Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2015) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

2015 Conference on Systems Engineering Research

Distributed System Behavior Modeling with Ontologies, Rules, and Message Passing Mechanisms

Mark Austin, Parastoo Delgoshaei, and Alan Nguyen*

Associate Professor, Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
Graduate Student, Ph.D. Program in Civil Systems, Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, USA. Systems Engineer, Constellation Energy, Washington D.C.

Abstract

Modern societal-scale infrastructures (e.g., buildings, roads, railways, and power supplies) that are defined by spatially distributed network structures, concurrent subsystem-level behaviours, distributed control and decision making, and interdependencies among subsystems that are not always well understood. During both Hurricanes Katrina and Sandy, it quickly became evident a disturbance in one system can impact other networks in ways that are both unexpected and undesirable. Such outcomes put engineering designers and urban planners (decision makers) in a tough spot where quantitative decision-making regarding the adequacy of system infrastructure is complicated by the presence of newfound system interactions. This paper takes a first step toward providing designers and planners with computational support for simulation of distributed system behaviours with system-level interactions. We describe an experimental software prototype for distributed event-based system behaviour modelling with ontologies, rules checking and message passing mechanisms. Key features of the software architecture are demonstrated through the development of two scenarios: (1) A family interacting with a school system, and (2) Simulation of adjustments to the Washington DC Metro System schedule in response to a severe storm.

Peer-review under responsibility of the scientific committee of Stevens Institute of Technology.

Keywords: Distributed System Modeling, Ontologies, Rule Checking, Jena API

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: author@institute.xxx

Peer-review under responsibility of the scientific committee of Stevens Institute of Technology.

1. Problem Statement

One of the important outcomes of remarkable advances in computing and communications (especially over the past twenty five years) is the way in which they have opened doors to the design and management of new types of systems that have superior levels of performance, extended functionality and good economics. While end-users applaud, model-based systems engineers are faced with a multitude of new design challenges that can be traced to the presence of heterogeneous content (multiple disciplines; multiple physics), network structures that are spatial, multi-layer, interwoven and dynamic, and behaviors that are distributed and concurrent. These characteristics are found in a wide range of modern systems -- planes, trains, and automobiles, buildings, large urban environments, even biological and environmental systems – and they make design a lot more difficult than it used to be.

In a decentralized system structure, no decision maker knows all of the information known to all of the other decision makers, yet as a group, they must cooperate to achieve system-wide objectives. Communication and information exchange are important to the decision makers because communication establishes common knowledge among the decision makers which, in turn, enhances the ability of decision makers to make decisions appropriate to their understanding, or situational awareness, of the system state, it's goals and objectives. While each of the participating disciplines may have a preference toward operating their domain as independently as possible from the other disciplines, achieving target levels of performance and correctness of functionality nearly always requires that disciplines coordinate activities at key points in the system operation. And even if the resulting cross-domain relationships are only weakly linked, they are nonetheless, still linked. When part of a system fails, there exists a possibility that the failure will cascade across interdisciplinary boundaries. During Hurricanes Katrina and Sandy, for example, it quickly became evident a disturbance in one system (e.g., rising sea level) can impact other networks (e.g., transportation and electrical networks) in ways that are unexpected, undesirable, and very costly. The underlying tenet of our research is that in order to understand how such failures might be best managed (or contained), we need as a prerequisite, the ability to model the exchanges of data and information at the disciplinary boundaries and to model their subsequent impact within the disciplinary boundaries.

The objectives of this paper are to explore this opportunity. After explaining the relationship of ontologies and rules to our related work in model-based systems engineering, we describe the software architecture for an experimental platform for assembling ensembles of community graphs and simulating their discrete, event-based interactions. We employ the Jena API [1] for the development of ontologies and Jena Rules for the rule definition and representation of domain-specific constraints. In this setup, each ontology is paired with an interface for communication with other ontologies and hosts a set of domain specific rules as well as the system interaction rules. Key features of the software architecture are demonstrated through the development of two scenarios: (1) A family interacting with a school system, and (2) Simulation of adjustments to the Washington DC Metro System schedule in response to a severe storm. Our long-term research objective is to use this framework as a stepping-stone to the study of problems involving simulation of behavior in cities and cascading system failures that occur as the result of extreme external events.

2. Background to Ontologies and Rules

Fig. 1 illustrates the appeal of behavior modeling with ontologies and rules. The upper right-hand side of the figure shows the relationship among classes and properties in a simplified family ontology. A person has properties: hasAge, hasWeight, and hasBirthDate. Male and Female are subclasses (specializations) of class Person. Boy is a specialization of Male. A Child is a Person who may (or may not) attend Preschool. The upper left-hand side of Fig. 1 shows one fact and three rules. Sam is a boy born October 1, 2007. Given a birthdate and a current time, a built-in function getAge() compute's Sam's age. Further rules can be defined for when a person is also a child and when children attend Preschool. The schematic along the bottom of Fig. 1 shows the evolution of a graph defining the properties of Sam as a function of time. Some of the data (e.g., Sam's birthdate) remains constant over time. Other data (e.g., such as whether or not Sam attends preschool) is dynamic and is controlled by the family rules.

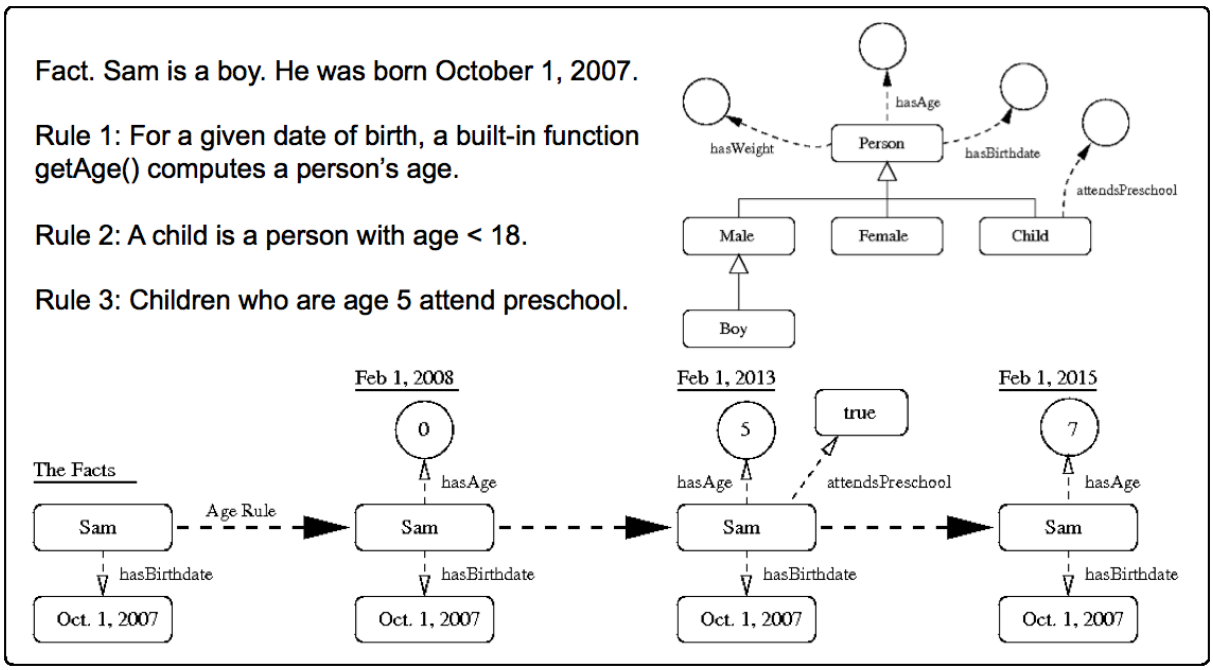

Fig. 1. Simplified framework for modeling with ontologies and rules

From a systems engineering standpoint, this scenario is appealing because it suggests an opportunity for modeling systems behavior and structure with graphs that dynamically evolve in response to events. However the scenario is overly simplistic in the sense that Rules 2 and 3 are presented as part of the family, whereas in reality they are defined by the school system. This observation suggests that instead of modeling the dynamic behavior of systems with centralized control and one large catch-all ontology, we really need to explore mechanisms for modeling networks of discipline-specific (or community) networks. Each community will have a graph that evolves according to a set of community-specific rules, and subject to satisfaction of constraints. Communities will interact when then need to in order to achieve system-level objectives (e.g., put school-age children on a pathway to a good education). If goals are in conflict, or resources are insufficient, then negotiation will need to take place.

3. Related Work in Model-Based Systems Engineering

Model-based systems engineering (MBSE) development is an approach to systems-level development in which the focus and primary artifacts of development are models, as opposed to documents. A tenet of our work is that methodologies for strategic approaches to design will employ semantic descriptions of application domains, and use ontologies and rule-based reasoning to enable validation of requirements, automated synthesis of potentially good design solutions, and communication (or mappings) among multiple disciplines [2-4]. A key element of required capability is an ability to identify and manage requirements during the early phases of the system design process, where errors are cheapest and easiest to correct.

The systems architecture for state-of-the-art requirements traceability and the proposed platform model is shown in the upper and lower sections of Fig. 2. In state-of-the-art traceability mechanisms design requirements are connected directly to design solutions (i.e., objects in the engineering model). Our contention is that an alternative and potentially better approach is to satisfy a requirement by asking the basic question: What design concept (or group of design concepts) should I apply to satisfy a requirement? Design solutions are the instantiation/implementation of these concepts. The proposed architecture is a platform because it contains

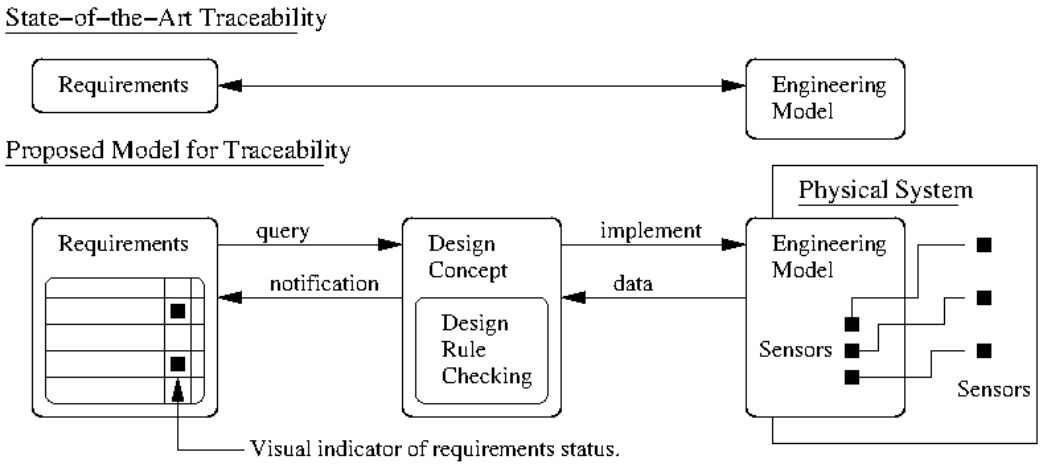

Fig. 2. Schematic for: (top) state-of-the-art traceability, and (bottom) proposed model for ontology-enabled traceability for systems design and management.

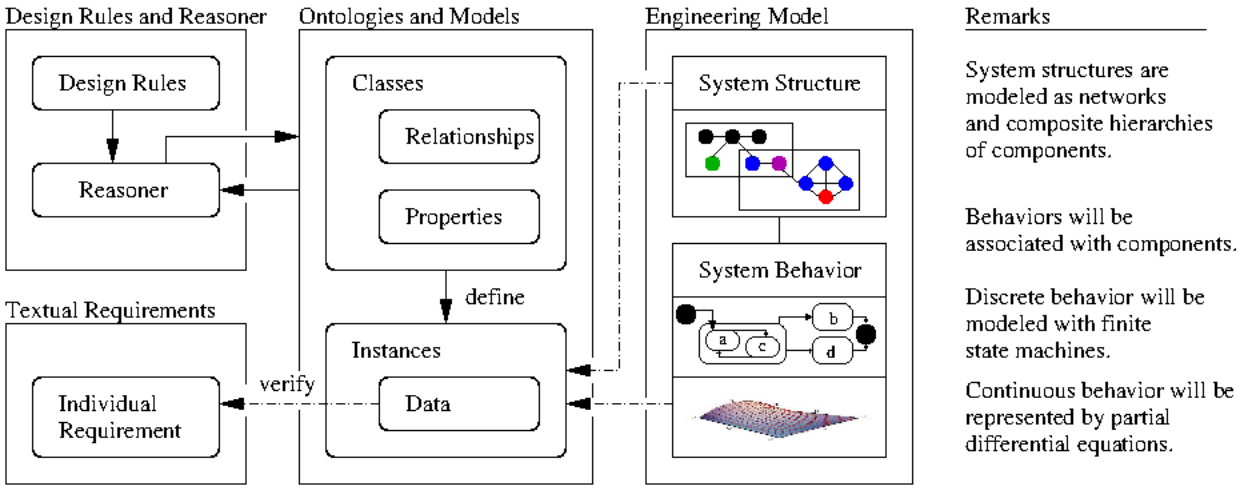

Fig. 3. Framework for the implementation of ontology-enabled traceability and design assessment.

collections of domain-specific ontologies and design rules that will be reusable across applications. In the lower half of Fig. 2, the textual requirements, ontology, and engineering models provide distinct views of a design: (1) Requirements are a statement of ``what is required." (2) Engineering models are a statement of ``how the required functionality and performance might be achieved," and (3) Ontologies are a statement of ``concepts justifying a tentative design solution." During design, mathematical and logical rules are derived from textual requirements, which, in turn, are connected to elements in an engineering model. Evaluation of requirements can include checks for satisfaction of system functionality and performance, as well as identification of conflicts in requirements themselves. A key benefit of our approach is that design rule checking can be applied at the earliest stage possible -- as long as sufficient data is available for the evaluation of rules, rule checking can commence; the textual requirements and engineering models need not be complete. During the system operation, key questions to be answered are: What other concepts are involved when a change occurs in the sensing model? What requirement(s) might be violated when those concepts are involved in the change? To understand the inevitable conflicts and

opportunities to conduct trade space studies, it is important to be able to trace back and understand cause-and-effect relationships between changes at system-component level and their affect on stakeholder requirements. Present-day systems engineering methodologies and tools, including those associated with SysML [5] are not designed to handle projects in this way.

Fig. 3 pulls together the different pieces of the proposed architecture shown in Fig 2. On the left-hand side the textual requirements are defined in terms of mathematical and logical rule expressions for design rule checking. Engineering models will correspond to a multitude of graph structure and composite hierarchy structures for the system structure and system behavior. Behaviors will be associated with components. Discrete behavior will be modeled with finite state machines. Continuous behaviors will be represented as the solution to ordinary and partial differential equations. Ontology models and rules will glue the requirements to the engineering models and provide a platform for simulating the development of system structures, adjustments to system structure over time, and system behavior. This is a work in progress [6].

4. Experimental Architecture for Distributed System Behaviour Modelling

Fig. 4 shows the software architecture for distributed system behavior modeling for collections of graphs that have dynamic behavior defined by ontology classes, relationships among ontology classes, ontology and data properties, listeners, and message passing mechanisms. The abstract ontology model class contains concepts common to all ontologies (e.g., the ability to receive message input). Domain-specific ontologies are extensions of the abstract ontology classes. They add a name space and build the ontology – classes, relationships among classes, properties of classes – for the domain. Instances (see Fig. 3) are semantic objects in the domain.

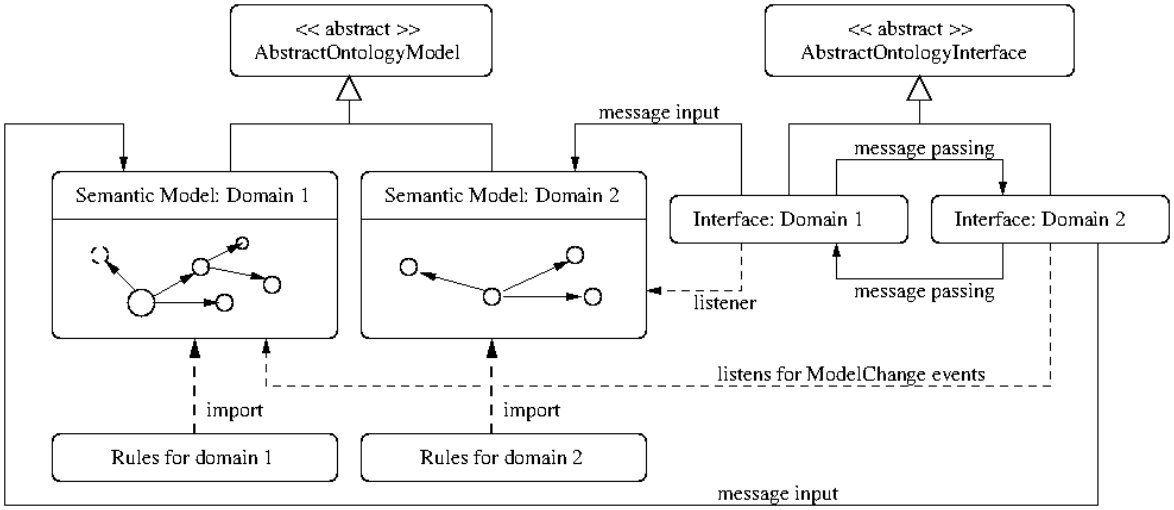

Fig. 4. System architecture for distributed system behavior modeling with ontologies, rules, and message passing mechanisms.

By themselves, the ontologies provide a framework for the representation of knowledge, but otherwise, cannot do much and really aren't that interesting. This situation changes when domain-specific rules are imported into the model and graph transformations are enabled by formal reasoning and event-based input from external sources. Simulations of the type illustrated in Fig. 1 can be accomplished with a single semantic model, a single set of rules, and a clock that marches forward in time. Distributed behavior modeling involves multiple sematic models, multiple sets of rules, mechanisms of communication among semantic models, and data input, possibly from multiple sources. We provide this functionality in our distributed behavior model by loosely coupling each semantic model to a semantic interface. Each semantic interface listens for changes to the semantic domain graph and when required, forwards the essential details of the change to interfaces that have registered interest in receiving notification of such

changes. They also listen for incoming messages from external semantic models. Since changes to the graph structure are triggered by events (e.g., the addition of an individual; an update to a data property value; a new association relationship among objects), a central challenge is design of the rules and ontology structure so that the interfaces will always be notified when exchanges of data / information need to occur. Individual messages are defined by their type (e.g., MessageType.miscellaneous) and a reference to the value of the data being exchanged. The receiving interface will forward incoming messages to the semantic model, which, in turn, may trigger an update to the graph model. Since the basic message passing infrastructure is common to all semantic model interfaces, it makes sense to define it in an abstract ontology interface model.

5. Case Study Problems

The system architecture illustrated in Fig's 3 and 4 open the door to many exciting opportunities for distributed rule-based systems simulation. But there are challenges. For example, in order for the domain specific systems to work nicely together, there has to be a degree of cooperation that involves exchange of messages. While each domain will not need to know everything about the other domains, it does need to know enough to enable data exchange. Cooperation among domains will be complicated if the semantics (e.g., choice of names for classes) are domain dependent. A second open question is as follows: Can the domains cooperate simply by exchanging messages? Will the system operate a lot more efficiently if they exchange messages, plus a small set of rules for how messages are to be triggered and interpreted?

5.1. Case Study A: Behavior Modeling for a Family-School System

To illustrate the capabilities of our experimental architecture, we now present the essential details of a simulation framework for the behavior modeling of a combined family-school system, defined by ontologies, rules, and exchange of information as messages.

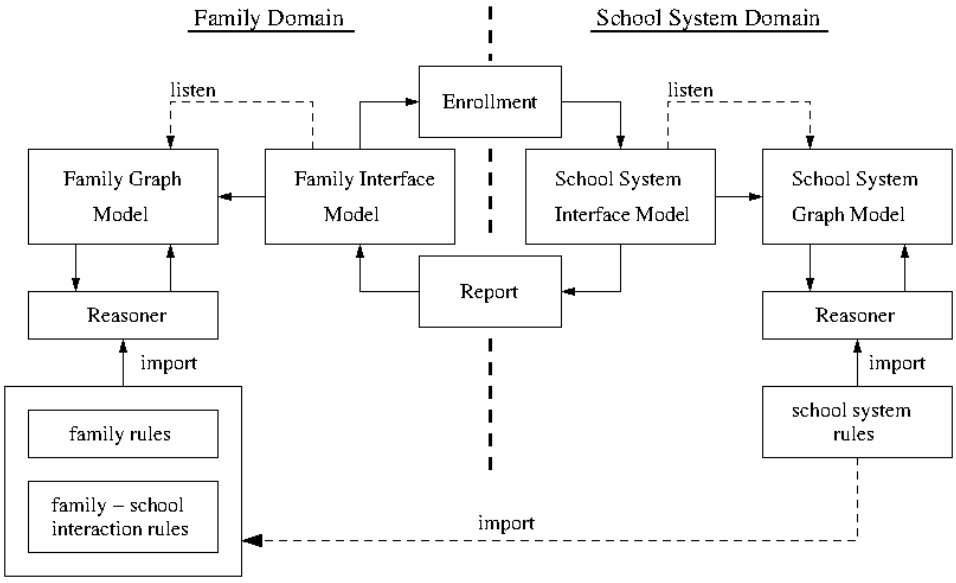

Fig. 5. Software architecture for distributed behavior modeling in the family – school system software prototype.

Fig. 5 is an instantiation of the concepts introduced in Fig. 4 and shows the software architecture for a family-school interaction. We assume in this scenario that a family (i.e., mom, dad, and school-age children) chooses to enroll their children in a nearby school. As every parent knows, the decision to enroll your children in a school system resides with the family. The enrolment process involves the exchange of specific information, such as the name, birth date,

home address and social security number of each child. Then, once the child is accepted the school system takes over. They figure out what grade level is appropriate for each child, what classroom the child will be in, the schedule of learning activities, when the school year will begin, and when it will end. Parents are informed on the schedule for ``professional days'' when, as a result, they need to make arrangements for childcare. If there is a delay in the school opening due to bad weather, parents will be informed, and adjustments to schedules need to be made. Thus, it is evident that comprehensive behavior modeling for a family-school system involves the exchange of information among family, school, weather and transportation systems. Decision making is distributed across the family, transportation and school systems, with dependency relationships defining pathways of cause-and-effect relationships cascading among systems.

Abbreviated Family Rules

```
@prefix af:  <http://austin.org/family#>.

// Rule 01: Propagate class hierarchy relationships ....

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) -> [ (?a rdf:type ?y) <- (?a rdf:type ?x)] ]

// Rule 03: Compute and store the age of a person ....

[ Age01: (?x rdf:type af:Person) (?x af:hasBirthDate ?y) getAge(?y,?z) -> (?x af:hasAge ?z) ]
```

Abbreviated Family-School Interaction Rules

```
@prefix af:  <http://austin.org/family#>.

// Rules 02: Children aged 6 through 18 attend regular school ....

[ School01: (?x rdf:type af:Person) (?x af:hasBirthDate ?a)
            getAge(?a,?b) ge(?b, 6) le(?b, 18) -> (?x af:attendsSchool af:True) ]
```

Abbreviated School Rules

```
@prefix af:  <http://austin.org/school#>.

// Rules 01: Elementary school rules ...

[ Elementary01: (?x rdf:type af:Student) (?x af:hasBirthDate ?a)
                getAge(?a,?b) ge(?b, 6) le(?b, 10) -> (?x af:attendsElementarySchool af:True) ]

[ Grade01: (?x af:attendsElementarySchool af:True) (?x af:hasBirthDate ?a)
           getAge(?a,?b) equal(?b, 6) -> (?x af:isInGrade af:Grade01) ]
```

Fig. 6. Abbreviated Jena rule files for behavior modeling of the family, family-school system interface and school system.

We assume in this case study that behavior modeling for each system will be explicitly defined by sets of rules governing graph transformations. Graph transformations can occur due to input (e.g., the family graph changes because they adopt a new child) or time (e.g., this time next year all of the family members will be one year older than today). Fig. 6 shows an abbreviated list Jena rules for behavior modeling of the family, the family-school system interface, and the school system. Notice that the family and school system rules operate under completely

different name spaces. The family-school interface is a small set of rules that a school system distributes to families informing them when a child is eligible to attend regular school. It is written in such a way that families will understand what to do. When a child turn six, the data property *attendsElementarySchool* will be added to the family graph. When this change is detected by the family interface, an enrollment form is filled out, and the child is off to school.

5.2. Case Study B: Modeling Weather-Transportation System Interactions

Now let us simulate the impact of an extreme weather event on a transportation system. For the weather event, we assume a heavy snowfall in the Washington D.C. metropolitan area. For the transportation system, we model a fragment of the Washington D.C. metro system.

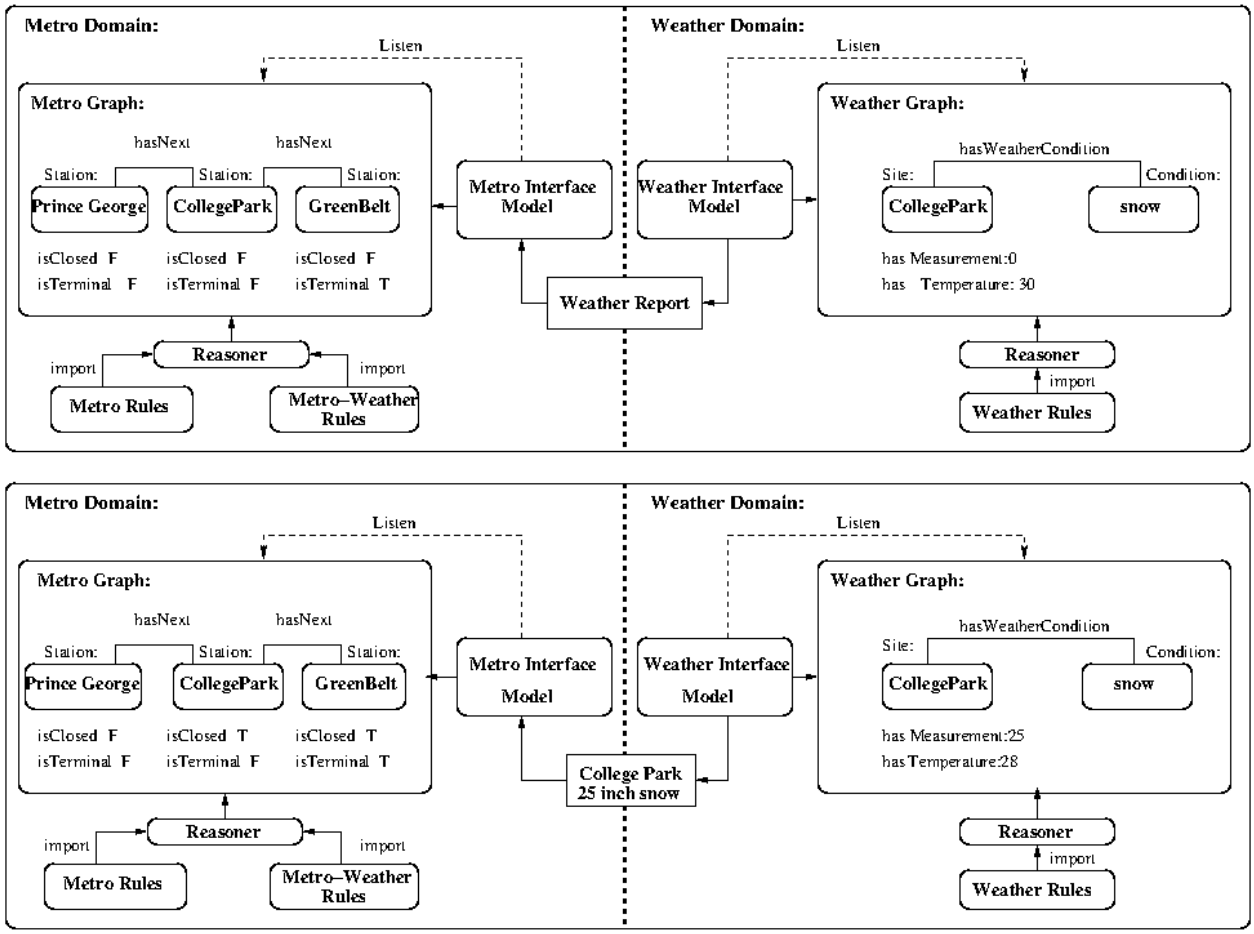

Fig. 7. Schematic of a weather system model impacting a simplified model of the Washington DC Metro System.

The details of our scenario date back to January 1996, when Washington D.C. was battered by a severe winter storm that resulted in the several feet of snow. The Metro System provided continuous train service until a red line train with one hundred passengers got stuck for five hours near Takoma Park. To avoid a repeat of this calamity, now trains only operate below ground during snow storms. If a particular metro station needs to be closed due to excessive snow, then all of the neighboring stations along the line also need to be shut down.

Abbreviated Metro Rules

```
@prefix tr:  <http://transportation.org/metro#>.

//rule01 - rule02: Defining the chain of stations
[ rule01: (?s1 rdf:type tr:Station) (?s2 rdf:type tr:Station) notEqual(?s1,?s2)
          (?s1 tr:hasPrevious ?s2) noValue(?s1 tr:hasNext ?s2)-> (?s2 tr:hasNext ?s1)]

[ rule02: (?s1 rdf:type tr:Station) (?s2 rdf:type tr:Station) notEqual(?s1,?s2)
          (?s1 tr:hasNext ?s2) noValue(?s1 tr:hasPrevious ?s2) -> (?s2 tr:hasPrevious ?s1)]


// rule03: If one station Closed, close the next station

[ rule03: (?s1 rdf:type tr:Station ) (?s2 rdf:type tr:Station) (?s1 tr:hasNext ?s2)
          (?s1 tr:isClosed "true"^^xs:boolean) -> (?s2 tr:isClosed "true"^^xs:boolean)]
```

Abbreviated Weather-Metro Interaction Rules

```
@prefix tr:  <http://transportation.org/metro#>.


// rule01: Shut down the station affected by snow

[ rule01: (?s1 rdf:type tr:Station) (?s1 tr:hasSnowEvent ?m)
          greaterThan(?m,10) -> (?s1 tr:isClosed "true"^^xs:boolean)]
```

Abbreviated Weather Rules

```
@prefix we:  <http://weather.org/weather#>.

//rule01: snow accumulation alert

[ rule01: (?s1 rdf:type we:Site) (?s1 we:hasWCondition ?c)(?c rdf:type we:Snow)
          (?s1 we:hasMeasurement ?m) greaterThan(?m,2)
           -> (?s1 we:snowAccumulation ?m)]
```

Fig. 8. Abbreviated Jena rules for the weather system, metro transportation system, and the weather – transportation interface.

Fig. 7 illustrates the impact of the notifications originating in the weather system model on a simplified model of the Washington, DC. Metro system. The upper part of the figure shows ontology graphs populated with individuals. Specfically, ``CollegePark" is an individual from class "Station" with an object property, "hasNext", and Boolean data properties "isClosed", "isTerminal". The lower half of the figure shows how a change to the weather graph model, a snow event, propagates to the metro graph via ontology interface models. Fig. 8 displays an abbreviated list of Jena rules for the weather system, metro transportation system, and weather-transportation interface. The ``hasNext" property can be used to infer chains of adjacent stations. The weather – transportation rules define conditions for when a specific above ground station should be closed.

With the ontology and rules in place, the interaction of the weather and transportation systems occurs through a sequence of events and data propagations. First, the weather ontology receives a change in its data that "College Park" site has snow accumulation of 25 inches. The graph listener, implemented in weather interface model, receives this change and sends a message, along with College Park weather report to the Metro Interface Model. Next, the Metro Interface Model alters the metro ontology graph. The transportation system reasoner will perform reasoning and new inference data will be inserted into the graph. In this particular case, not only is the College Park station closed, but all of the adjacent (above ground) stations will be closed as well.

A more comprehensive example would include ontologies and rules for the metro system scheduler and individual trains. Rules for system interaction would cover scheduler-train communications, scheduler-weather systems interactions, and metro train behaviors. Under this problem setup, we ought to be able to constrain the trains to operate in only in sections of the track having underground stations. This is a work in progress.

7. Conclusions

Our long-term research objective is to use this framework as a stepping-stone to the study of problems involving simulation of behavior in complex systems, and studies of cascading system failures that occur as the result of extreme external events. The design and management of energy-efficient buildings and modern urban areas are our current target areas of interest. This paper has focused on the design and implementation of a semantic infrastructure (and associated rule processing) to support this effort. While the underlying graph representations are discrete, as illustrated along the right-hand side of Fig. 3., real-world systems have behaviors defined by mixtures of discrete and continuous behaviors. Our future work will include investigation of opportunities for linking discrete-continuous behaviors through the use of libraries of built-in functions within the Jena rules. In the family-school application, this is how the getAge() function is computed, and there seems to be no reason why it cannot be scaled up. A second important topic for future work is linkage of our simulation framework to tools for optimization and tradeoff analysis. Such tools would allow decision makers to examine the sensitivity of design outcomes to parameter choices, understand the impact of resource constraints, understand system stability in the presence of fluctuations to modeling parameter values, and potentially, even understand emergent interactions among systems. In this paper we have exercised the proposed framework with a scenario that involves an environmental system impacting an urban system. However, the framework could also be used to identify and understand ways in which large urban areas are impacting the local environment and causing changes to the global climate.

References

1. Jena - A Java API for RDF. See http://www.hpl.hp.com/semweb/
2. Austin M.A., Mayank V., and Shmunis N. Ontology-Based Validation of Connectivity Relationships in a Home Theater System. 21st International Journal of Intelligent Systems, 21(10):1111–1125, October 2006.
3. Austin M.A., Mayank V., and Shmunis N. PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. Systems Engineering: The Journal of the International Council on Systems Engineering, 9(2):129–145, May 2006.
4. Nassar N., and Austin M.A. Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs. In 11th Annual Conference on Systems Engineering Research (CSER 2013), Georgia Institute of Technology, Atlanta, GA, March 19-22 2013.
5. Fridenthal S., Moore A., and Steiner R. A Practical Guide to SysML. MK-OMG, 2008.
6. Delgoshaei P., Austin M.A, and Veronica D.A. A Semantic Platform Infrastructure for Requirements Traceability and System Assessment. The Ninth International Conference on Systems (ICONS 2014), February 2014.