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

Computational Modeling of City Systems-of-Systems Behavior with Hierarchical Component-Based Networks

*Alan Nguyen, Mark Austin

Graduate Students, Masster of Science in Systems Engineering Program, University of Maryland, College Partk, MD 20742, USA,
Associate Professor, Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, USA.

Abstract

With 70% of the world's population expected to live in cities by 2050, there is an increasing need to model the behaviour of connected city subsystems since, for better or worse, outcomes in one system can strongly affect other subsystems. This paper explores the use of higraph-based frameworks for modelling relationships and cascading system-of-systems behaviour occurring among different city systems. We present a software architecture for the component-based modelling of city subsystems and exercise its capabilities by simulating the effect of a severe weather event on a fragment of the Washington D.C. Metro System.

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

Keywords: model-based systems engineering; city systems; component-based modeling

1. Problem Statement

Modern cities are responsible for two thirds of global energy usage, 60% of water consumption, and 70% of all greenhouse gases produced worldwide. It is now evident that with the Worlds population projected to be in the nine-to-ten billion range by 2050 (with 70% living in urban areas), increasing demand for limited resources will drive the need for major changes in the way we design and manage (e.g., congestion, energy efficiency) urban environments [1]. Much of the current thinking in ``smart cities'' design is that advanced technologies – sensing, wireless

* 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.

communications, public access to information – will play a central role in the management of urban areas. Our contention is that while stove-piped approaches to this problem will have their place, as city system structures and behaviours become progressively complex and interwoven, strategies for efficient urban management will become increasingly dependent on an ability to measure and model concurrent subsystem behaviours and their interactions. In other words, long-term solutions to urban management will require a system-of-systems perspective. This paper lays a foundation for the behaviour modelling and management of city systems in response to extreme weather events. We assume that urban areas are networks of intertwined networks, the latter being defined by a multitude of dependency relationships and flows along arcs connecting networks. This formalism is a good starting point for simplified modelling behaviours in a typical city where transportation networks are dependent on electrical networks, which, in turn, can be affected by extreme events in environmental systems. We explore the use of higraphs for modelling hierarchy and graph relationships, and capturing cascading failures across different city systems. We propose a software architecture for the component-based modelling of city subsystems, and exercise its capabilities by simulating the effect of a severe weather event on a fragment of the Washington D.C. Metro System.

2. System-of-Systems Modeling with Higraphs

Higraphs are mathematical abstractions that combine the notion of Venn diagrams and graphs and allow for depth and orthogonality. The most basic element of a higraph is what Harel refers to as a blob [2,3,4]. A blob is essentially a node in the graph and can contain other blobs. From this point forward, we will define a blob with no sub-elements as a node and a blob that contains sub-elements as a cluster. Edges in a higraph can connect any node or cluster together to represent a relationship between them. Edges can be directed implying a one-way relationship or undirected, implying a symbiotic relationship between elements.

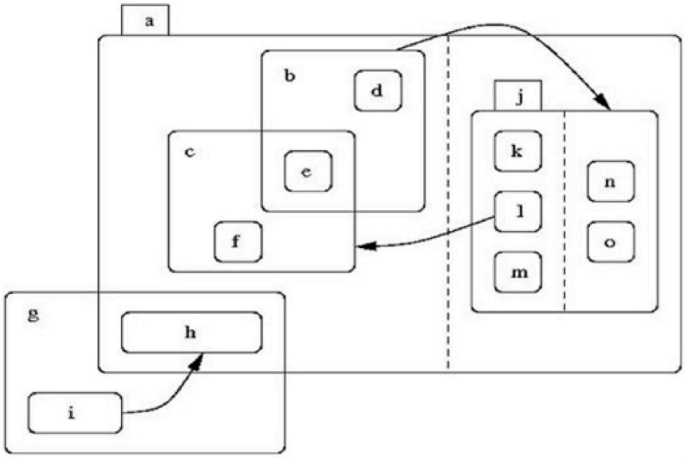

Fig. 1. Example of a higraph.

Fig. 1 is a visual representation of a higraph. A few key points to note are as follows:

1. Nodes can belong to multiple clusters (e.g., a metro station can belong to multiple transportation lines).
2. Single nodes can affect clusters. For example, node l has an arc pointing to cluster c, meaning it will affect all nodes within cluster c.
3. Clusters can affect other clusters. For example, any node in cluster b will affect all nodes in cluster j.

The grouping of nodes to clusters naturally creates a hierarchy of components, which serves two purposes systems of systems modelling. First, it allows for systems to be represented more accurately. Nodes (which represent components) can belong to multiple clusters (which represent systems), which is more representative of real-life cities. The other purpose of higraphs is to allow for different viewpoints. Users can create their own clusters and

group components how they please. Cities, however, are a real-time system, so there must be a mechanism that allows for communication between city systems and state changes. The proposed framework attempts to solve this issue with the introduction of an abstract concept called controllers.

3. Framework for Component-Based Modeling of Cities

Fig. 2 shows a class diagram of the meta-model for the proposed framework. According to the framework, a city is a collection of one or more systems. Systems are sets of components designed to achieve a task. Examples of engineered systems include transportation systems and electrical systems. Of course, the city will also be affected by natural systems, such as weather. There are two types of components that make up a system. The first is called a node, which represents a single object of a system and contributes to the expected behaviour of the system. For instance, train stations may be represented as nodes of a subway system. Each node is defined by attributes that determine its state. A change in the state of a node causes information to be passed to the rest of the city. The second type of component is called a cluster. A cluster is a group of one or more components (i.e., nodes and clusters) with similar characteristics. Clusters can be used to show different views of a system, or can be used to represent different levels of abstraction. Note that the composite hierarchy design pattern was used to represent components of systems. This allows us to treat nodes and groups of nodes in a similar manner.

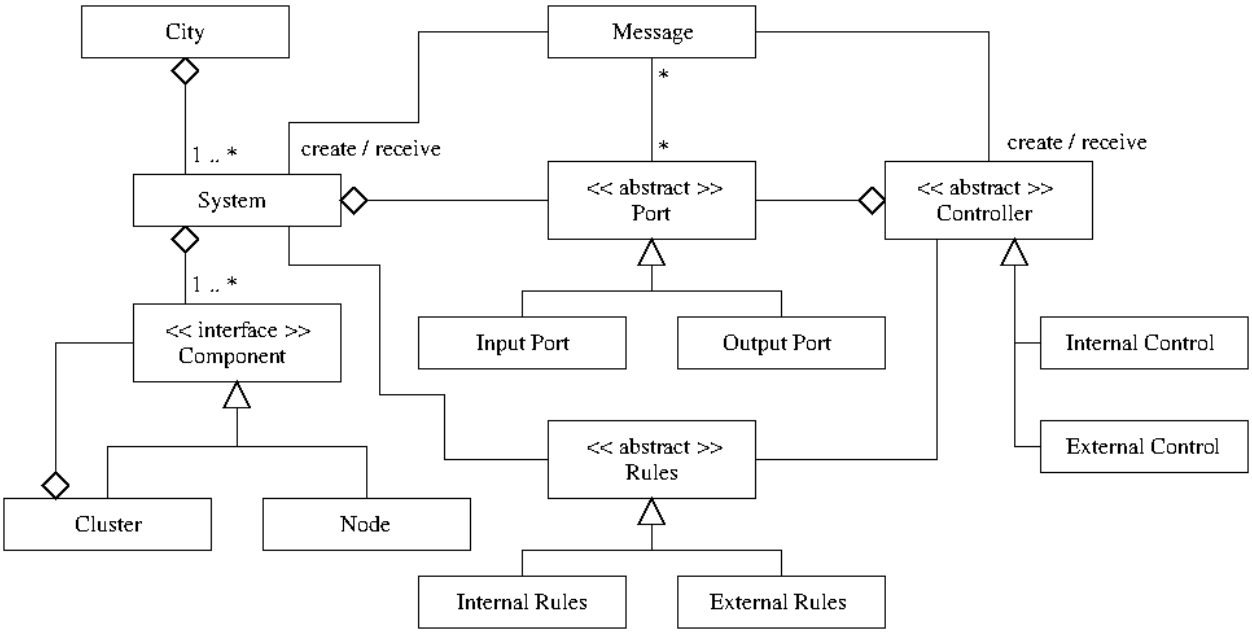

Fig. 2. Meta-model for component-based modelling of city behaviours.

Systems are governed by a set of rules which are contained in an abstract object called a controller. Rules are logic that describes how a change in one system affects the state of components in another system. Controllers and rules can be classified as internal or external. Internal rules define how a state change in one system affects other components in the same system. External rules define how a state change in one system affects components of a different system. Controllers can only contain one type of rule, making them either internal controllers or external controllers. External controllers can only contain rules between 2 distinct systems. This means that a city having n systems, there may be as many as $n(n + 1)/2$ controllers, n of them being internal controllers.

When a component in a system changes states, it creates a message describing the change that occurred. Messages may also originate from controllers, and will represent an effect on a particular node or cluster due to a

state change. Messages are passed to other parts of the model via an abstract mechanism called ports. Ports are used to send and receive messages between systems and controllers. Ports are also responsible for disseminating the information from messages to nodes and clusters. Once a component changes its state, the system will began the process of sending messages to determine if any other systems are affected by this change.

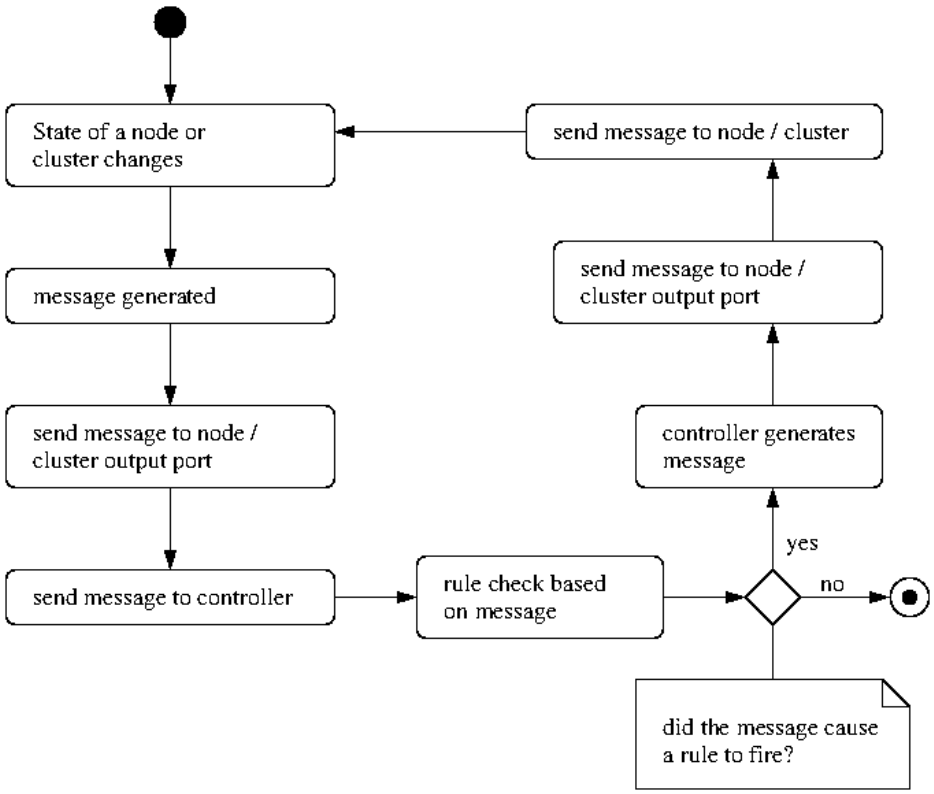

Fig. 3. Activity diagram.

Fig. 3 is an activity diagram that visually shows this process of messages propagating through the system.

4. Hierarchical Block-Diagram Model for Washington D.C.

Fig 4. is a hierarchical block diagram model for the weather, electrical and transportation systems operating in Washington D. Our purpose is to model a transportation system (Washington DC Metro) and an electrical system and how they are affected by a weather event. Clusters are used inside the transportation system in order to group individual nodes by specific characteristics. Each cluster must contain at least one node; however a particular node may belong to multiple clusters. For example, a train station such as Fort Totten belongs to both the green and red line cluster.

Table 1. Node Type and Atrributes

Node Type	Attributes
Transportation	Station name, location, open, closed, parking lot size.
Electric	Name, location, power transmitted, component service, component serviced by.
Weather	Rainfall rate, snowfall rate, wind speed, visibility, temperature, surge, lightening frequency.

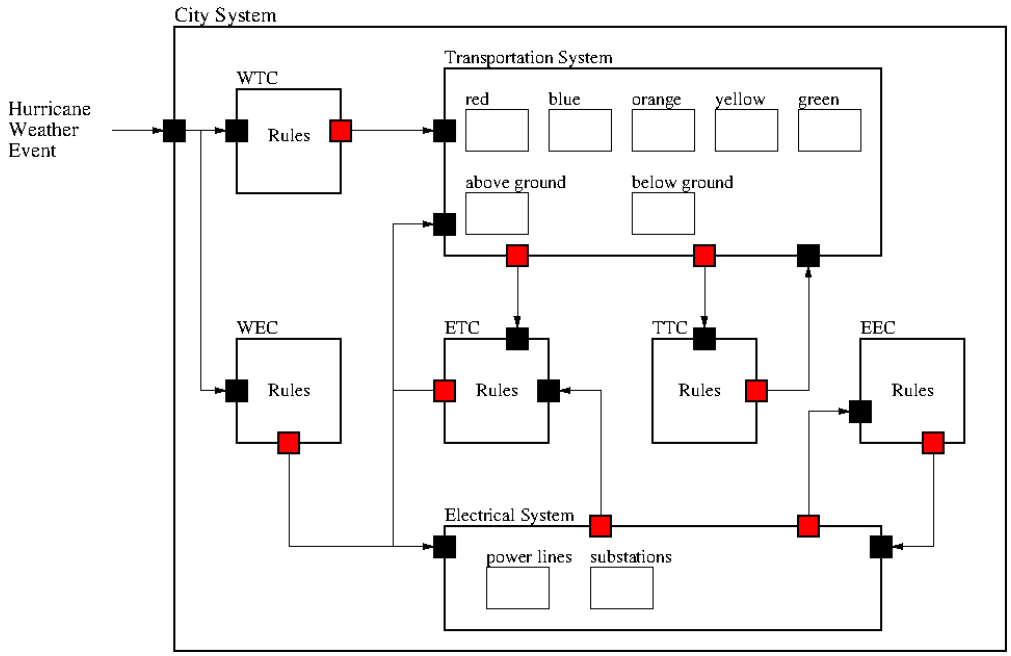

Fig. 4. Hierarchical block diagram model for a city system, subsystems, and rules.

The example contains five controllers, each containing rule sets that control how the states of components in one system affect the states of inter-dependent systems. The controllers have a very specific naming convention that identifies what system produces the incoming message to the system and which system the message that is generated in the controller will be sent to. A controller that is named TTC stands for the Transportation-Transportation Controller. The first word is the system in where the incoming message was generated in and the second is for where the message that is generated in the controller is going to be sent. Thus, our Washington DC model has TTC, ETC,WTC, WEC, and EEC controllers. Controllers that have repeating letters are internal controllers, while controllers with different letters correspond to external controllers. Below are example rule sets that are found within controllers

WTC : Weather - Transportation Controller
If Snowfall Rate >= 8 inch per 8 hours
Close AboveGround Cluster
If Snowfall Rate >= 1 inch per 4 hours
Delay AboveGround Cluster
If Rain Rate >= 4 Inches per Hour
Close Flood Zone Cluster
TTC: Transportation -Transportation Controller
If AboveGround Cluster is closed
Put BelowGround Cluster into limited service
If BelowGround Cluster is closed
Put AboveGround Cluster into limited service

Note how WTC makes changes to the transportation system based off of conditions in the weather system and TTC makes changes to the transportation system based off of conditions in the Transportation system. Information is propagated throughout the model by passing messages to other components via ports and wires. From Fig. 3, the black squares indicate input ports receiving messages while red squares indicate output ports that disseminate

messages. Note that it is possible to send messages from an input port to another input port. This scenario must exist in order to allow rule propagation to clusters.

5. Demonstration of Cascading Failures across City Subsystems

Fig's 5 through 7 illustrate a sequence of system states for a cascading failure propagating throughout the city. Affected portions of the model are highlighted in red. The key points to note are as follows:

1. In Fig. 5 a hurricane is affecting the city. A message is created in the framework with details of the hurricane, such as wind speed and rainfall rate. The message is passed to the WTC and WEC controllers for rule processing.
2. Fig. 6 illustrates the consequences of Step 1. Within the controllers, the rules in red are triggered. In the WTC, a message to close all above ground train stations is sent to the transportation system, while in the WEC, a message that affects power lines is generated and sent to the electrical system.
3. Fig. 7 shows how information is disseminated to the appropriate cluster after messages have been sent to their respective systems. The states of all nodes within the cluster are changed in accordance to the message. At this point, all above ground stations are closed due to the hurricane and overhead transmission lines are damaged because of the high winds. Since the states of the nodes have changed, new messages detailing the changes from the transportation and electrical systems are created and sent to the TTC and EEC controllers for additional rule checking. The damage done to the transmission lines means the substations are without power. A message is sent from the EEC back to the electrical system, closing all substations.

A change in state for the substations creates a new message that is sent to the ETC. Without power coming from the substations, every station in the transportation system is closed. This example shows how this model can predict cascading failures throughout a city. It shows systematically how the effects of a hurricane affect the whole city. At the beginning of this example there was a fully functioning city before the hurricane occurred. However, by the end it showed major failures in the electrical system as well as many closures and limited service of stations in the transportation system. Cities and SoS in general are comprised of much more than two component systems. As the number of components grow, the number of controllers and connections in the framework increases at an exponential rate. Therefore, it is necessary to create software capable of managing and simulating the elements of the framework.

5. Software Prototype in Java

Our software prototype is implemented in Java, and as illustrated in Fig. 8, makes extensive use of hierarchical tree modelling abstractions, interface definitions and message passing mechanisms to send, receive, and process information as information propagates throughout the model. Nodes contain a list of attribute values, but no real (executable) processing. Since there are many different types of nodes (i.e., transportation or electrical nodes), the node class can be extended to create subclasses with unique attributes. The component interface defines all variables and methods to be used in the classes BaseComponent and MetaComponent. BaseComponent is a node that cannot contain other components. MetaComponent is a container for a group of components, including other MetaComponents. Base component are associated with component engines that systematically process streams of data and information. Meta component process methods loop through the list of internal components and call each components process method. To facilitate data flow processing within a graph structure, we attach abstract objects called ports to the components and connect components together with wires. Ports are implemented using a hierarchy of interfaces as showing in Fig. 8. Input ports receive information sent from output ports via wires. Ports also utilize Java generics, meaning they are capable of receiving any type of value. A wire is defined by connecting

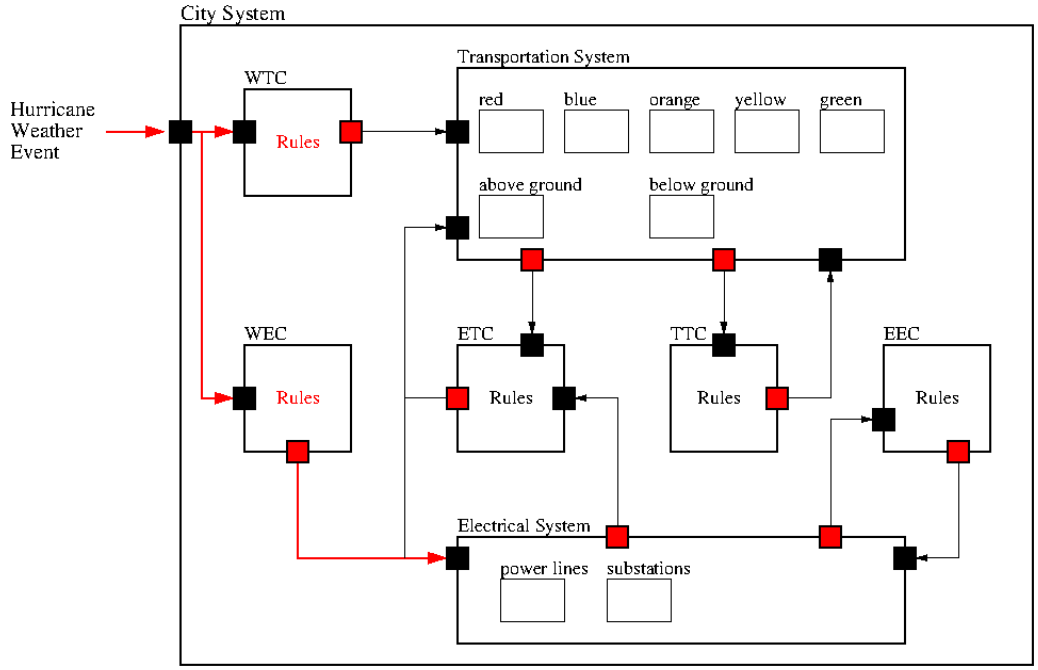

Fig. 5. Hierarchical block diagram for a city system. Propagation of the weather event to the Weather-Transportation (WTC) and Weather-Electrical (WEC) controllers.

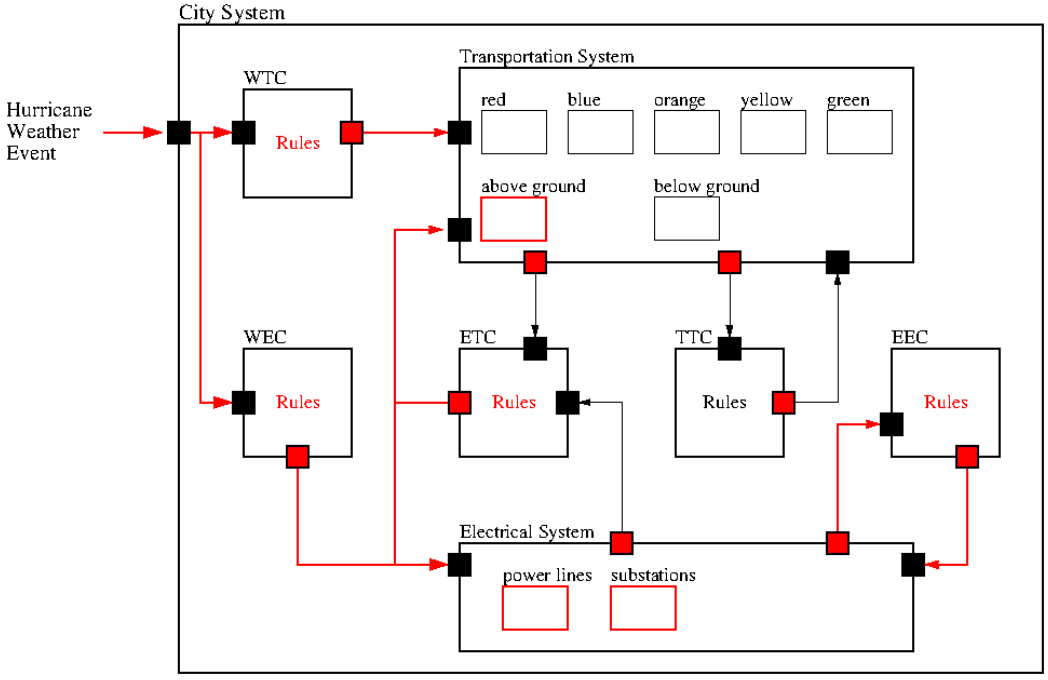

Fig. 6. Hierarchical block diagram of a city system.

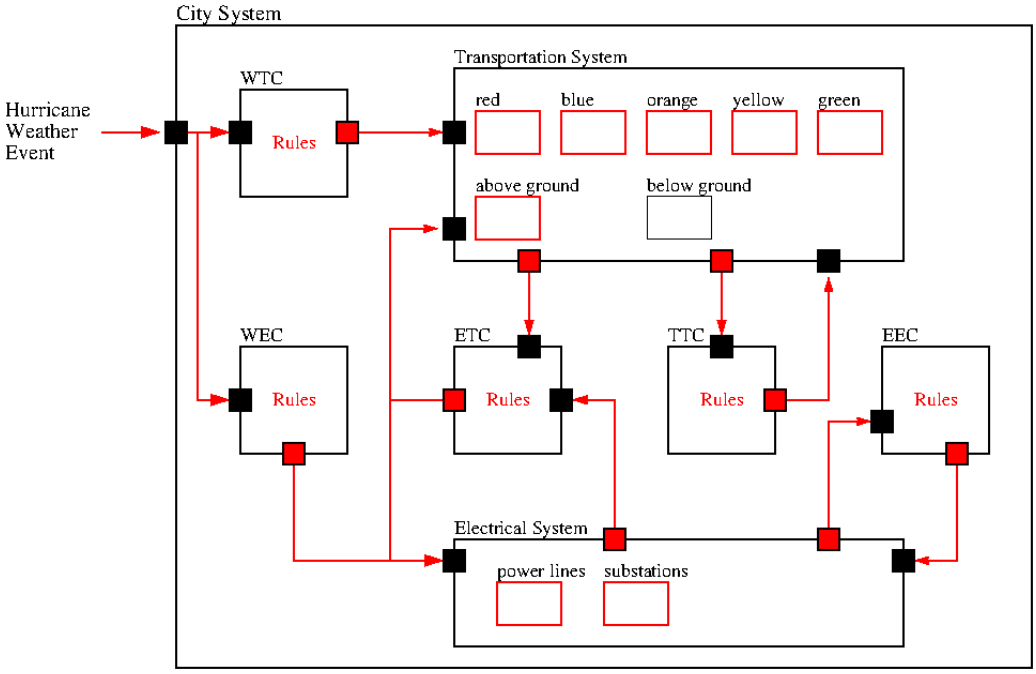

Fig. 7. Hierarchical block diagram of a city system

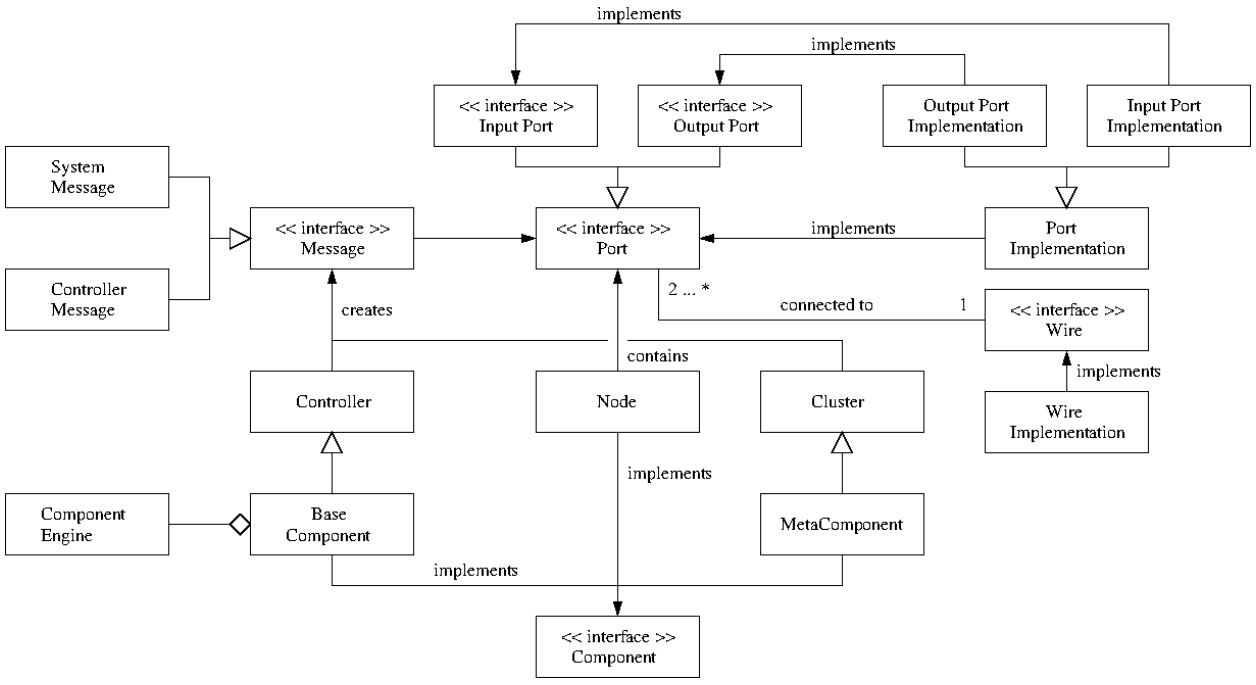

Fig. 8. Architecture of classes and interfaces in the software prototype.

a single output port to one or more input ports, implying a one-to-many connectivity between source and target ports.

There are two types of messages: controller messages and system messages. Controller messages originate from controllers. They change the state of a node or cluster. When the state of a component changes, a system message is sent back to the controller with information about the component's current state and to trigger rule processing. When a node receives a message from the controller, the process() method extracts the attribute affected and the value to be modified. Clusters resemble MetaComponents, but only propagate incoming messages to the components listed in the message. When the state of a node changes, the system message created contains the list of all of the node's attributes and its values for evaluation in the controller. The controller contains a component engine with rules defined for the specific controller. For instance, the TTC controller will contain a TTC component engine containing rules related to transportation related behavior. Likewise, the ETC controller will contain a ETC component engine governing interactions between the electrical and transportation domains. When a rule is triggered, the controller will create a controller message that is sent back to the originating system, applying a change to the node's attributes and creating an additional system message. This process repeats itself until the system reaches a ``steady state" where no rules are triggered and the nodes do not change state.

Fig. 9 shows the essential features of a case study implementation that models the effect of a hurricane weather event in the Washington D.C. Metropolitan area.

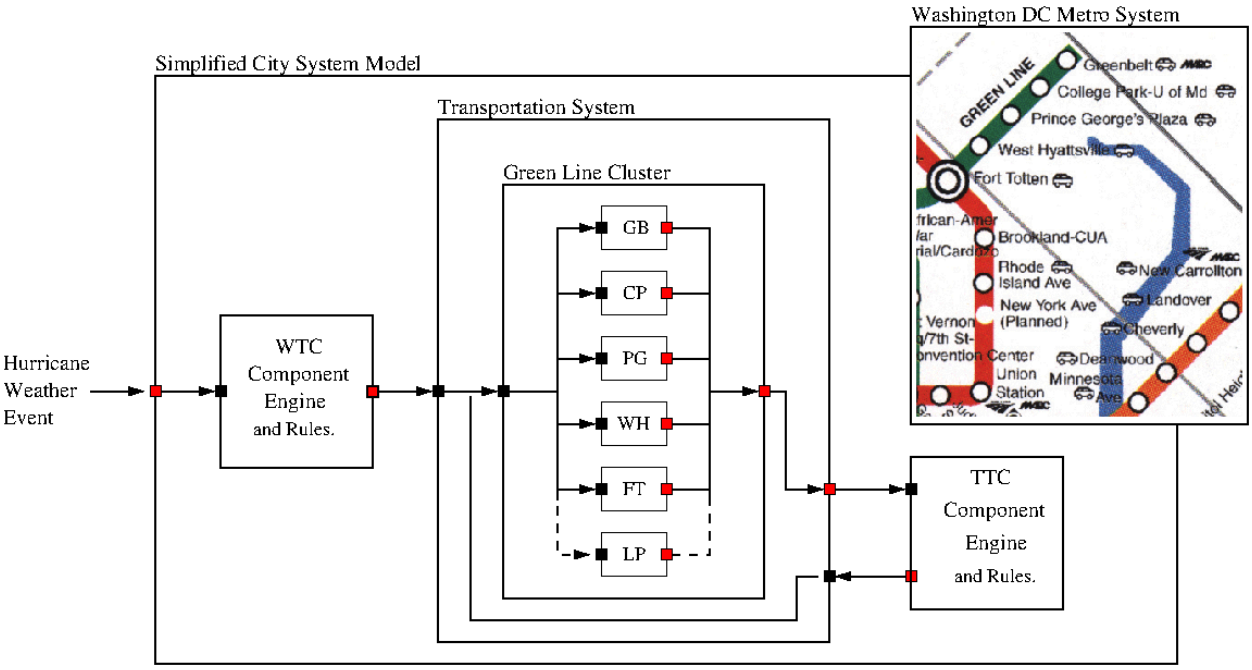

Fig. 8. Hierarchical block diagram model for a fragment of the Washington D.C. Metro system.

To keep things simple, the city only contains a transportation system, a subset of the Washington D.C., WTC and TTC controllers, and a single cluster of metro stations called the ``Green Line.'' Metro stations are subclasses of the class Node. The TTC component engine contains a set of if/else statements that look for the attribute names and performs logic. In this case, we have a rule that states that if water accumulation at the station is greater than 2.5, the station status will be set to closed. The proposed message passing and rule processing scheme is complicated by the presence of loops in the system connectivity. We handle loops by first determining the order in which objects should be processed, and then systematically traversing the hierarchy of graphs. In our prototype implementation, each

component in the model contain a directed graph representation of itself as well as logic for traversing and processing the objects, which we will call the GraphManager. We employ JGraphT (a general purpose library for the representation of graph objects, plus algorithms) for the GraphManager implementation. The vertices of the graph contained in GraphManager will be made up of several types of objects. These objects include the root component, the ports (external and internal) registered to the root Component, the components connected to the root component (either externally or internally), and the InputPorts of the connected components. Thus, the entire model can be viewed as a hierarchy of graphs. In order to traverse to lower levels of this hierarchy, it is important to ensure there are no cycles within a graph. We have devised a depth-first search algorithm to detect the presence of cycles in the hierarchical graph organization, and to deal with them using iterations of data propagation and rule evaluation that will continue until the system reaches a steady state.

6. Conclusions

With higraphs as the underlying mathematical abstraction, we have proposed a component-based model for representing the structure and connectivity of city subsystems, and simple strategies of control for how these subsystems will interact in response to extreme disturbances. To help simplify the modelling representation, component nodes for the city infrastructure are separated from component nodes responsible for controlling subsystem interactions. This infrastructure is sufficient for simulating cause-and-effect relationships that cascade across subsystems. However, manual setup of the objects of a model can be very tedious – indeed, over 150 lines of Java source code were needed to set up a simple city with one system, six nodes, and a controller. To mitigate these weaknesses, future work will focus on improvements to the rule processing portion of the model and in allowing end users to specify problem setup and scenarios at a higher level of abstraction, perhaps using a scripting language.

References

1. Burdett R. and Sudjic D., The Endless City, Phaidon Press, 2008.
2. Harel D. On Visual Formalisms. Communications of the ACM, 31:514–530, 1988.
3. Fogarty K. and Austin M.A. , Systems Modeling and Traceability Applications of the Higraph Formalism , Systems Engineering: The Journal of the International Council on Systems Engineering, Vol. 12, No. 2, Summer 2009, pp. 117-140.
4. Venn Diagram Survey, The Electronic Journal of Combinatorics, June, 2005. For details, see http://www.combinatorics.org/files/Surveys/ds5/VennSymmEJC.html (Accessed April 3, 2014).