Software Patterns for Traceability of Requirements to Finite State Machine Behavior:

Application to Rail Transit System Design and Management

Parastoo Delgoshaei and Mark Austin

Department of Civil and Environmental Engineering
The Institute for Systems Research
University of Maryland, College Park, USA

Technical Presentation, INCOSE, Rome, Italy, July 11, 2012

Outline

Problem Statement

Current Trend :

Increasing demand for engineering applications in which long-term managed evolution and/or managed sustainability is the primary development objective

Challenges :

- Understanding for how and why system entities are connected together
- Formal procedures for assessing the correctness of system operations
- Estimating system performance
- Understanding trade spaces involving competing design criteria

Proposed Solution :

- Mechanisms where requirements are connected to models of engineering
- Entities with traceability connections through one or more ontology classes (ontology-enabled traceability). Moreover, to connect these ontology concepts to the engineering objects behavior modeled with finite state machines.

Objective and Scope

Study and Understand

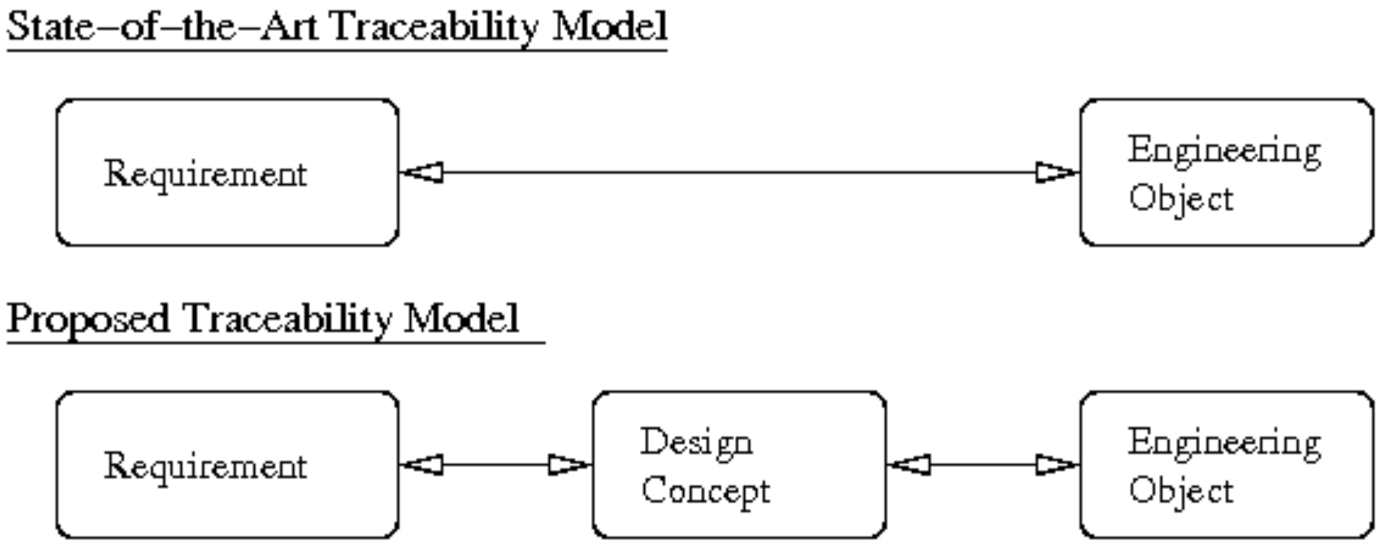
- Software design patterns (e.g., model-view-controller, mediator, observer, adapter, composite, visitor)
- Mixtures of graph visualizations (state machine, ontology)
- Mixtures of tree visualizations (requirements, rules, constraint)
- Semantic web technologies (Web Ontology Language, Pellet Reasoner, Semantic Web Rule Language SWRL)

Implement

Traceability mechanisms from requirements to elements of finite state machine behavior (e.g., actions, states, transitions and guard conditions) providing early design analysis, verification, and validation of information age engineering systems

Here’s What’s New…..

New idea: Ontology-enabled Traceability Mechanisms.

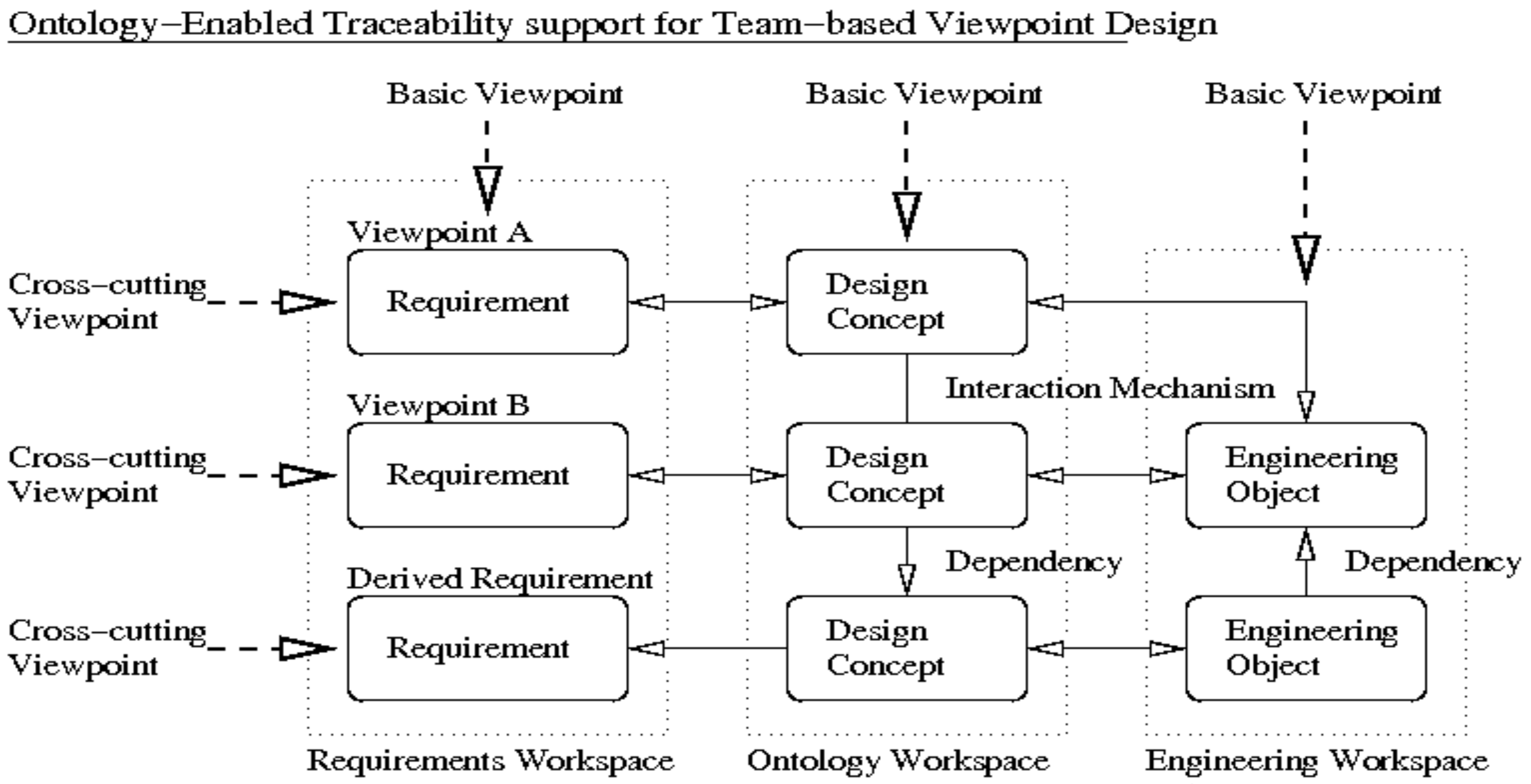


Approach: **Requirements are satisfied through implementation of design concepts**. Now traceability pathways are threaded through design concepts.

Key Benefit: Rule checking can be attached to “design concepts” (ontology), therefore, we have a pathway for **early validation**.

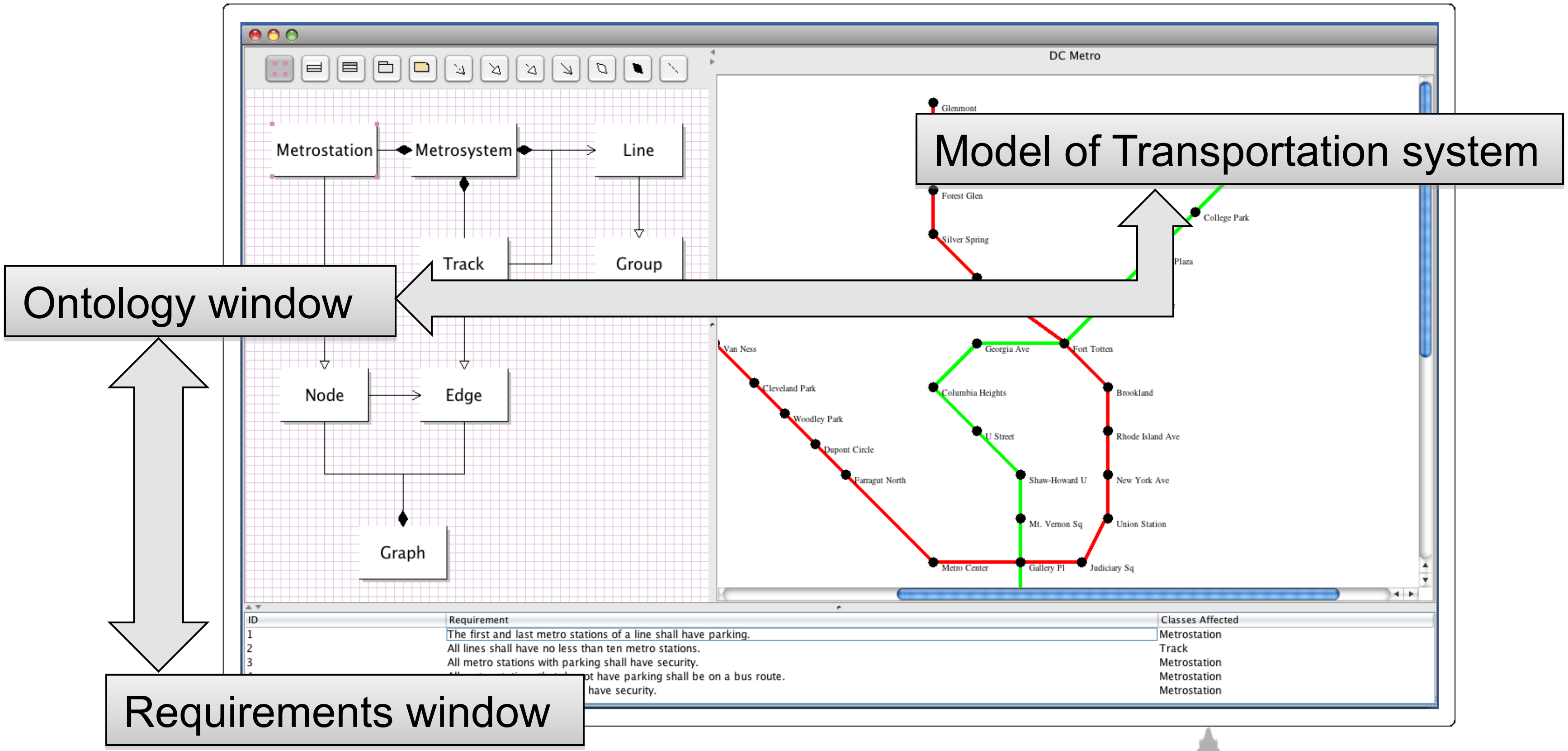
Support for Multiple-Viewpoint Design

Team-based design is a multi-disciplinary activity. We need a model for multiple-viewpoint design and mechanisms for capturing interactions between design concerns.



Prototype Implementation: Ontology – Enabled Traceability For Washington D.C. Metro System

Very simple, UML representation for one ontology. All traceability relationships are hard-coded. Visualization cuts across stages of system development.

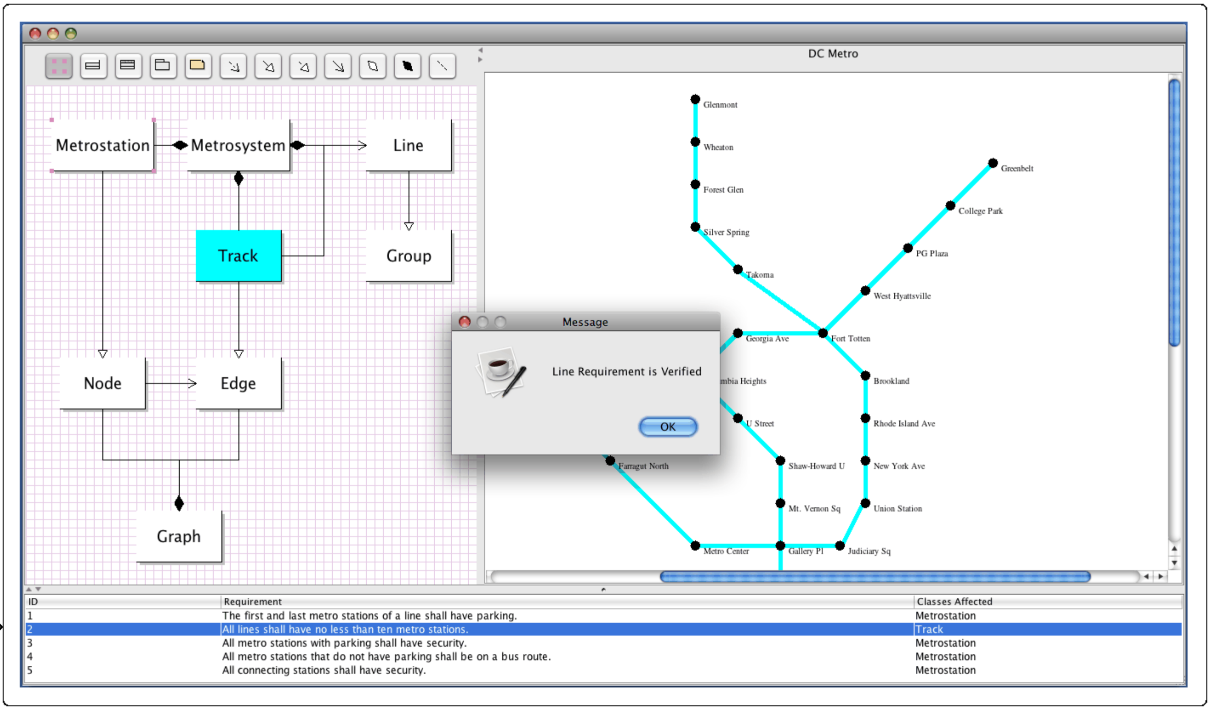


ID	Requirement	Classes Affected
1	The first and last metro stations of a line shall have parking.	Metrostation
2	All lines shall have no less than ten metro stations.	Track
3	All metro stations with parking shall have security.	Metrostation
[illegible]	All metro stations that do not have parking shall be on a bus route.	Metrostation
[illegible]	[illegible] have security.	Metrostation

Credit: Cari Wojcik, MS Thesis, 2006.

Prototype Implementation: Ontology – Enabled Traceability (with very basic rule checking)

Key Advantage: **Design rules and procedures for design rule checking can be attached to ontologies**.



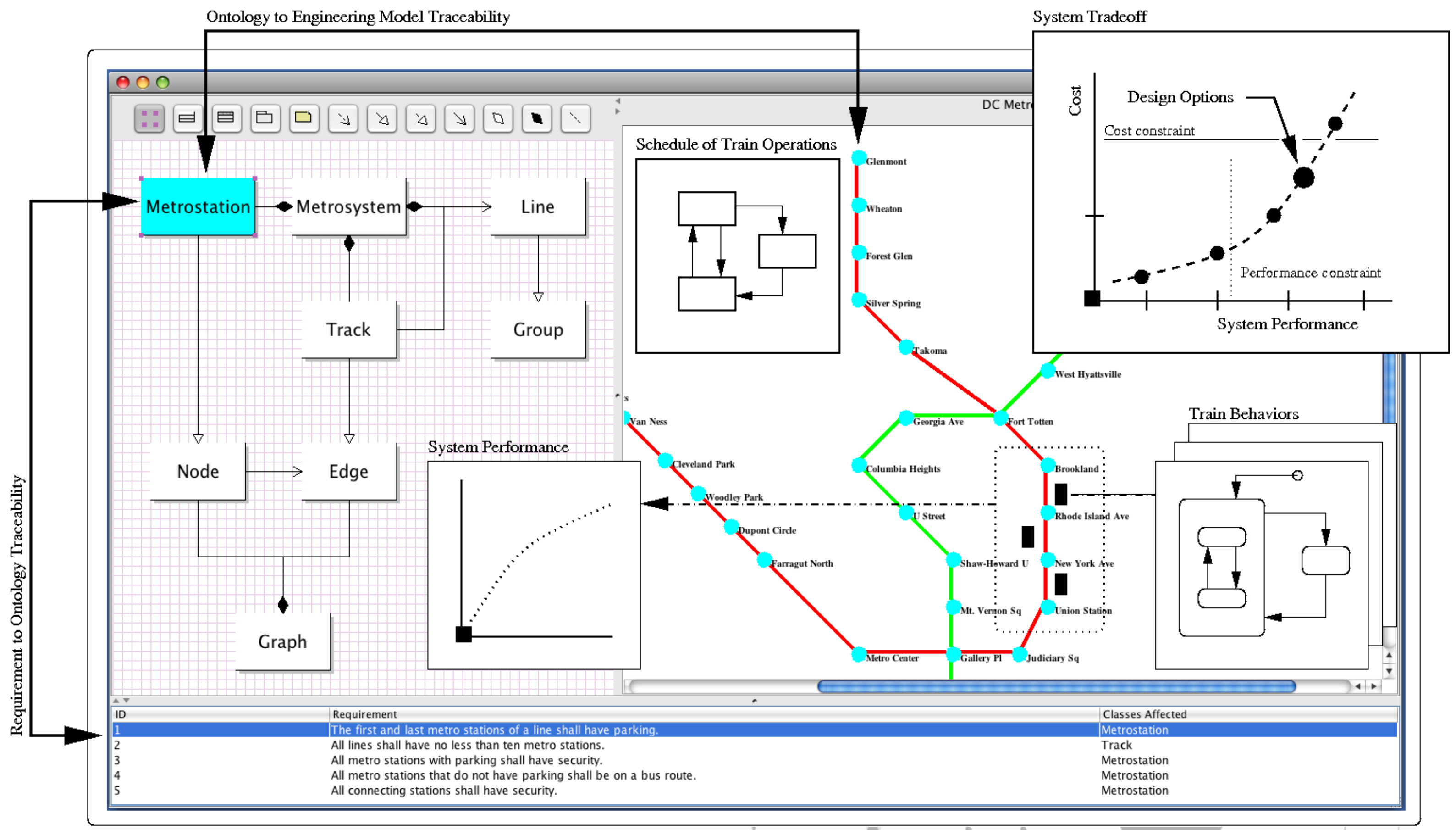
Line Requirement is Verified

ID	Requirement	Classes Affected
1	The first and last metro stations of a line shall have parking.	Metrostation
2	All lines shall have no less than ten metro stations.	Track
3	All metro stations with parking shall have security.	Metrostation
4	All metro stations that do not have parking shall be on a bus route.	Metrostation
5	All connecting stations shall have security.	Metrostation

Design rule checking is triggered by double clicking on a requirement. Visualization shows the extent of ontologies and engineering entities involved in the rule checking.

Schematic For Version II

- Creating workspaces and multiple views for system requirements, design ontologies, and engineering developments.
- Adding time component and state machine diagram for modeling behavior.



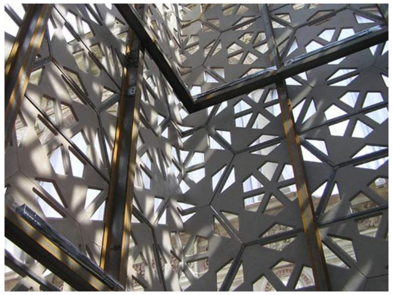
ID	Requirement	Classes Affected
1	The first and last metro stations of a line shall have parking.	Metrostation
2	All lines shall have no less than ten metro stations.	Track
3	All metro stations with parking shall have security.	Metrostation
4	All metro stations that do not have parking shall be on a bus route.	Metrostation
5	All connecting stations shall have security.	Metrostation

Design Patterns

Motivation: Experienced designers know that instead of returning to first principles, routine design problems can be best solved by adapting solutions to designs that have worked well for them in the past.

Definition: A design pattern is simply….

A description of a recurring problem
AND
A description of a core solution to that problem stated in such a way that it can be reused many times.



Software Design Patterns: A few examples …

Behavioral	Structural	System
Command	Adapter	Model-View-Controller
Interpreter	Bridge	Session
Mediator	Composite	Router
Observer …	Decorator …	Transaction …

MODEL-VIEW-CONTROLLER (MVC) DESIGN PATTERN

Approach and Benefits

Divide a component or subsystem into three logical parts – model, view, and controller – making it easier to modify or customize each part.

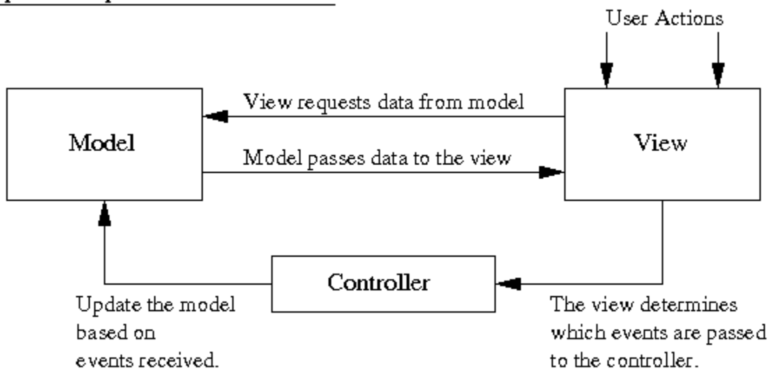
Purpose of Logical Components:

Model: Store the element's state and provide a means for changing the state.

View: Representation of the component or subsystem.

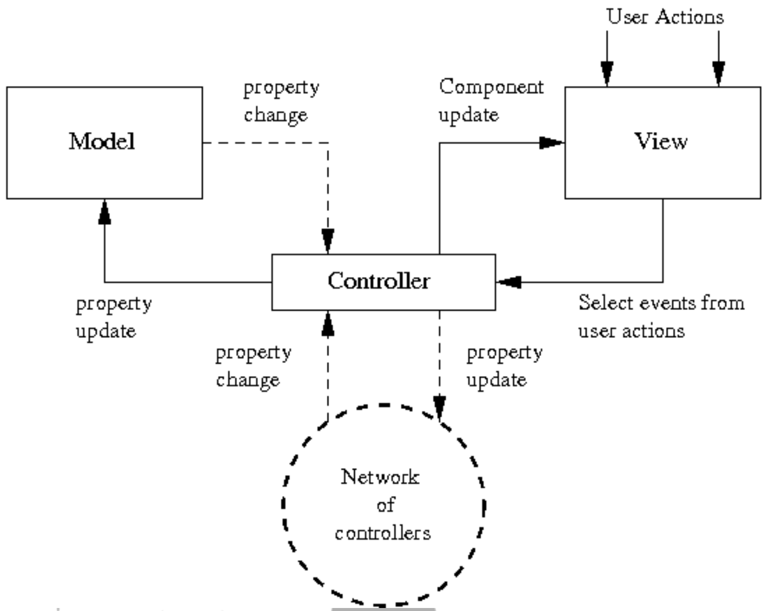
Controller: Map incoming actions to their impact on the model.

Simplified Implementation of MVC

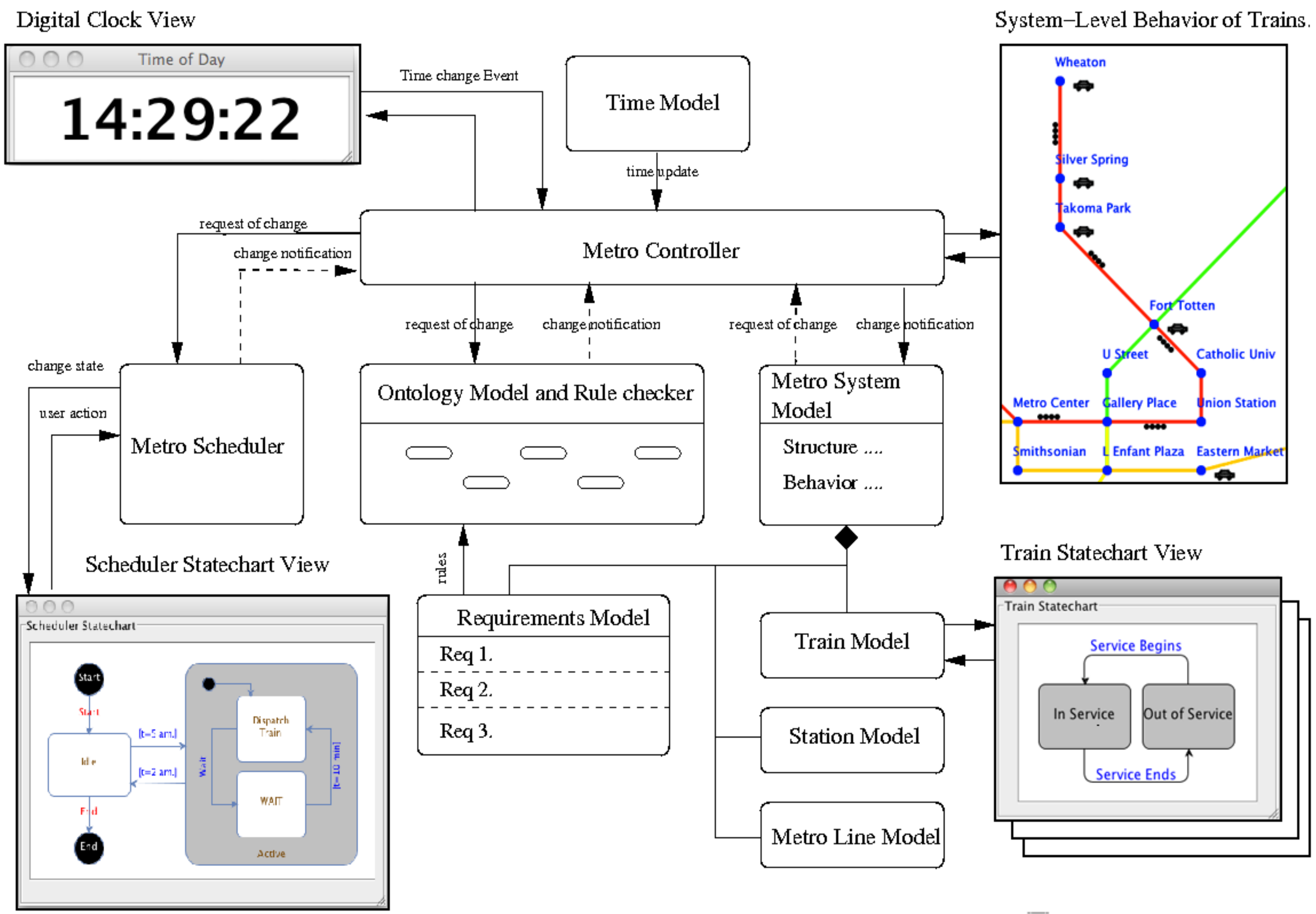


User Actions
View requests data from model
Model
Model passes data to the view
View
Controller
Update the model based on events received.
The view determines which events are passed to the controller.

Implementation of MVC with the Controller acting as a Mediator



Architecture of Current Prototype – A Graph of Communicating Finite State Machines



Requirements Workspace

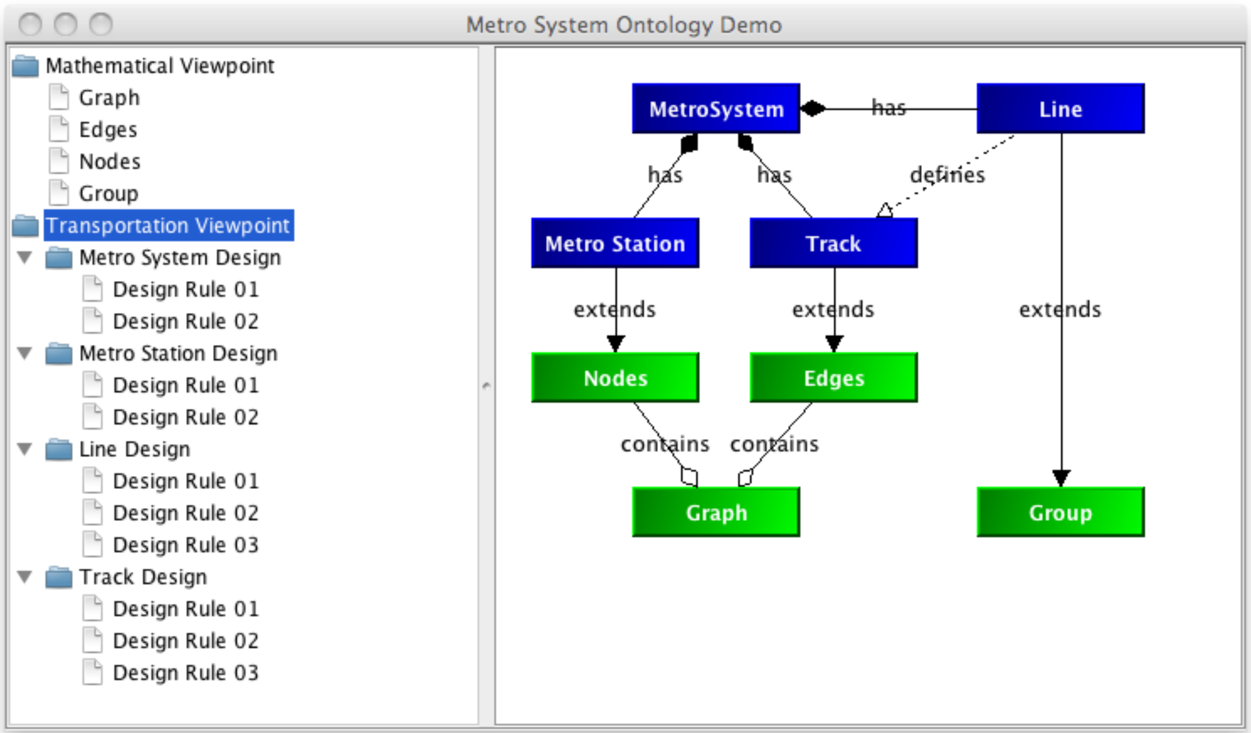
- Requirement Model : Captures textual description of a physical and functional need that the railway system component must be or perform.
- Requirement View: A graph, tree or table representation of the requirement model

Scheduler Requirements	Expected Behavior	Train Requirements	Expected Behavior
The metro system will open at 5 am.	Trains will start running on engineering view when clock shows 5 am.	Trains park at the end of the line when not in use	Train state chart will be in "Park" state
The metro system will close at 2 am.	Trains will stop running and will be parked when clock shows	Trains need not to stop at every station	Train state chart can be in "stop" or "go" in super state "At the Station"
Trains will be dispatched every 10 minute	Train leaves station at the end of the line every 10 minute	Train listen to the scheduler for dispatch time	Train state chart will be in "listen to scheduler" sub state
		Train creates its own timetable	Train state chart will be in "Create timetable" state
		Minimum time to go is 1 minute	The guard condition, [t >1 min], from "Stop" to "Go" state will be satisfied

Requirements to Ontology Traceability

Challenge: Use adapter, observer, composite and model-view-controller patterns to synchronize visualizations in response to user inputs/actions and internal system state change.

Rules are attached to design concepts



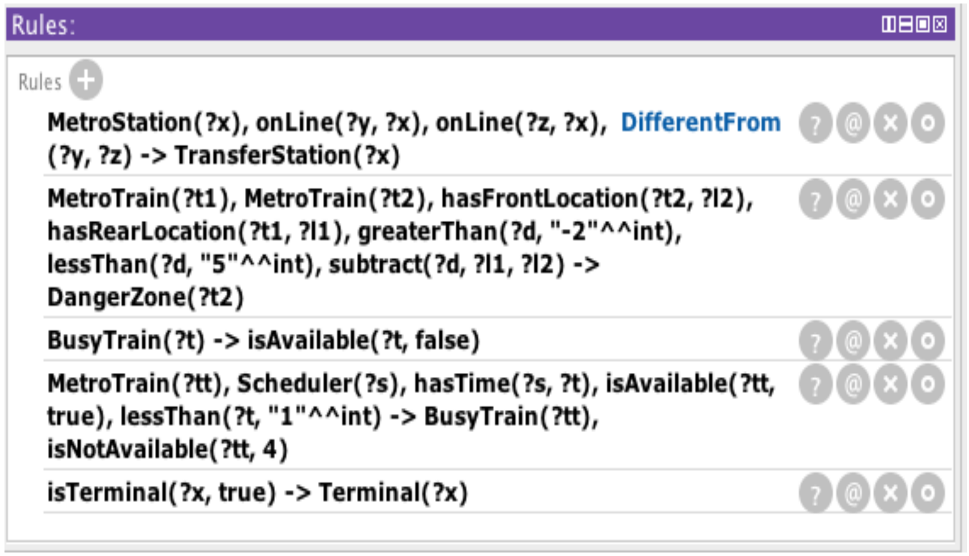
Ontology and Reasoner Workspace

Ontology :

Metro Ontology is created with OWL using Protégé framework.
Ontology instances, objects are added or modified through Jena framework.

Reasoner :

- Pellet reasoner takes action following a change in the ontology model.
- Pellet reasoner supports Semantic Web Rule Language (SWRL), ensuring the rules are satisfied through internal state changes of the system.
- The reasoner decision may involve removing/adding new instances.



```
MetroStation(?x), onLine(?y, ?x), onLine(?z, ?x), DifferentFrom (?y, ?z) -> TransferStation(?x)
MetroTrain(?t1), MetroTrain(?t2), hasFrontLocation(?t2, ?l2), hasRearLocation(?t1, ?l1), greaterThan(?d, "-2"^^int), lessThan(?d, "5"^^int), subtract(?d, ?l1, ?l2) -> DangerZone(?t2)
BusyTrain(?t) -> isAvailable(?t, false)
MetroTrain(?tt), Scheduler(?s), hasTime(?s, ?t), isAvailable(?tt, true), lessThan(?t, "1"^^int) -> BusyTrain(?tt), isNotAvailable(?tt, 4)
isTerminal(?x, true) -> Terminal(?x)
```

Sample SWRL repository

Engineering Workspace

Structural Aspect :

- Engineering objects with associated attributes: status, coordinates, geometry, size

  Implementation : Engineering maps, Design artifacts

  Examples: Metro Station, Metro line, Track, Train

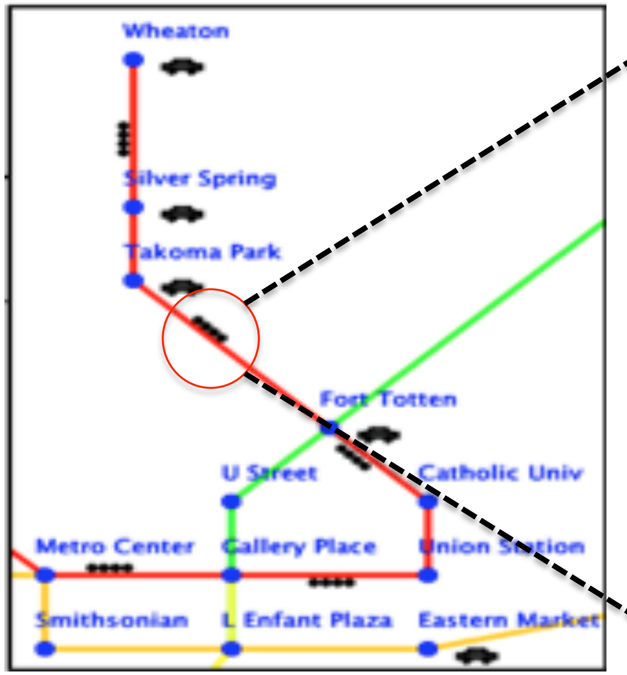
Behavioral Aspect :

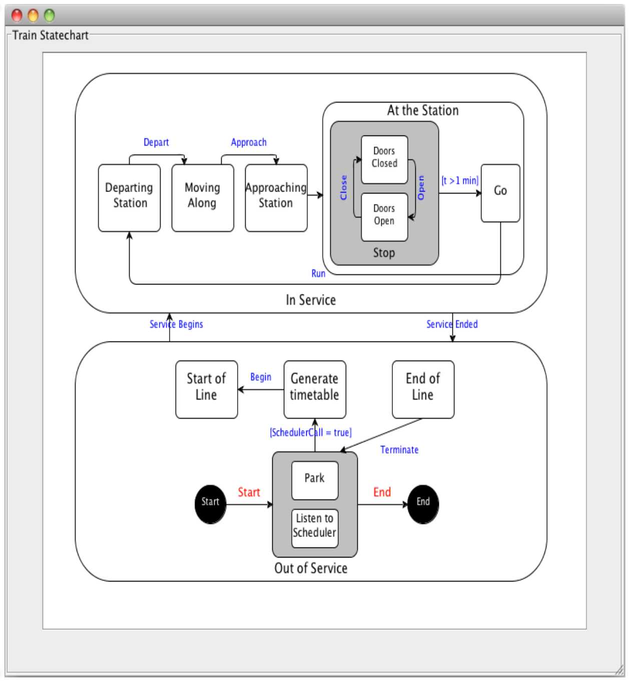
  Implementation : Statechart diagram

  Examples: Train, Scheduler

Engineering Workspace

Plan View (structural and behavioral view) versus Statechart View (behavioral view)

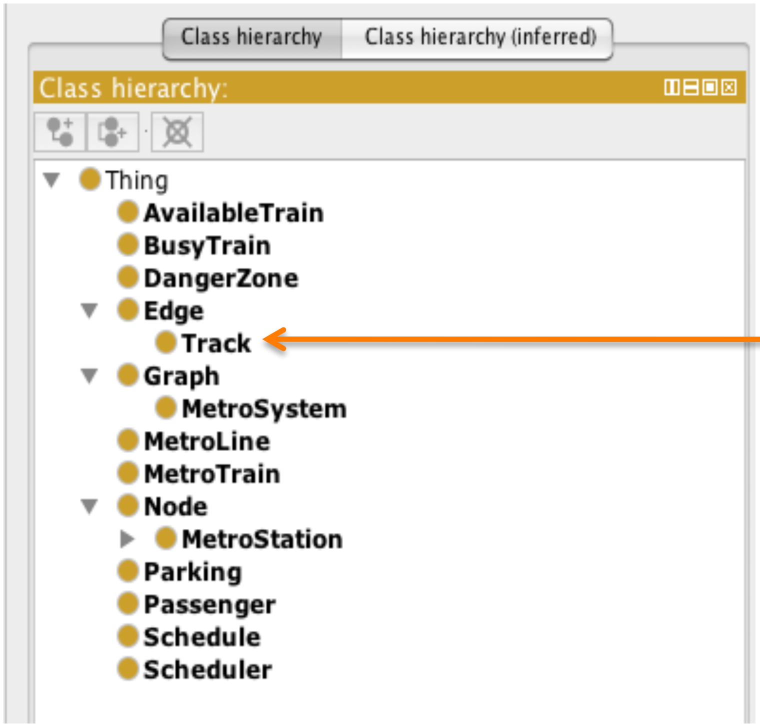


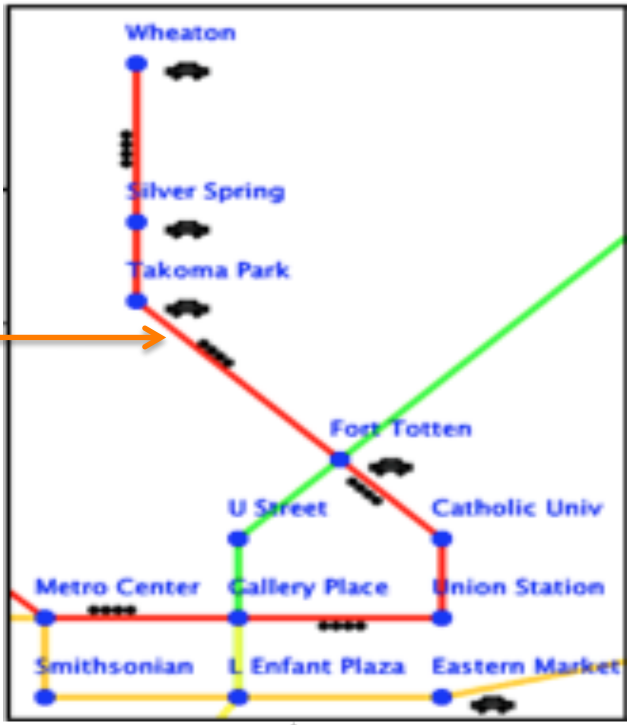


The train statechart transitions between states as the train moves along the track

Ontology to Engineering Traceability

Challenge: To synchronize visualizations in the ontology domain with visualizations in the engineering workspace.





Requirements-to Statechart Traceability

Requirement level (textual representation)

The metro system will start working at 5 am.

Rule level (SWRL)

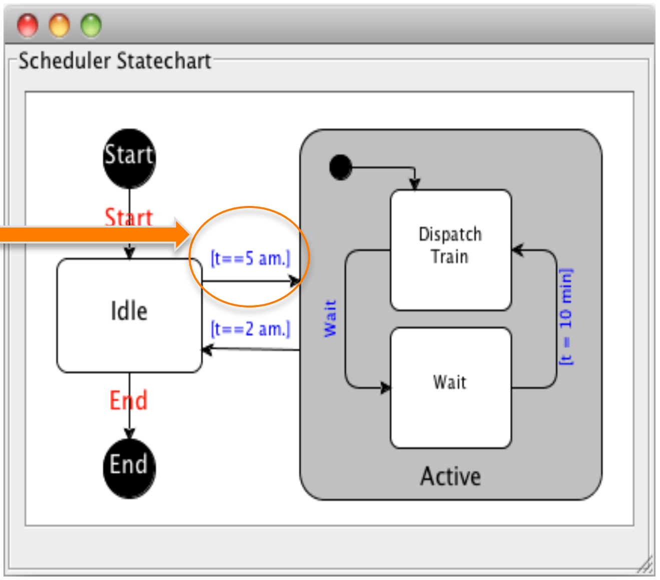
scheduler(?s)^ hasTime(?s,?t) ^ swrlb:greaterThan(?t,5) ^ train(?tr)

^ isAvailable(?tr,true)=>sendTrain(?s,?tr)

Guard Statement

The transition from idle to active is conditional on

" [t == 5 am.]" evaluation results.



Expected Behavior

- The scheduler statechart will transition from idle to active at 5:00 am.
- The statechart of at least one train will transition to the "At Station" state.

Requirements-to Statechart Traceability (2)

Requirement level (textual representation) => The distance between two trains is not allowed to be less than 2 meters.

Rule level (SWRL) => train(?t1) ^ train(?t2) ^ hasLocation(?t1, ….

State Transition => The train's statechart will transition to "stop" state.

Expected Behavior

- The train in the danger zone will stop moving in the plan view. It starts moving when the safe distance rule is no longer violated.

Summary and Conclusion

- Phase II of this research provides a framework for enhanced traceability mechanisms.
- Requirements are translated to the rules attached to ontology classes
- The reasoner performs rule checking in response to changes in the system.
- Requirements are traced to elements of finite state machine behavior (e.g., actions, states, transitions and guard conditions)
- Ontology classes are connected to the set of requirements they need to satisfy, and statecharts are connected to engineering objects (e.g., trains).
- The traceability thread between requirements, ontology and statechart components results in better verification procedures and improved awareness of system operations.

FUTURE WORK AND POTENTIAL BENEFITS

Proposed Work (2012 and beyond):

1 Applying visitor and composite design patterns to integrate all views across the workspaces into a single unified framework.

2 Enhance package to support simulation (performance assessment) and trade studies.

3 Use composite design patterns for targeted/hierarchical visualization of workspace content.

Benefits

- Fewer design/management errors due to superior representation of traceability relationships.
- Support for scalability and multiple viewpoint system architecture.
- Built-in support for design rule checking at the earliest possible moment (Semantic Web).
- Formal verification procedures and improved awareness of system operations.

THE END!

Questions?