Open Analytic Runtime Models for CPS

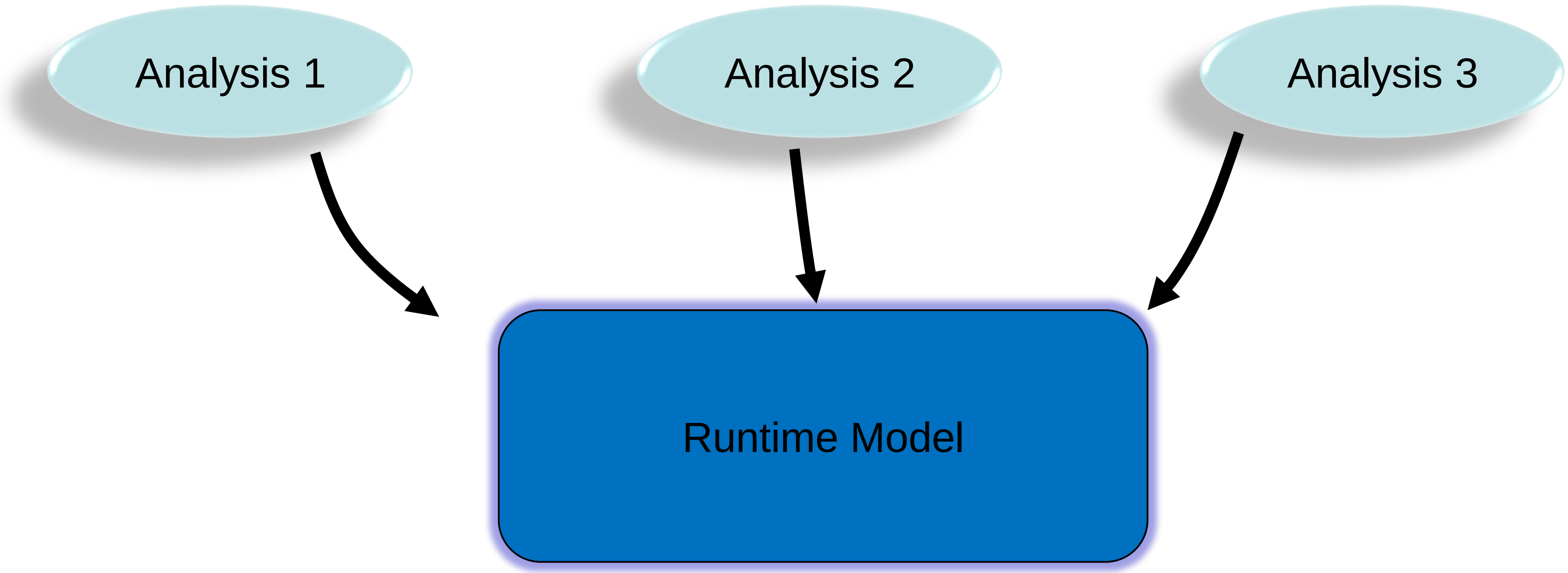
Min-Young Nam
Dionisio de Niz
Lutz Wrage
Lui Sha

Presented at the Workshop on Architectures for CPS
11 April 2011 CPS Week

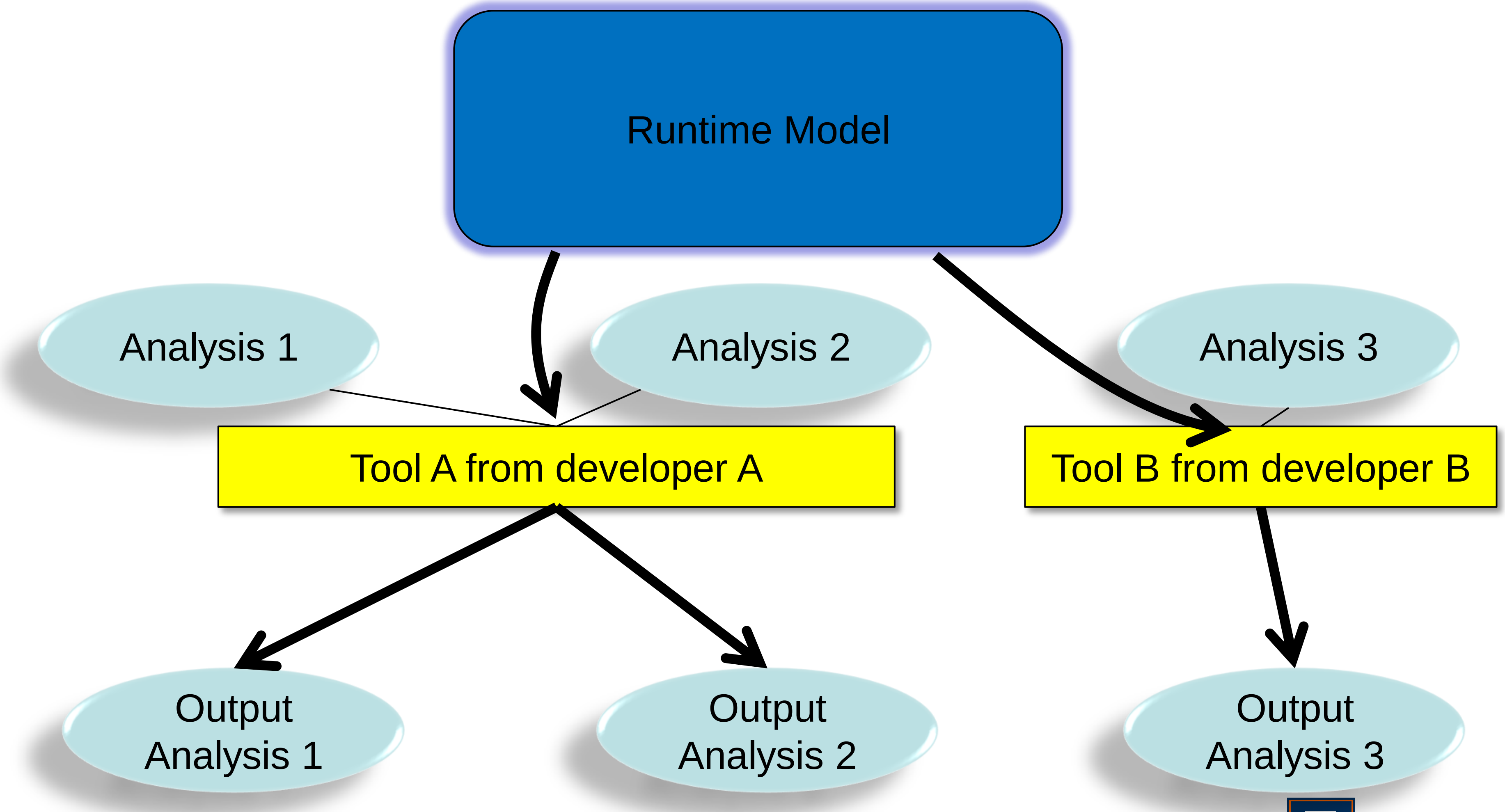
Motivation

- Cyber-Physical Systems (CPS) are *complex* software-reliant systems that *interact* with physical processes
- Analytic algorithms are used to verify safety critical properties at a higher level of abstraction and synthesize the low level behavior of design decisions
- Model-Based Engineering (MBE) is a promising solution for early analysis
- For large complex systems with multiple analysis support, the general method of implementing analysis tools is error prone, costly and limits the benefits of MBE.

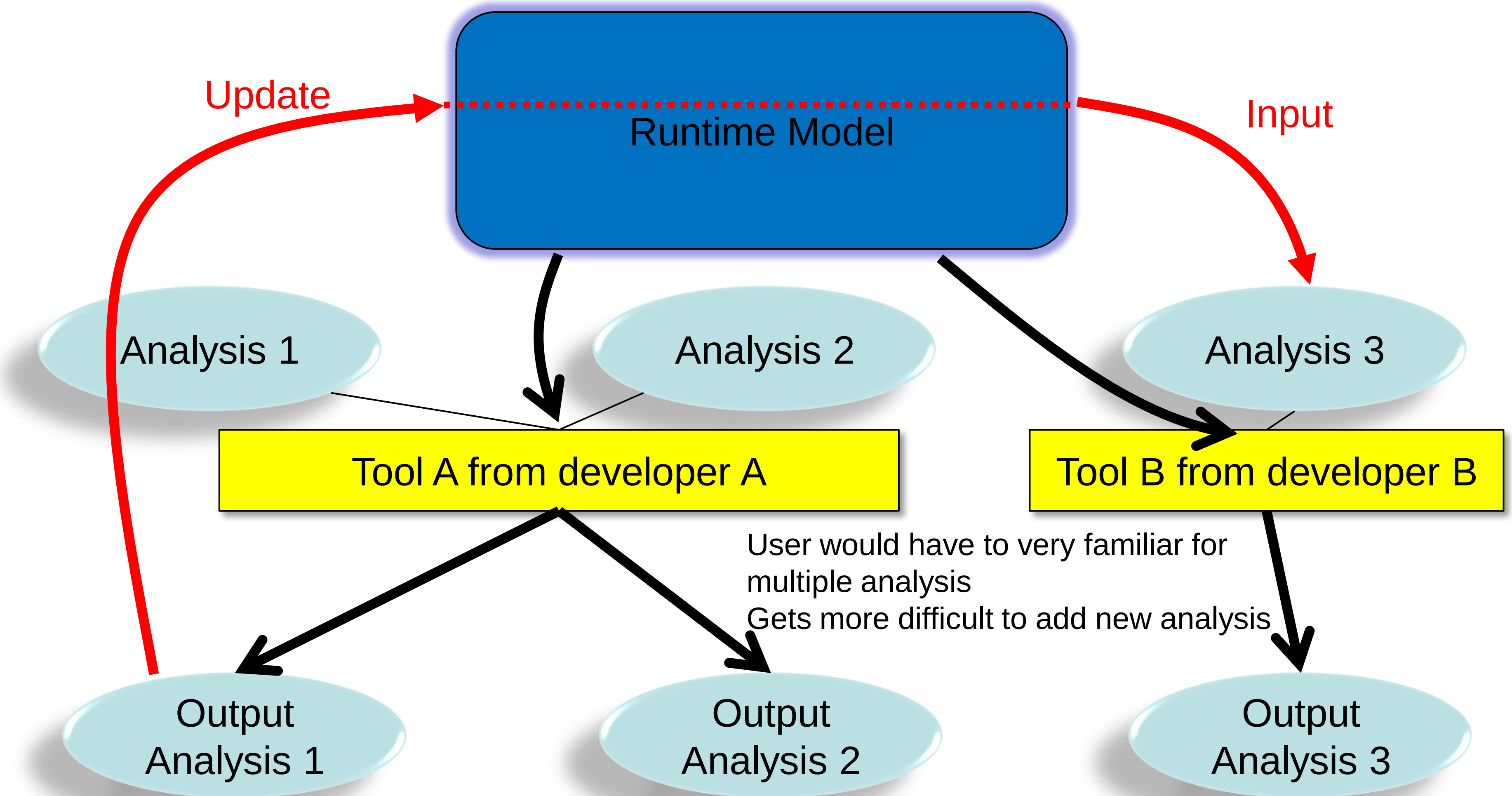
Motivation Scenario



Motivation Scenario

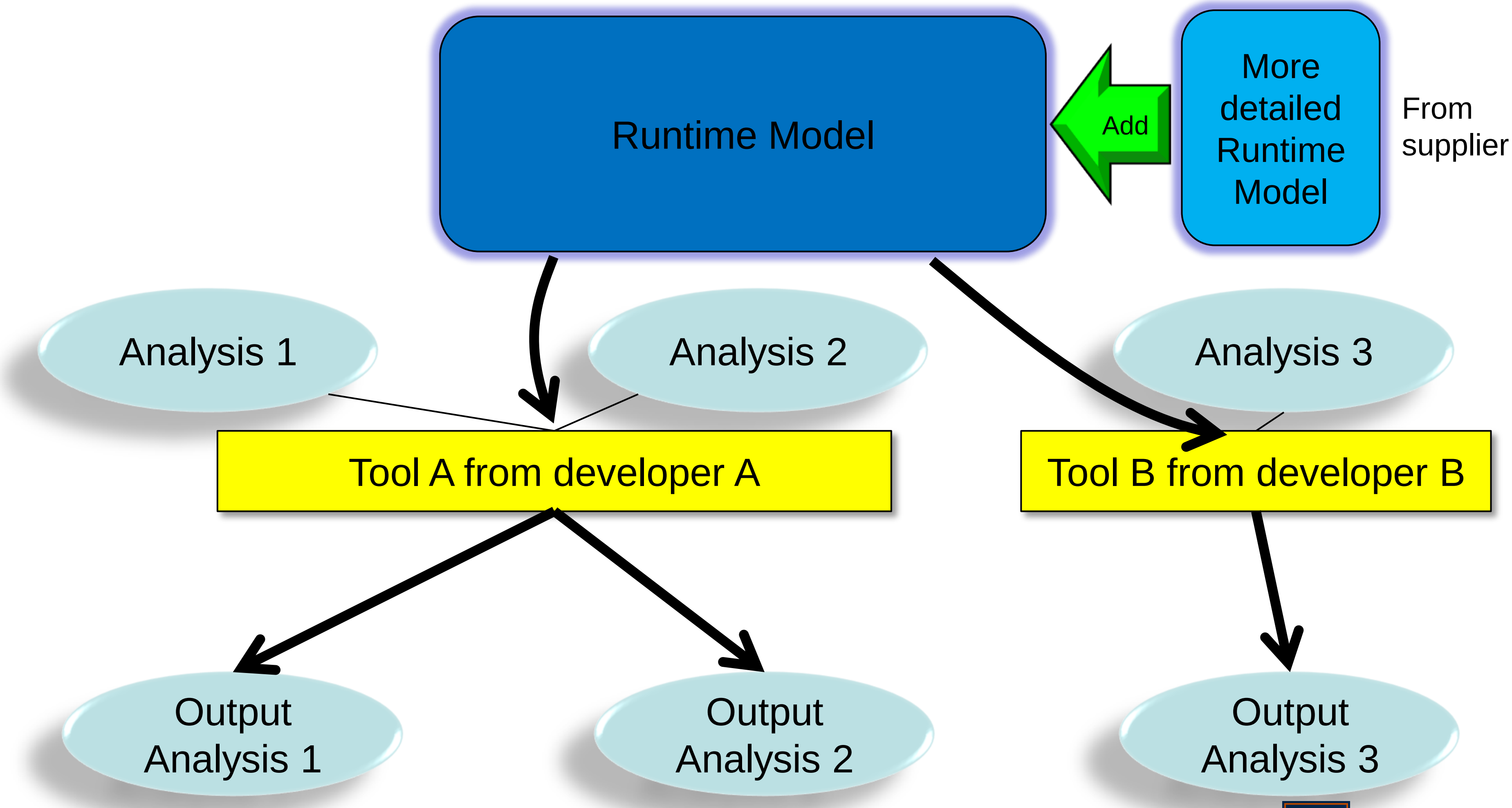


Motivation Scenario 1

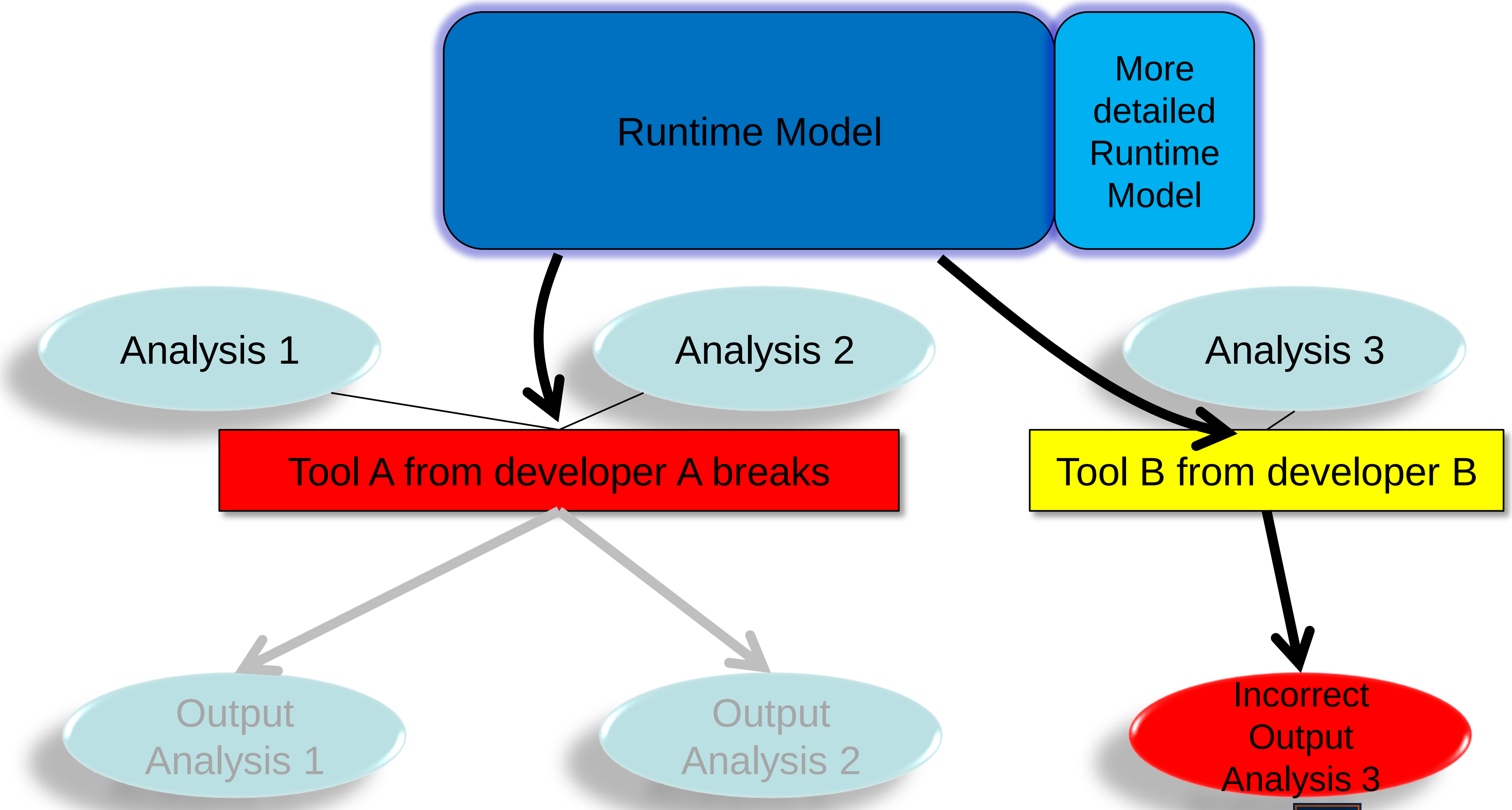


User would have to very familiar for multiple analysis
Gets more difficult to add new analysis

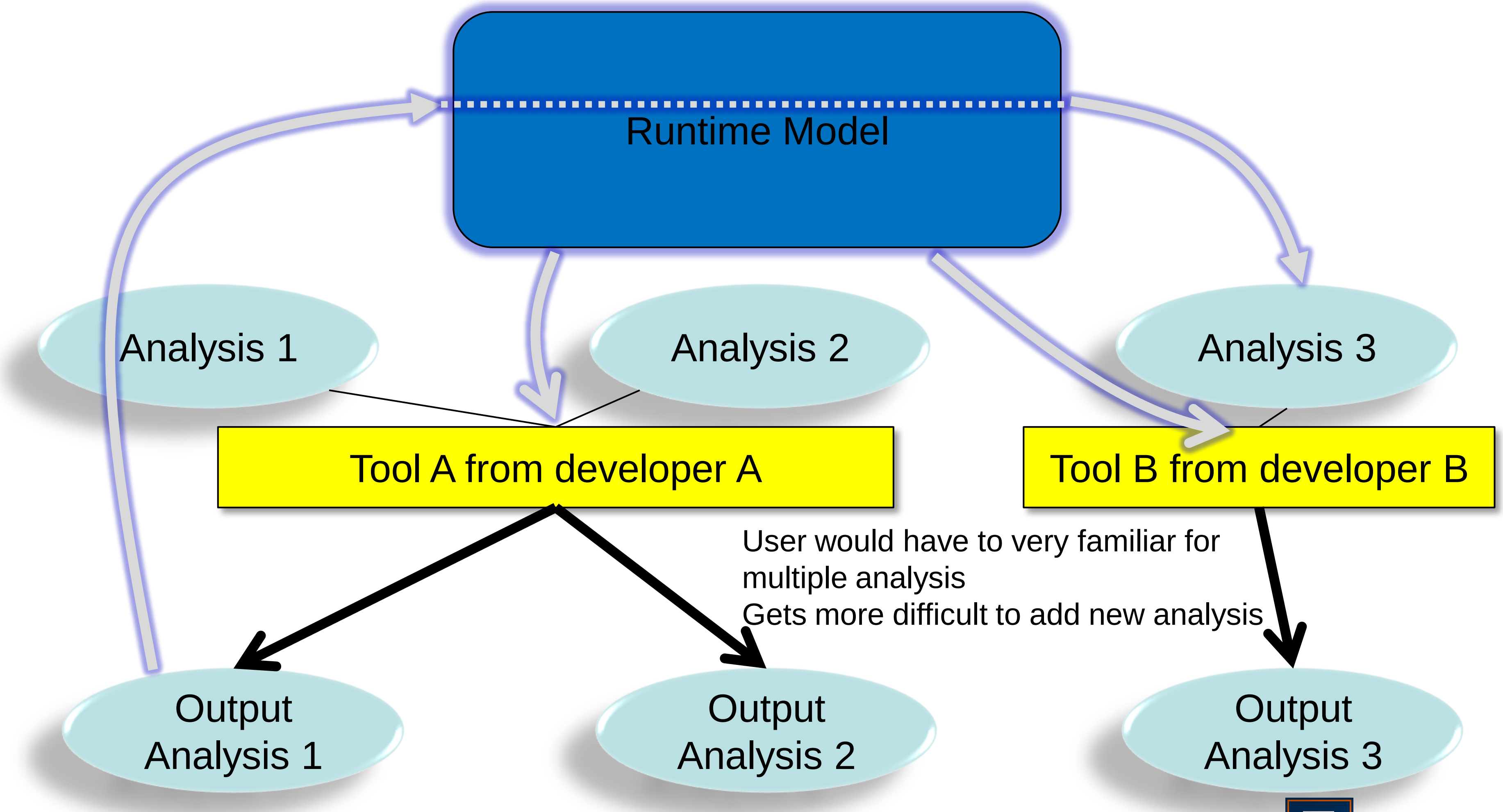
Motivation Scenario 2



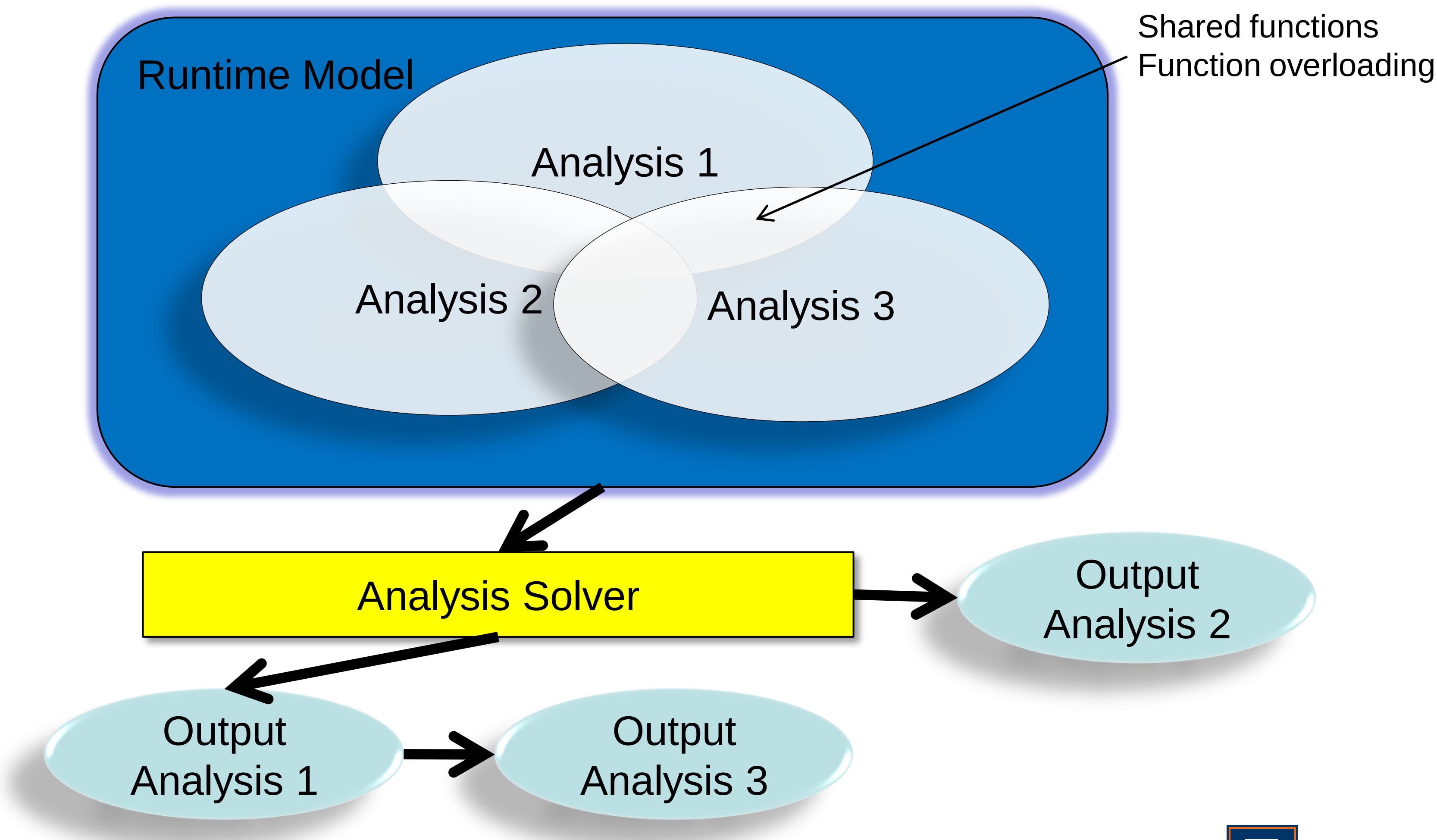
Motivation Scenario 2



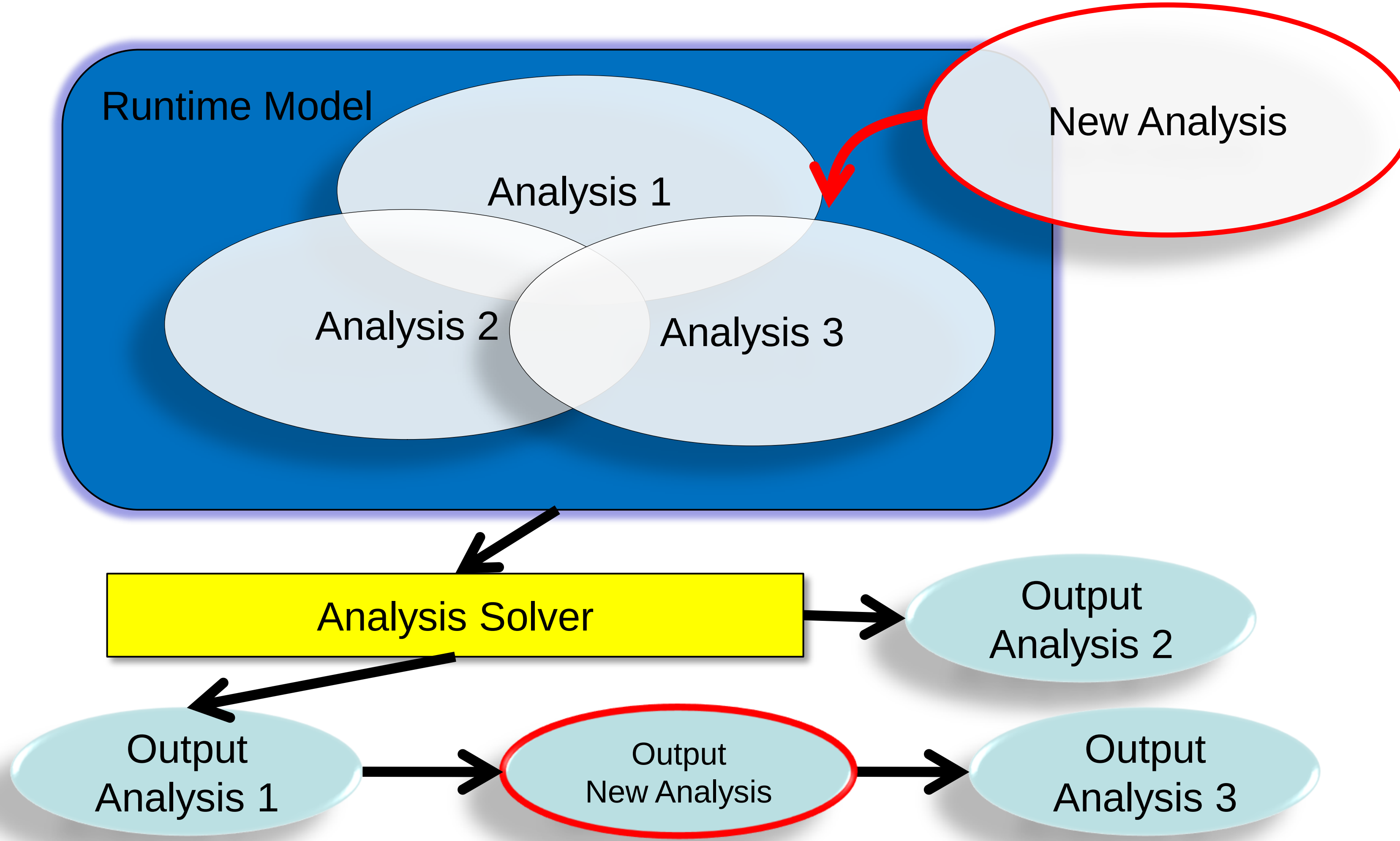
Problem: Users are in the Dark



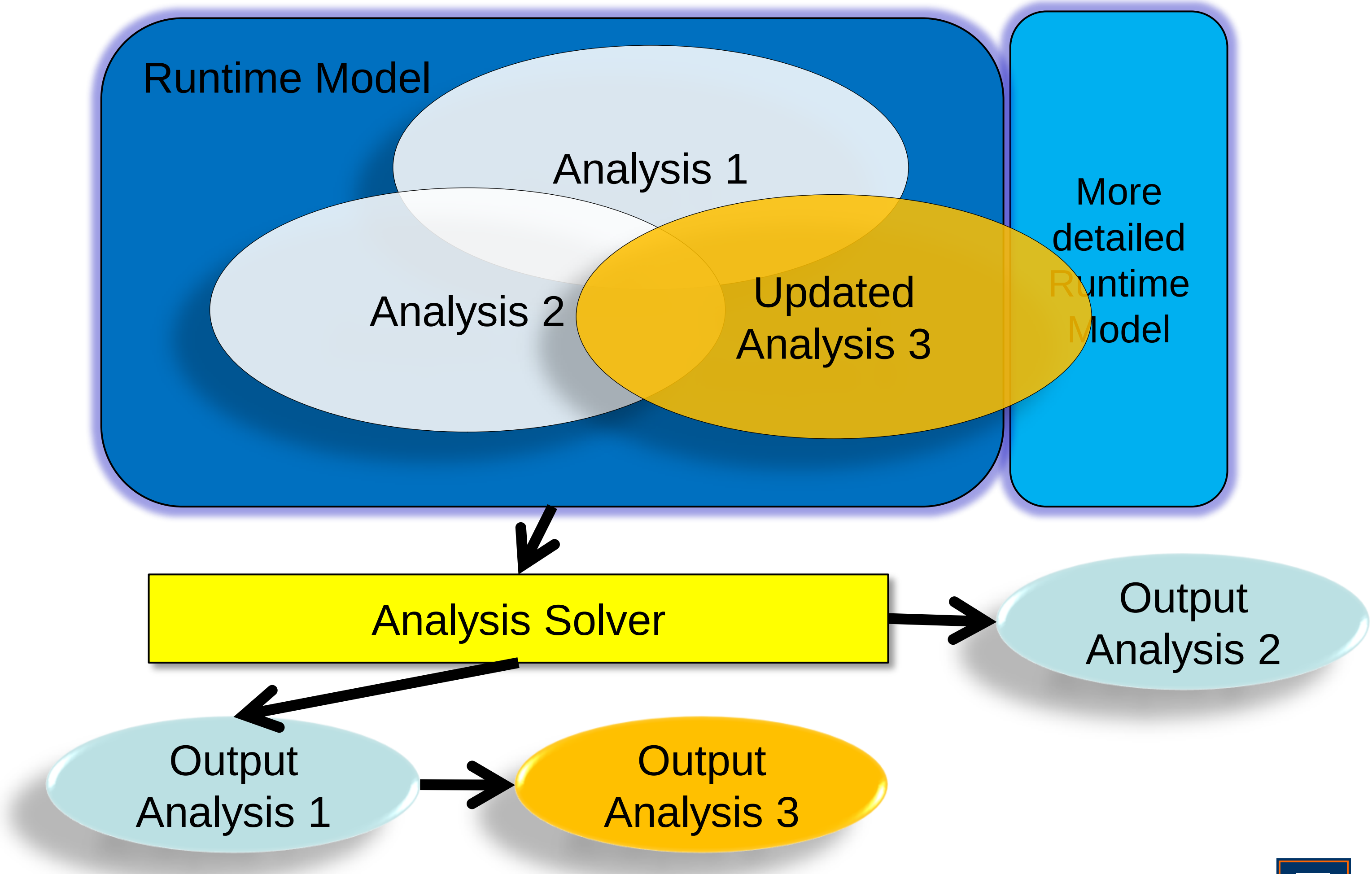
Our Solution: OAR models



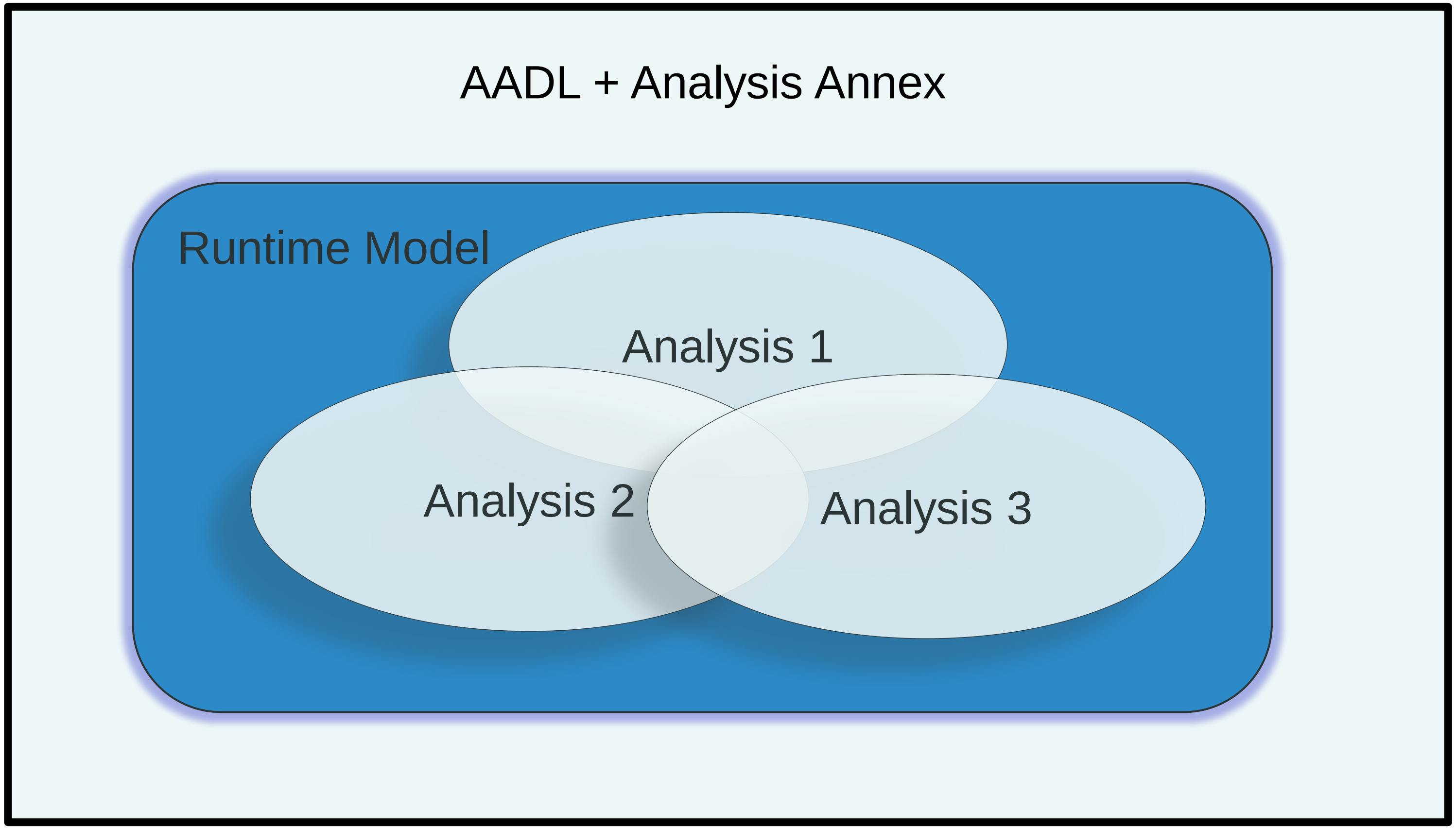
Our Solution: OAR models



Our Solution: OAR models



Our Solution: OAR models



Contents

- Problem Statement
- Features of OAR models
- OAR models in AADL → Analysis Annex
- Type system for Model Reuse
- Example model and analysis
- Performing Analyses
- Conclusion

Problem Statement

- Hidden Semantics of Analysis Algorithms
  - Hides complex behavior that the algorithm explores implicitly
  - Interpretation of the model
    - What units are used for execution time
  - Assumptions of the behavior of other parts of the model
- These difficulties makes the model-based engineering benefits limited for models

Features of OAR models

- Models embed the executable analytic algorithms used to verify properties of the system.
- Have complete declaration of the parts of the model they read and write.
- Functional decomposition of the algorithm that follows the decomposition in the model.
- Type system for safe analysis reuse.

OAR Models in AADL

- SAE AADL (Architecture Analysis & Design Language)
  - Notation for specification of runtime architecture of real-time, embedded, fault-tolerant, secure, safety-critical, software-intensive systems - designed for Model Based Engineering
  - Fields of application: Avionics, Aerospace, Automotive, Autonomous systems, Medical devices – current focus
  - Based on 15 years of research & industry input
  - International Standard approved & published Nov 04, V2 Jan 09
  - Industry driven standard
  - www.aadl.info

OAR Models in AADL (cont'd)

- *Analysis Annex* for AADL
  - Structure
    - » *properties*
      - – AADL properties that queries and functions access
    - » *queries*
      - – Model Query Language (MQL)
    - » *functions*
      - – Represent the analysis algorithm with a set of functions
    - » *updates*
      - – Identifies the functions that modify the model along with the part of the model that these functions modify

properties and *updates*

- *properties*
  - AADL properties that queries and functions access
  - List property names separated by commas
- *updates*
  - <element type>.<property access> <– IDENTIFIER()
  - Ex) processor:p.ACE::CurrentFrequency <- minFrequencyPairs();

queries: Model Query Language

- Used to traverse the model
  - IDENTIFIER <– <type specification>:IDENTIFIER [in IDENTIFIER] | <filter condition>
    - Ex 1) higherPriorityThreads <- **thread**:x | x.Actual_Processor_Binding =t**his**.Actual_Processor_Binding and x.Deadline(ms) <= **this**.Deadline(ms);
    - Ex 2) AllSubProcessors <- **processor:**p in **this**;
    - Ex 3) BPAllProcesses <- process:p in this;
    - BPAllThreads <- thread:t in BPAllProcesses;
  - IDENTIFIER <–(<type specification>:<var name>[in IDENTIFIER] | <filter condition>).<property access>

functions: Analysis Functions

- Contains the set of functions that comprise the analysis algorithm
  - <return value type> <function_name> (<list of argument definition>)
  - Calling local function
    - this.<function_name>(<list of arguments>)
  - Accessing local AADL property
    - this.<property access>
  - Calling functions of query result
    - <query_result_name>.<function_name>(<list of arguments>)
  - Accessing AADL property of query result
    - <query_result_name>.<property access>

functions: Example

```
thread PowerEfficientThread
  annex analysis {**
  queries
   higherPrioThrds <- thread:x |
    x.Actual_Processor_Binding =
     this.Actual_Processor_Binding and
     x.Deadline(ms) <= this.Deadline(ms);
  functions
   double EnergyMinFreq(int maxClockFreq){
    double omega;
    d = this.Deadline(ms);
    ...
    omega = this.
     Compute_Execution_Time[max](ms) /
      maxClockFreq;
    ...
    subOmega = higherPrioThrds.
     getSubOmega(omega, maxClockFreq);
    subIdleDuration = higherPrioThrds.
     getSubIdleDuration(omega, d);
    ... }
   double getSubOmega(double omega,
    int maxClockFreq){
    this.Compute_Execution_Time[max](ms) /
     maxClockFreq * (Floor[(omega /
     this.Period(ms))] +1) }
   double getSubIdleDuration(double omega,
    double deadline){ ... }
  **};
```

Query result name

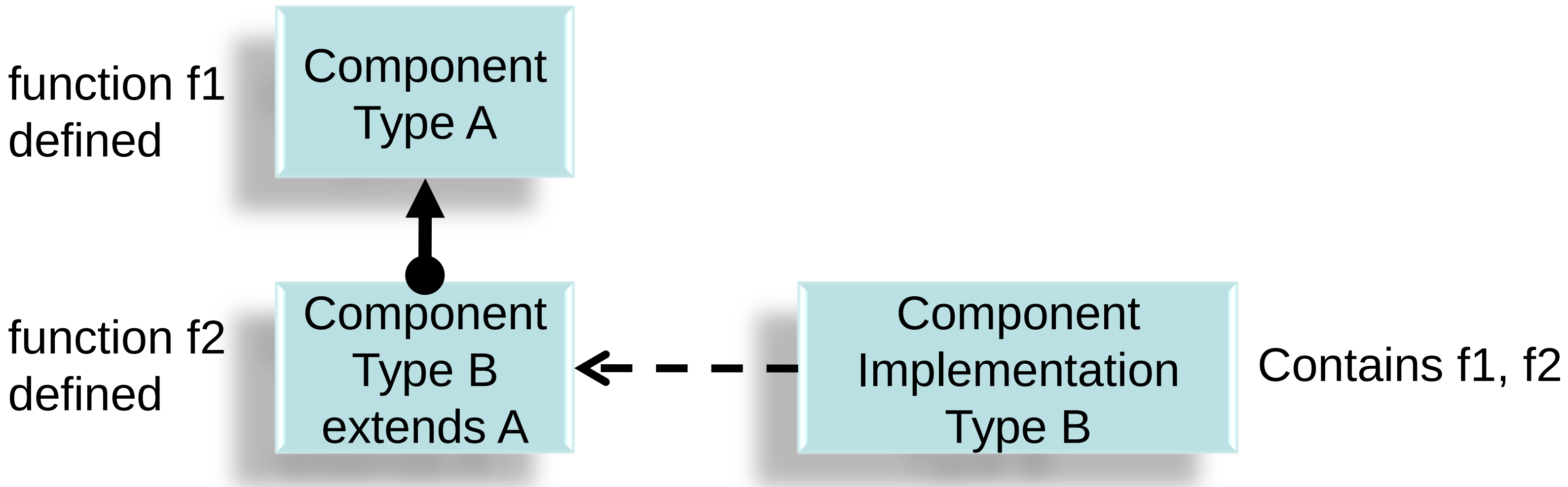
Using local AADL property

Calling functions of query results

Decomposing of functions

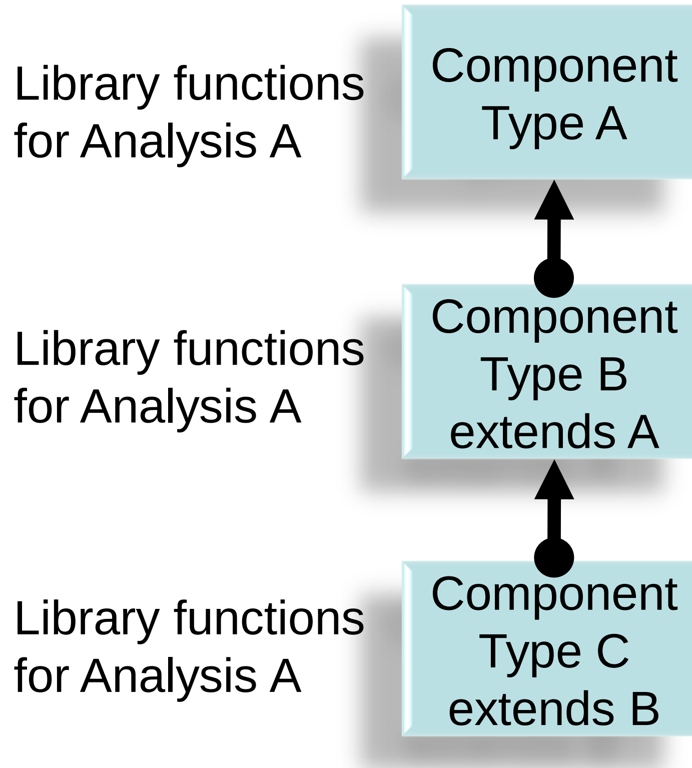
Type System for Model Reuse

- Similar to Object-Oriented programming
  - Code -> Analysis algorithm
  - Data structure -> Model
- Analysis annex Type System
  - For data, already implemented in AADL
  - Inheritance
    - Analysis annex generally follow the inheritance of AADL components
    - Queries are not inherited for avoiding confusion about query conditions



Type System for Model Reuse

- Analysis Libraries can be defined
- Inheriting Multiple Annexes
  - For a system that supports more than two analysis of different types each type would have to extend one another for the system to inherit all analysis
  - This is due to the single-parent inheritance of AADL



Contents

- Problem Statement
- Features of OAR models
- OAR models in AADL → Analysis Annex
- Type system for Model Reuse
- Example model and analysis
- Performing Analyses
- Conclusion

Example Model

- Seven applications on a battle ship
  - RadarTracking: interfaces with the radar device and creates tracks of the objects in the sky
  - UAVTracking: consolidates the tracking information received from UAVs (Unmanned Aerial Vehicles)
  - EngagementPlanning: processes the tracking information received from the RadarTracking and UAVTracking threads and develops engagement strategies
  - AssetControl: receives the engagement strategies from the EngagementPlanning threads and coordinates the assets in an engagement
  - RequestPressRelease: receives press release requests from news agencies
  - PressReleaseClearance: sanitizes the engagement information received from the engagement planning module and generates responses to the press release request forwarded by the RequestPressRelease thread
  - PressReleaseDissemination: transmits the sanitized information to the news agencies
- Four processors available to use

Example Analysis

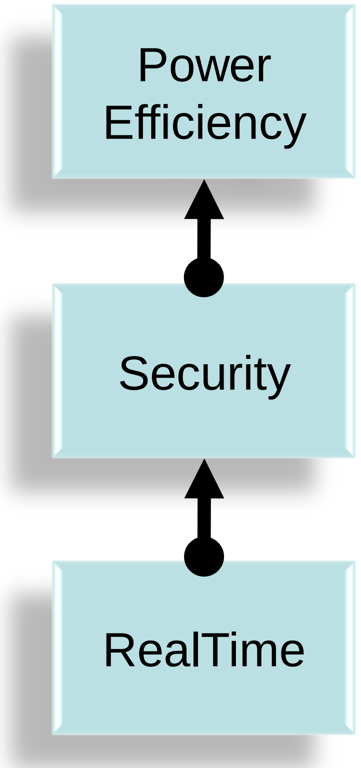
- Confidentiality assurance algorithm (Security package)
  - Using the security level of applications, determines which threads should not executed by the same processor for security

- Bin packing algorithm (RealTime package)
  - Assigns tasks to processors

- Frequency scaling algorithm (PowerEfficiency package)
  - Reduces the frequency of processors while ensuring deadlines of real-time tasks are met

Example Analysis

- Confidentiality assurance algorithm (Security package)
- Bin packing algorithm (RealTime package)
- Frequency scaling algorithm (PowerEfficiency package)



Example Analysis

- Confidentiality assurance algorithm (Security package)
- Bin packing algorithm (RealTime package)
- Frequency scaling algorithm (PowerEfficiency package)

Listing 2 Example Model for Confidentiality Assurance

```
package Security
  public
    thread SecureThread extends
        PowerEfficiency::PowerEfficientThread
      annex analysis {**
      functions
       String getSecurityClass() {
            this.Security_Attributes::Class }
          **};
    end SecureThread;

    system SecureSystem extends
        PowerEfficiency::PowerEfficientSystem
      annex analysis {**
        functions
         int Security(){
          ...
          classes = AllThreads.
           getSecurityClass();
          ... }
        updates
         thread:t.Not_Collocated <- Security();
      **};
    end SecureSystem
end Security;
```

Example Analysis

- Confidentiality assurance algorithm (Security package)
- Bin packing algorithm (RealTime package)
- Frequency scaling algorithm (PowerEfficiency package)

Listing 3 Example Model for Bin packing

```
package RealTime
  public
    thread RealTimeThread extends
        Security::SecureThread
      annex analysis {**
      functions
       double getUtilization(){
        ...
        p=this.Period(ms);
        c=this.Compute_Execution_Time[max](ms);
        ... }
       int getNotCollocated(){
        this.Not_Collocated } **};
    end RealTimethread;

    system RealTimeSystem extends
         Security::SecureSystem
      annex analysis {**
      queries
       BPAllProcesses <- process:p in this;
       BPAllThreads <- thread:t in
        BPAllProcesses;
       AllProcessors <- processor:pr in this;
      functions
       ...
       BinPack(){ ... }
      updates
       thread:t.Actual_Processor_Binding <-
        BinPack();
      **};
    end RealTimeSystem;
end RealTime;
```

Example Analysis

- Confidentiality assurance algorithm (Security package)
- Bin packing algorithm (RealTime package)
- Frequency scaling algorithm (PowerEfficiency package)

```
system PowerEfficientSystem
  annex analysis {**
  queries
   AllSubProcessors <- processor:p in this;
  functions
   int minFrequencyPairs(){
    ...
    minfreq = AllSubProcessors.
     MaxEnergyMinFreq();
    allprocs = AllSubProcessors;
    ... }
  updates
   processor:p.ACE::CurrentFrequency <-
    minFrequencyPairs();   **};
  end PowerEfficientSystem;
end PowerEfficiency;
```

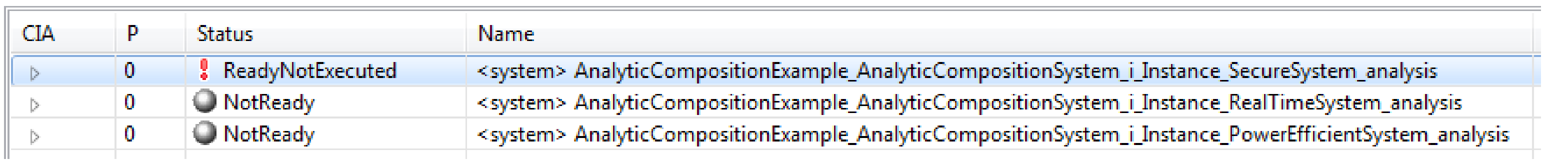
Integrated System

Listing 4 Example Integration System

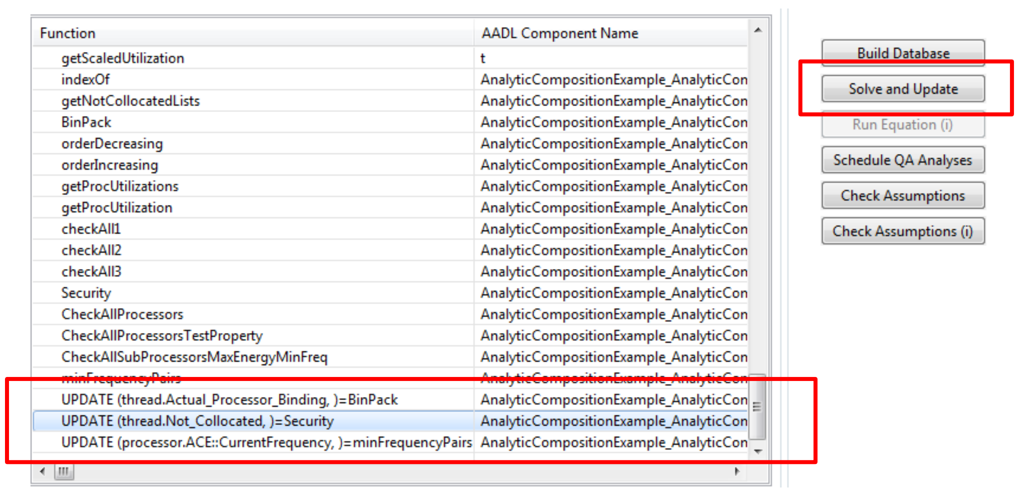
```
thread RadarTrackingThread extends
  RealTime::RealTimeThread
  properties
   Security_Attributes::Class => top_secret;
   Period => 100 ms;
   Deadline => 100 ms;
   Compute_Execution_time => 25 ms .. 30 ms;
end RadarTrackingThread;

process implementation RadarTrackingProcess.i
  subcomponents
    t: thread RadarTrackingThread;
end RadarTrackingProcess.i;

system AnalyticCompSystem
  extends RealTime::RealTimeSystem
end AnalyticCompSystem;

system implementation AnalyticCompSystem.i
  subcomponents
    p1: processor
        RealTime::RealTimeProcessor;
    ...
    p4: processor
        RealTime::RealTimeProcessor;
    radarTracking: process
        RadarTrackingProcess.i;
    engagementPlanning: process
     EngagementPlanningProcess.i;
    ...
  connections
    c1: event data port radarTracking.tracks ->
     engagementPlanning.tracks;
    ...
end AnalyticCompSystem.i;
```

Performing Analyses

- Scheduling Analysis



- Executing Updates through solver



Analysis Result

Analysis	Updated components	Property type	Property value
Confidentiality Assurance	assetControl (secret)	Not_Collocated	engage, clearance, dissemination, radar, req_Release, uavTracking
	engagementPlanning (top_secret)	Not_Collocated	asset, dissemination, req_Release
	pressReleaseClearance (top_secret)	Not_Collocated	asset, dissemination, req_Release
	pressReleaseDissemination (unclassified)	Not_Collocated	asset, engage, clearance, radar, uavTracking
	radarTracking (top_secret)	Not_Collocated	asset, dissemination, req_Release
	requestForPressRelease (unclassified)	Not_Collocated	asset, engage, clearance, radar, uavTracking
	uavTracking (top_secret)	Not_Collocated	asset, dissemination, req_Release
Bin Packing	assetControl	Actual_Processor_Binding	processor p1
	engagementPlanning	Actual_Processor_Binding	processor p2
	pressReleaseClearance	Actual_Processor_Binding	processor p2
	pressReleaseDissemination	Actual_Processor_Binding	processor p3
	radarTracking	Actual_Processor_Binding	processor p2
	requestForPressRelease	Actual_Processor_Binding	processor p3
	uavTracking	Actual_Processor_Binding	processor p2
Frequency Scaling	processor p1	ACE::CurrentFrequency	0.3
	processor p2	ACE::CurrentFrequency	1.0
	processor p3	ACE::CurrentFrequency	0.6
	processor p4	ACE::CurrentFrequency	0.0

(Security class is described inside parenthesis)

Conclusion

- We present a new modeling approach called, *Open Analytic Runtime models* where analysis and their interface with the model is no longer hidden in tool implementation
- *Analysis Annex* is presented as an implementation of OAR models for AADL
- OAR models enables the definition of standard analysis libraries with a standardized implementation that avoids diverting interpretations by different parties

Questions?