - Resilience
    - Performance
    - Reliability

Considered change in the design

- OS-CFAR FPGA HW Unit
  - Implements Constant False Alarm Rate
  - Threshold:
    - Low → detects more targets, but more clutter
    - High → detects less clutter, but fewer targets



- Two designs:
  - Old: FPGA inside the console (radar processor)
  - Considered: FPGA in the pedestal

Ripple Effect

- Multiple consoles can use output of a single sensor (pedestal & antenna)
- FPGA is an expensive piece of HW, there is opportunity to reduce cost
- Compar

SSR1 (FPGA inside the console)		SSR2 (FPGA inside the pedestal)	
Pros:	Cons:	Pros:	Cons:
Each console operator can set a different threshold	Each console must contain its own OS-CFAR FPGA	Only one OS-CFAR FPGA per radar sensor	Every console operator of the same radar sensor must use the same threshold at any given time
Functionality	Cost	Cost	Functionality

CHALLENGE:

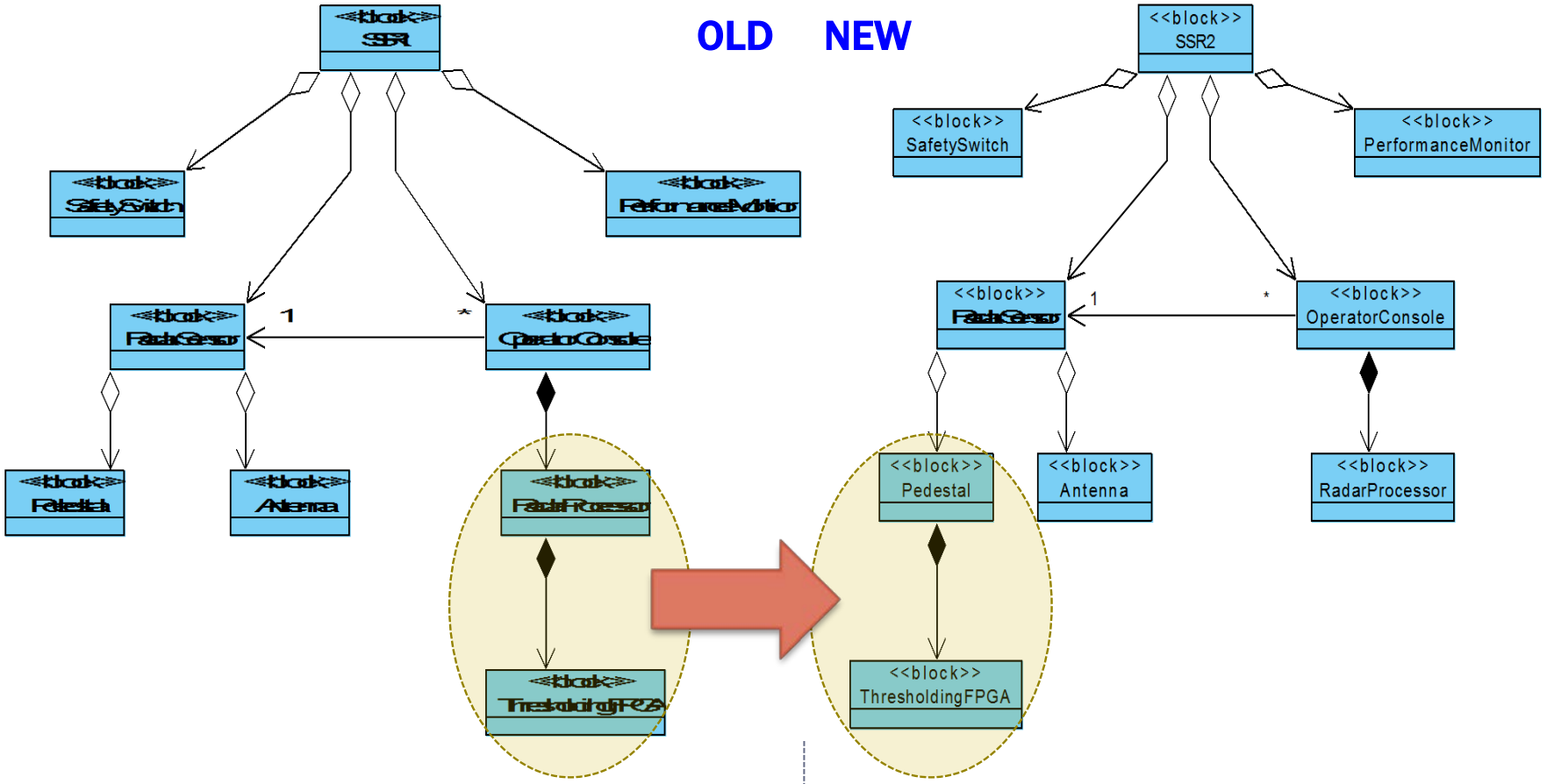
1. Formally represent allocations and assess the impact on different metrics.

2. Provide means to viewing the tradespace.

SysML Requirements and Allocations

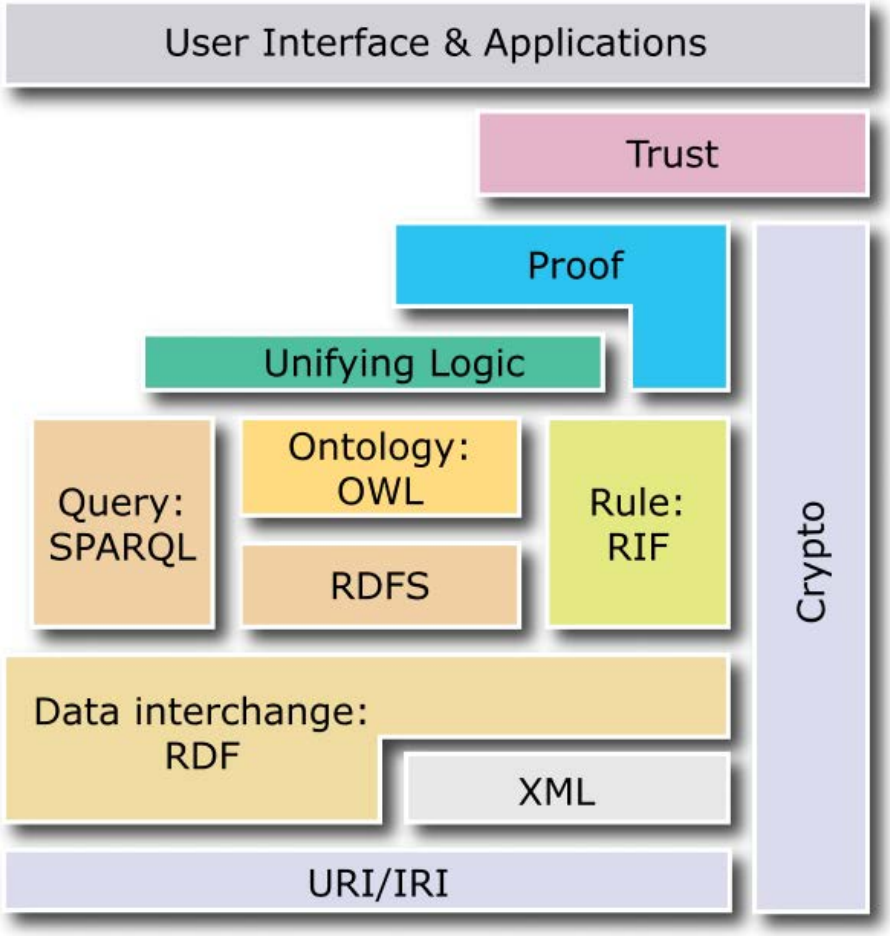
- The problem is to allocate requirements to components (HW or SW)
- Requirements are kept in SysML Requirements Diagrams
  - Functional requirements must be realized
  - Non-functional are represented by objective functions or constraints
- Objective functions use arguments (parameters) captured in SysML Parametric Diagrams
- Constraints expressed with equations
- Allocations defined using meta-associations
  - *<<allocate>>*
  - *<<allocatedFrom>>*
- All necessary input can be collected from existing SysML model

SSR Reallocation Scenario in SysML



Representation Language Stack

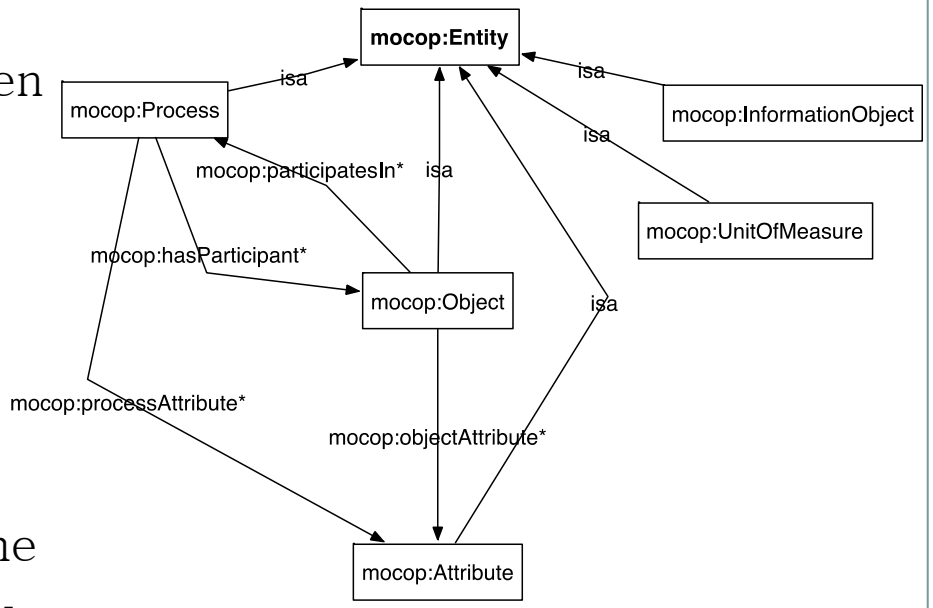
- Knowledge representation
  - Web Ontology Language, OWL (2004) and OWL 2 (2009) – widely adopted in the Semantic Web community
  - Semantics based on Description Logics (DL)
  - Decidable fragment of First-Order predicate Logic (FOL)
- Query Language
  - SPARQL
- Rule Language
  - Rule Interchange Format



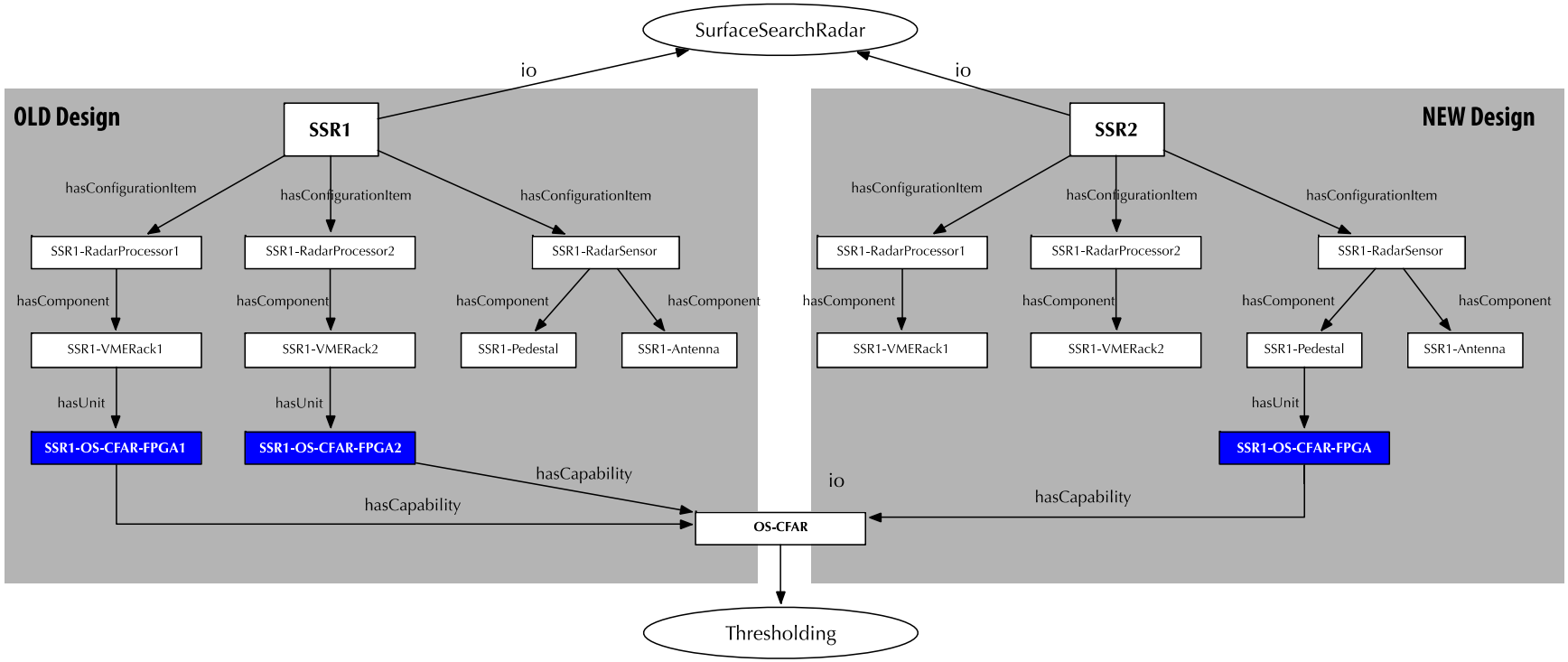
The "Layer Cake" (Tim Berners-Lee)

MOCOP Upper-level ontology

- Based on DOLCE
  - Object (endurant)
    - Wholly represented at any given snapshot of time
    - Here: systems, configuration items, components, units
  - Process (perdurant)
    - Can be represented only partially at any snapshot of time
    - Here: capabilities, functionality, requirements



Old and New Design in OWL



Assessment of the *Ripple Effect* – Functionality

- System functionality measured in terms of requirements it meets
  - Hard requirements – must be met, otherwise the allocation is invalid
    - E.g. radar system must have an antenna
  - Soft requirements – might be let go of, depending on the objectives, e.g. cost reduction
    - E.g. radar system must have two operator consoles
- There are no "*user features*"
  - The system must meet all requirements, at minimum cost

CFAR Soft Requirement

- Intent:
  - *The system shall allow for independent threshold selection for each operator console connected to the same radar sensor.*
- Encoded as OWL Restriction class:
  - SSRSoftRequirement1 ⇔
    SurfaceSearchRadar **and** hasConfigurationItem **some**
    (RadarProcessor **and** hasCapability **some** Thresholding)
- Allocation meets the requirement if it is inferred as its instance:
  - SSR1-1 **rdf:type** SSRSoftRequirement1

Functionality Assessment Rules

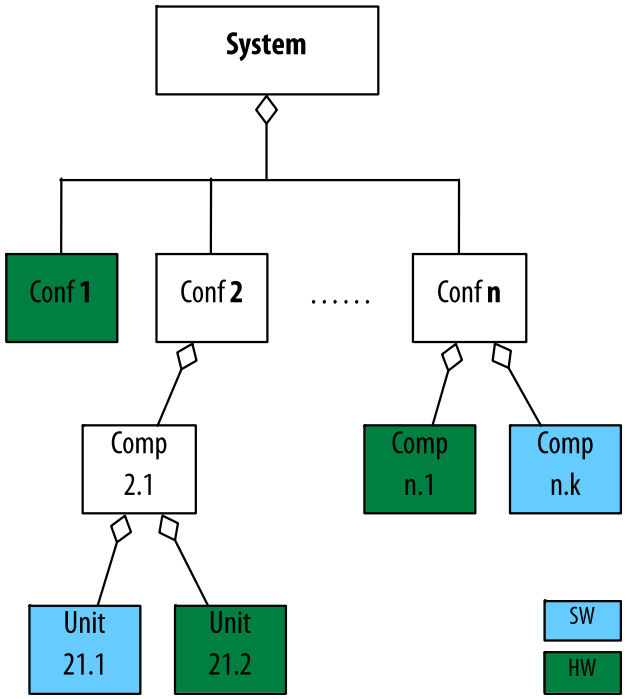
- Identify invalid systems
  - System that does not meet all of the hard requirements
- Identify hard requirements met
  - Systems that are instances of mocop:HardRequirement
- Identify soft requirements met
  - Systems that are instances of mocop:SoftRequirement

- Establish requirements coverage for each system
  - Compare requirements met vs. all requirements

- All rules and procedural attachments are generic
  - SSR-specific concepts are not included

Assessment of the *Ripple Effect* - Cost

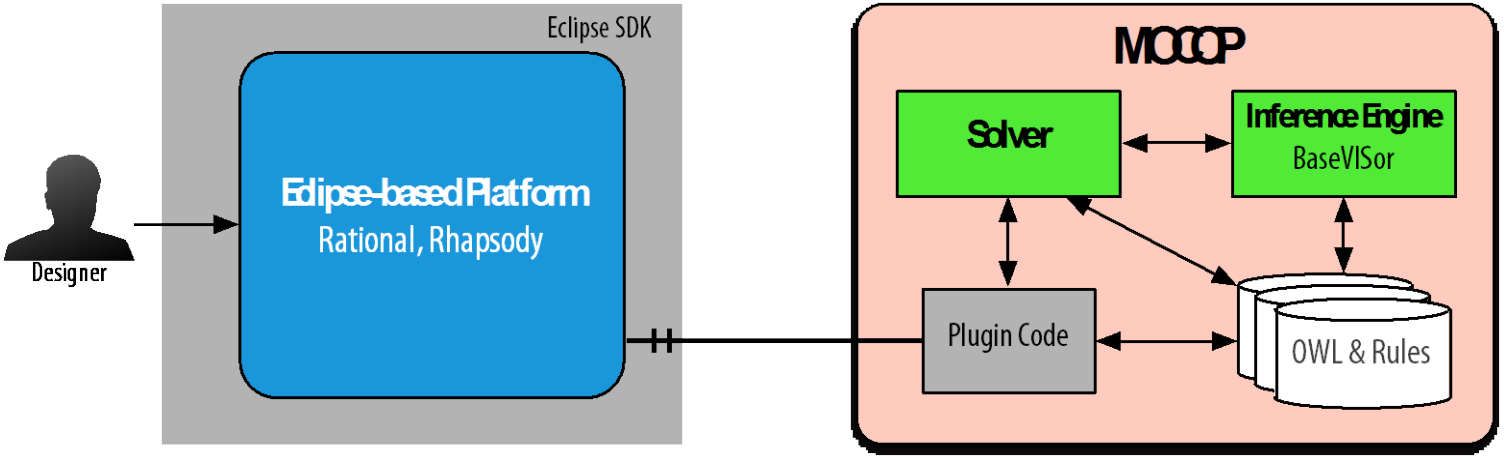
- Cost is not a single dimension:
  - Cost of production
  - Material cost
  - Sales price (third-party)
  - Operating cost
  - Maintenance cost
  - Sustainment cost
- One type of cost considered in Phase I
  - Measured in US dollars
- Depth-first search of the decomposition tree
  - Include cost of integration (middleware, enclosure)
  - OWL not suitable for this task
    - Rules are needed to "walk" the tree
    - Procedural attachments to do algebraic operations



Cost Assessment Rules

- Identify part-whole relationships:
  - System (1-*) ConfigurationItem
  - ConfigurationItem (1-*) Component
  - Component (1-*) Unit
- Identify the cost of each system part
- Sum the cost of system parts
  - For each system in the knowledgebase
- All rules and procedural attachments are generic
  - SSR-specific concepts are not included

MOCOP Prototype Architecture



- GUI
  - Designer interacts directly with well-known software:
    - IBM Rational, Rhapsody
- MOCOP:
  - Implemented as an Eclipse plugin
- Solver
  - Implements optimization algorithm
- Inference Engine
  - Matching
  - Decomposition
  - Ripple Effect assessment
- Ontologies & Policies
  - Formally represented library of models and functions

Implementation Details

- Inference engine: BaseVISor
  - OWL 2 RL
  - Custom semantic rules
  - Procedural attachments
  - Embeddable, JVM environment
- Ontologies developed using Protégé
  - MOCOP ontology
  - SSR ontology extends MOCOP, domain-specific
- Rules expressed in BVR
  - In the future, they could be expressed in RIF/SBVR
- Controller written in Java
  - Ripple effect is assessed and saved as an Excel spreadsheet

ConOps (1/2)
Building model library stage

1. **System engineer** responsible for a specific system element (unit, component, etc.) uploads relevant SysML diagrams

2. The **MOCOP** plugin converts the diagrams into OWL representation, displays a GUI with prepopulated values from the diagrams

3. **System engineer** provides additional input that was not possible to capture in the SysML diagrams

4. The MOCOP plugin stores the values entered in the GUI as OWL

Designer is not aware that OWL-based technology is used

ConOps (2/2)
Design stage

1. **Designer** provides necessary input:
   - Requirements for the system are specified using SysML Requirements Diagram
   - Objective functions and constraints are captured in SysML Parametric Diagrams

1. The **MOCOP plugin** displays the trade space
   - Each point is associated with a specific solution
   - Each solution represented in SysML diagrams: block, parametric, allocation, etc.

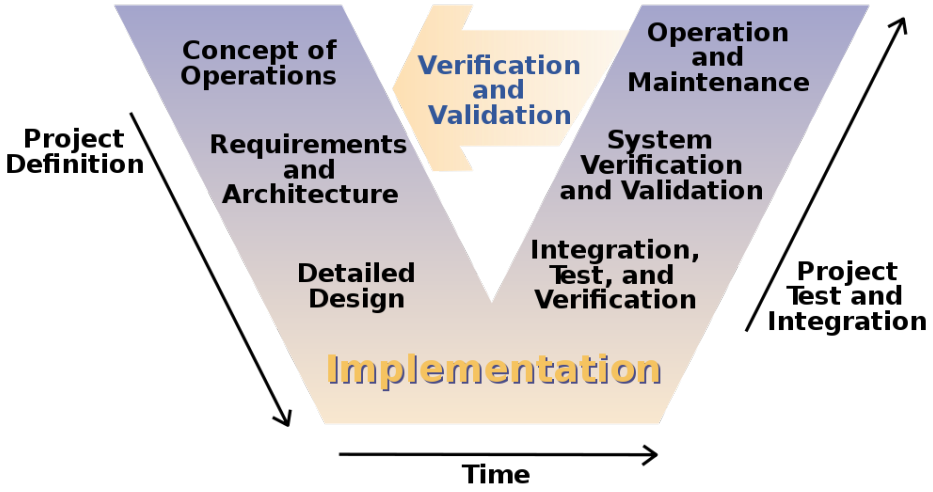
1. **Designer** might reject some solutions or change constraints and rerun the trade space analysis

2. The iterative process continues until the designer finds the best solution

Designer is not aware that OWL-based technology is used

Meeting the ERS objectives

- Our approach supports ERS:
  - Trade space analysis at early stage
  - Discover unintuitive solutions
  - Avoid integration problems



Thank you!

- Interested parties are welcome to contact VIStology:
- Jakub Moskal: jmoskal@vistology.com