SANDIA REPORT

SAND2015-7902
Unlimited Release
Printed September 2015

The PANTHER User Experience

Jamie L. Coram
James D. Morrow
David N. Perkins

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2015-7902
Unlimited Release
Printed September 2015

The PANTHER User Experience

Jamie L. Coram, James D. Morrow, David N. Perkins
5544, 5346, 5541
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0974

Abstract

This document describes the PANTHER R&D Application, a proof-of-concept user interface application developed under the PANTHER Grand Challenge LDRD. The purpose of the application is to explore interaction models for graph analytics, drive algorithmic improvements from an end-user point of view, and support demonstration of PANTHER technologies to potential customers. The R&D Application implements a graph-centric interaction model that exposes analysts to the algorithms contained within the GeoGraphy graph analytics library. Users define geospatial-temporal semantic graph queries by constructing search templates based on nodes, edges, and the constraints among them. Users then analyze the results of the queries using both geo-spatial and temporal visualizations. Development of this application has made user experience an explicit driver for project and algorithmic level decisions that will affect how analysts one day make use of PANTHER technologies.

ACKNOWLEDGMENTS

The PANTHER R&D Application development team would like to acknowledge the regular feedback provided throughout the development process by Kristina Czuchlewski, Randy Brost, Michelle Carroll, and Susan Stevens-Adams.

CONTENTS

FIGURES

1. INTRODUCTION

The purpose of the PANTHER R&D Application is to explore interaction models for the construction of geospatial-temporal semantic graph queries as well as visualization techniques to support analysis of query results. The R&D Application is a proof-of-concept user interface and is not intended for integration with any specific analyst workflow or deployment environment.

The R&D Application implements a graph-centric interaction model. Users define queries by constructing search templates based on nodes, edges, and the constraints among them. A query returns a set of matches that satisfy the search template's constraints. Users can analyze matches geo-spatially and temporally. Users can refine and rerun queries or save results and use them as building blocks in other queries.

Graph Analytics Workflow

The R&D Application supports a query-based workflow that begins with the analyst constructing a query. The analyst runs the query, analyzes results, and then iteratively refines and reruns the query until the question has been answered.

This query-based workflow is only one piece of a larger graph-analytics workflow that begins when new data becomes available. There are several steps that need to take place before an analyst can make use of that data in a query. These steps – which include classification or discretization of the data, conditioning the data for graph ingestion, and actually building or updating the graph – are data source specific, meaning that a different approach might be needed to support conditioning and ingestion of each new data type into the graph.

Once the data has been ingested into a graph, the analyst is likely still not ready to construct a query. Graph exploration is an important step to allow the analyst to understand the extent of the data in the graph and the types of questions the graph will allow the analyst to answer. The analyst might or might not have a concrete idea of the questions that need to be answered and the queries that need to be constructed in order to answer them. Graph exploration involves understanding the geographic and temporal extent of the data, the raw data types available for querying, and the distribution of the available data in space and time. The analyst might also be interested in exploring when data was collected and if any gaps in data collection exist that could impact query results.

The query-based workflow that the R&D Application supports is based on building, running, and refining *a single query*. However, there are likely cases where the results of multiple queries need to be aggregated in order to answer one or more higher-order questions. This workflow becomes even more complex when multiple analysts need to collaborate to answer a set of questions.

We chose to begin with the query construction aspect of the workflow because developing that portion of the user interface addressed a key PANTHER research question of how analysts should interact with GeoGraphy's graph algorithms. Development of a query capability also allowed the PANTHER team to visually demonstrate those algorithmic capabilities to potential

customers. However, analysts are unlikely to adopt this technology without appropriate tool support for the earlier stages of this larger graph analytics workflow.

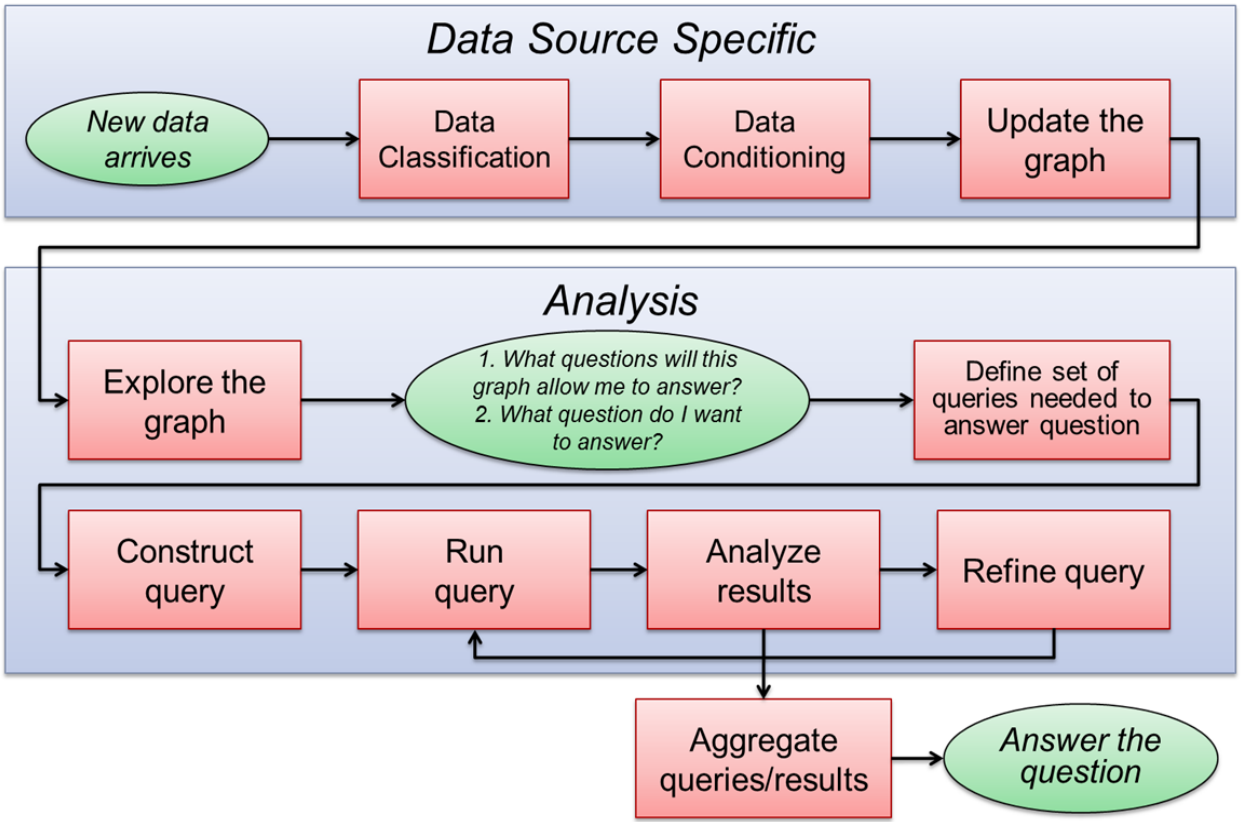

Figure 1: Graph analytics workflow

Technologies

The R&D Application is implemented using technologies that were chosen based on how well they supported the need for rapid prototype development. These technologies are not necessarily the best choices for a deployed system.

We chose to develop the majority of the user interface in Java FX, which is the modern replacement for Java Swing. Java FX provides a well-stocked toolbox of user interface components and also provides tools for rapidly developing custom visualizations. Existing development experience on the PANTHER R&D Application team also made Java FX a good choice.

We chose NASA World Wind as a GIS alternative because it is open source and integrates easily with Java. We initially investigated the use of Google Earth (GE) primarily because of its current level of adoption within potential customer communities. However, GE does not integrate well with Java, and use of the GE thick client does not adequately support the two-way interaction model that we envisioned for users (i.e., the GE thick client supports displaying information on the map, but does not allow users to input information via the map and then pass that information from GE to another component of the application). Furthermore, the future of the GE thick client is in flux, and use of the JavaScript GE API would have mandated that we develop a web-based

user interface. Lack of experience with web development on the PANTHER team made this a risky option for rapid prototype development. In the end, we chose NASA World Wind but implemented a generic map/GIS interface that will support plugging in a different GIS solution as needed in the future.

We chose Java Native Access (JNA) to support communication between the Java user interface and GeoGraphy, which is implemented in C++. JNA is a third-party library that simplifies the use of Java Native Interfaces (JNI). Although JNA provided sufficient capabilities to support prototype development, it does have shortcomings. Most notable is its limitations in passing complex objects across the interface. Performance is also a concern. If a C++ to Java interface is needed for a deployed version of the application, the JNA interface would likely need to be replaced with something more robust.

Limitations

The R&D Application is a proof-of-concept user interface, and, as such, it has the following key limitations. Note that this list is not limited to purely user interface issues. To fix several of the issues mentioned here would require improvements to GeoGraphy as well as to the user interface code. However, this list is UI-focused and is not intended to be an exhaustive list of GeoGraphy's limitations.

Graph-centric interaction model

The R&D Application supports a graph-centric interaction model, which might limit analyst adoption. Most analysts are not semantic graph experts, and they shouldn't need to be to make use of this technology. Thus, additional work is needed to design and implement more intuitive interaction models, such as query-by-example. Query-by-example is an interaction model that the PANTHER team has discussed since very early in the project. However, implementation of a query-by-example interface was out of scope for PANTHER.

Single-user model

The current implementation of both GeoGraphy and the R&D Application supports use by a single user. GeoGraphy's underlying data base solution, SQLite, is not set up to support multiple concurrent users. Multiple users can make use of saved query results if they point to the same copy of the database. However, there is currently no support for sharing query templates or for collaboration among a team of analysts.

Long query execution times for large data sets

Queries against large data sets can take a long time to execute. This is partially because the GeoGraphy code is single-threaded and partially because it is research code that has not been optimized for performance. Additional work is needed to explore multi-threading, parallelization of the algorithms, and other optimizations to the search algorithms. Additionally, the user interface could provide better feedback to the user in terms of expected execution time, progress, and intermediate results.

R&D Application only supports star-graph search

The R&D Application currently only supports the construction of star-graph query templates, which define a single hub and a number of spokes. Defining a star-graph template tells

GeoGraphy to use a particular search algorithm. GeoGraphy also supports a connected-components search algorithm which is currently not exposed through the user interface.

No support for larger graph analytics workflow
Query construction and results analysis are key parts of a larger graph analytics workflow that begins when new data becomes available. The R&D Application workflow begins with query construction, which again could limit analyst adoption of the technology. Additional work is needed to support earlier steps in this larger workflow, including examining of data quality, conditioning of data for graph ingestion, and graph construction and exploration.

Limited support for graph exploration
The purpose of the geospatial-temporal semantic graph is to make it easier for analysts to search large quantities of disparate data. However, it is unlikely that an analyst is going to be able to build a graph and immediately begin to construct queries against it. First, the analyst needs to understand the types of data available in the graph and the parameters and relationships that are available for search. This exploration step is critical for analyst adoption of this technology. Although some initial steps have been taken to support graph exploration, additional work is needed.

Rendering of visualizations is inefficient
The R&D Application is research code. As such, technologies were chosen to support rapid prototype development, and performance of the code was not a priority. Consequently, rendering of query results is not as efficient as it could be in a deployed solution. The inefficiency is partly due to the use of Java Native Access. This third-party library makes Java Native Interfaces easier to implement, which was an attractive feature for prototype development. However, that ease of implementation comes with restrictions on capability.

Query templates are not backwards compatible
Any query template created through the user interface is rendered obsolete with significant changes to the software. Thus, as Geography's capabilities and the interface between GeoGraphy and the R&D Application continue to evolve, query templates often need to be recreated from scratch. This lack of backwards compatibility is clearly an inconvenience. However, a solution is non-trivial and was not considered a high priority for our research objectives. To accommodate software testing, we created several "canned" query templates in Java code that can easily be recreated. A more robust solution would be needed for a deployed system.

Future Work
R&D Application development has had significant impact on PANTHER as a whole. For the first time, questions about user experience are driving project priorities and decision making. Still, there is additional work to be done, as illustrated by the limitations outlined above.

Although the user interface exposes many of the intricacies of GeoGraphy's algorithms in a more intuitive and accessible manner, the current interaction model is likely too graph-centric to attract a wide range of users. Use of the R&D Application still requires significant knowledge of graph analytics and the design of the algorithms. It is best suited for expert users and for troubleshooting, when more details about the graph are needed to determine what went wrong.

To determine the most appropriate interaction model for these capabilities, application developers need to work closely with experienced analysts. Successful deployment of this technology will require engagement with specific analyst communities so that we can fully understand their current tools, capabilities, and workflows and integrate our capabilities accordingly.

The R&D Application was never intended for deployment into a real-world analyst environment, and, as a result, it has limitations in terms of performance and scalability. The application currently runs in a single-user, single-threaded environment. The current database solution does not support collaboration or sharing of query templates and results. Additional work is needed to improve query execution time for large data sets, and additional tools are needed to support data conditioning, graph construction, and graph exploration prior to building the first query.

2. BACKGROUND CONCEPTS

The design of the R&D Application is grounded in the following background concepts.

Semantic Graphs

Semantic graphs are made up of nodes and the edges. Each node in the graph is assigned a semantic label, giving it meaning. A stored graph has a number of nodes derived from land-cover models and other ingested data sourced. These nodes make up the primitive data types, or raw material, in the graph. Users can then define higher-level semantic concepts based on those primitives. For example, land-cover processing might distinguish buildings from grass and pavement. However, a user might be interested in locating a particular type of building, such as a Classroom Building. To search for Classroom Buildings, the user defines a semantic concept (or role) and constrains all building nodes in the graph based on attributes like area, perimeter, eccentricity, or orientation.

Graph Search Types

A Star Graph Search requires a search template that follows a specific topology – a single hub node surrounded by zero or more spoke nodes. Each spoke is connected to the hub via an edge. The hub node is required to have a cardinality of one, while the spoke nodes can have cardinality of zero to n. Cardinality for spoke nodes is configurable by the user through the user interface. A maximum cardinality of zero can be used to specify that a particular node should NOT be present in a match.

The user interface currently only supports the construction of a Star Graph Search. A Disconnected Star Search is also inherently supported due to the addition of interrupt nodes. There is no distinction made at the user interface level that two different algorithms are in use. If the user defines an interruption in the search template, then the Disconnected Star algorithm is run; if not, then the regular Star Graph Search algorithm is run. GeoGraphy supports a third search algorithm -- the Connected Component search, which is currently not exposed through the user interface.

Query Templates

When constructing a query, the user defines a query template, which is a sub-graph of nodes and edges used to locate marched in a stored graph. Query templates are made up of the following components.

Role Nodes

A role is a node in the user's search template that has semantic meaning for that user and that query. Queries are made up of at least one node, and usually more. A node's role is a thin façade that consists of a name, short name, explanation and color. This façade only defines the appearance of the role when it is visualized. The details of a role's definition are defined in a Node Specification.

Once a role is defined, it can be reused in multiple queries. Although the appearance of the role is reused, the underling Node Specification can be defined differently for each query.

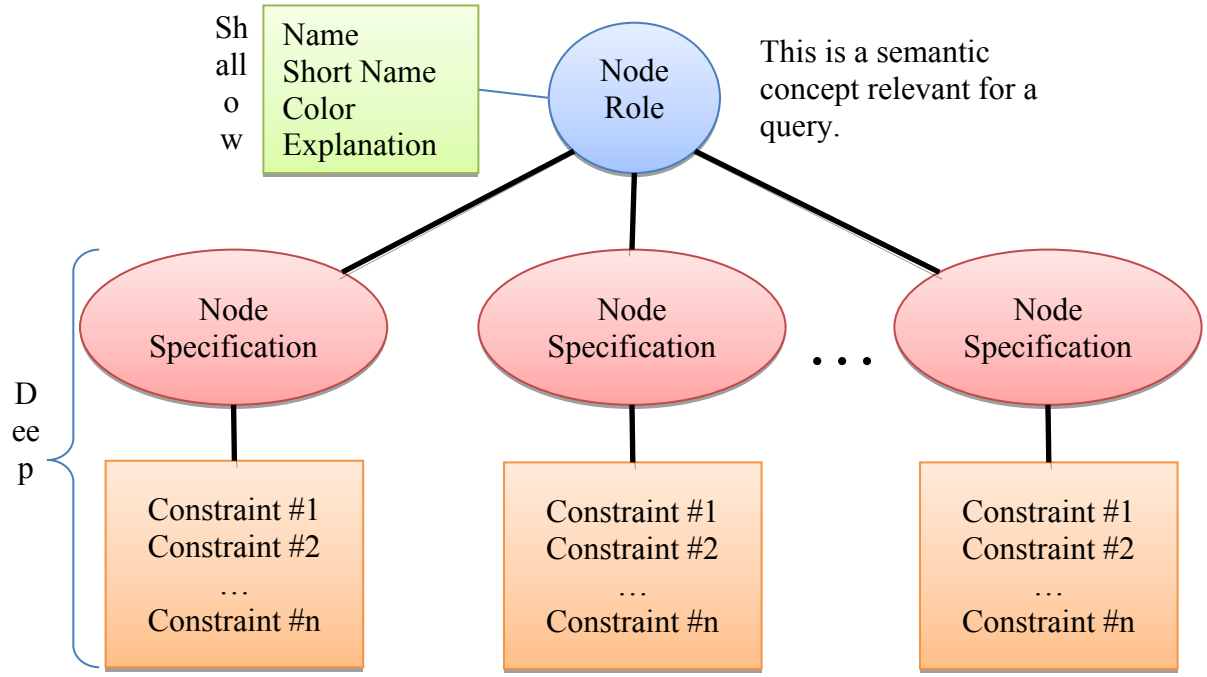

Figure 2: Role node specification

Node Specifications

A Node Specification defined the details of what it means to be a particular role – e.g., a Classroom Building or a Football Field. A Node Specification can contain a primitive type (such as a building, grass/shrub, or dirt) and it can contain a set of saved query results. A single node in a search template can also be defined by multiple Node Specifications, which are interpreted in OR fashion. As an example, consider that a parking lot could be either paved or dirt. And thus, a Parking Lot role in the search graph could be defined by either a paved primitive type or a dirt primitive type. Each is defined as a separate Node Specification and constrained individually.

Multiple Node Specifications for a role can span multiple data types. Take, for example, a Hospital role. The user could define a Node Specification based on primitive building nodes and constrain it to include only buildings of a certain size and perimeter. However, the stored graph might also include point data that defines the locations of hospitals in a given area. If, for instance, this point data is known to be incomplete, then the user might define a query that searches for hospitals using both the point and Building data available in the graph.

Each Node Specification in a search template has a number of attributes that can be used to filter the data in the graph. The set of available attributes depends on the type of Node Specification. Regions (such as buildings, grass, and paved areas) have area, perimeter, eccentricity, orientation, and other attributes suitable for two-dimensional region data. Point data can have many different attributes based on what information is present in the data files ingested into the graph.

As the user constrains Node Specifications in the search template, the application searches the database and dynamically updates a node count. This count is displayed next to each Node Specification in the user interface and provides immediate feedback to the user on how many results can be expected based on the current parameterization of the Node Specification.

Edges
Nodes in a search template are connected via edges. And, much like nodes, edges can also be constrained. Edge constraints allow the user to define relationships between two nodes in the graph based on space and time.

A common edge constrain is the maximum minimum distance between two nodes. Since nodes in the graph can be regions as well as points, there is not a single "minimum distance" value that can be defined. However, by specifying a maximum value for the minimum distance between two nodes, the user can mandate that the two nodes be "close together."

Interrupt Nodes
An interrupt node in a search template allows the user to specify that two nodes connected by an edge can optionally be interrupted by another node. For instance, depending on when and how imagery is collected, a building might have a shadow in any number of locations in the image. If the user specifies a query in which a building must be within some distance from a parking lot, the query might fail without the introduction of a shadow interrupt node.

To GeoGraphy, an interrupt node is like any other role node in the search template. At the user interface level, interrupt nodes are paired with the role node that they interrupt in order to simplify the rendering of the search template.

Query Results
Once the user is happy with a defined query template, the next step is to run they query. The user can analyze the query results through the user interface, refine and re-run the query, and optionally choose to save results back to the database. Queries can be run and results saved multiple times, which is useful if the underlying data in a stored graph is updated regularly and the user wishes to compare results of the same query over time. Each set of results is saved with a timestamp for when the query was run.

A saved set of results can also be used as a building block in a subsequent query. The user can drag a set of results from the results area back onto the query canvas. The saved results become a node specification for a role in the template. The user can then configure whether to use the entire match from the saved results of a subset of the match. For example, a High School search template might consist of a Classroom Building, a Football Field, and a Parking Lot. The user can save a set of results where each match contains all three of these roles. When using the High School saved results in a subsequent search, the user might wish to only make use of the Football Field nodes from the saved results. This is useful when additional roles are needed to define the initial matches or narrow results but the item of interest in the follow-on query is a subset of the overall match.

3. SYSTEM DESIGN AND ARCHITECTURE

The R&D Application is a Java application consisting of two key windows. The first window is an Editor Window and supports query construction, execution, and temporal analysis of results. The second window is a Map Window, which supports geo-spatial analysis of results. Communication flows from the Editor Window to the Map Window in both directions.

Figure 1 depicts the high-level architecture of the application. Blue boxes indicate Java code. Orange boxes indicate C++ code.

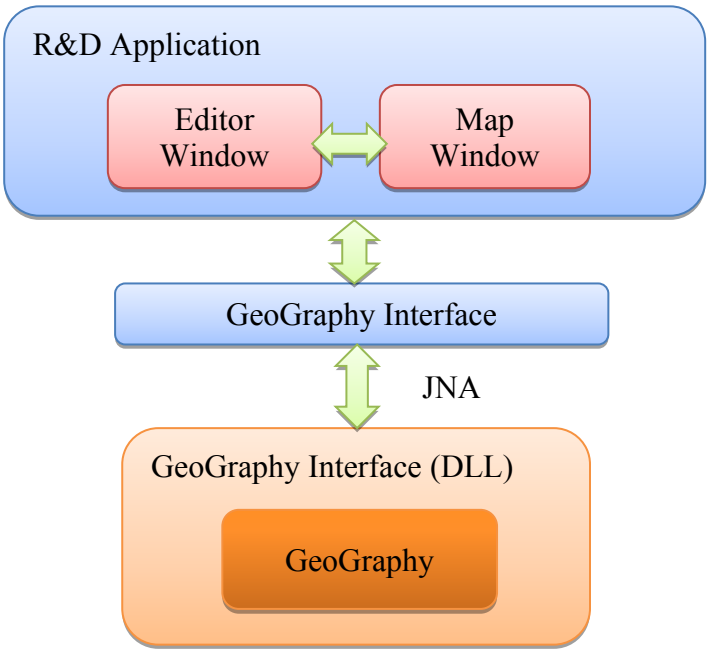

Figure 3: High-level R&D Application architecture

GeoGraphy Interface

The GeoGraphy Interface wraps GeoGraphy's graph search capabilities (written in C++), enabling two-way communication between GeoGraphy and the R&D Application. All invocations of GeoGraphy and database searches from the user interface go through the GeoGraphy Interface.

The GeoGraphy Interface exposes a subset of GeoGraphy's capabilities, which are then exposed through the user interface. To create the interface between the C++ and Java code, the GeoGraphy code base is first packaged into a DLL. The functionality wrapped in the DLL is then exposed to Java using Java Native Access (JNA).

Some explicit design decisions were made to minimize the communication with GeoGraphy and to request data over the interface on demand. For example, when a query completes, minimal information about each match is passed over the interface to the Java code. It is not until the user selects to see the results of a particular match that the detailed information about that match is requested over the interface. This "on demand" architecture is necessary because, although JNA

simplified the use of Java Native Interfaces (JNI), that simplification comes with a hit to performance. Once data is passed over the interface, it is cached on the Java side so that it does not have to be requested again.

Stored Graph Exploration

The purpose of the geospatial-temporal semantic graph is to make it easier for analysts to search large quantities of disparate data. However, before constructing a query, the analyst needs to understand the types of data available in the graph and the parameters and relationships that are available for search. This exploration step is critical for analyst adoption of this technology.

As an initial step toward this capability, the R&D Application displays for the analyst the geographic and temporal extents of data in the selected stored graph. The geographic extent is shown as a shaded rectangle on the map, and the temporal extent is shown on a timeline. This information allows the analyst to determine the "when" and "where" of the data available in the stored graph. If the analyst's questions fall outside of those geographic or temporal bounds, and the analyst knows immediately that additional data, or a different stored graph, is needed.

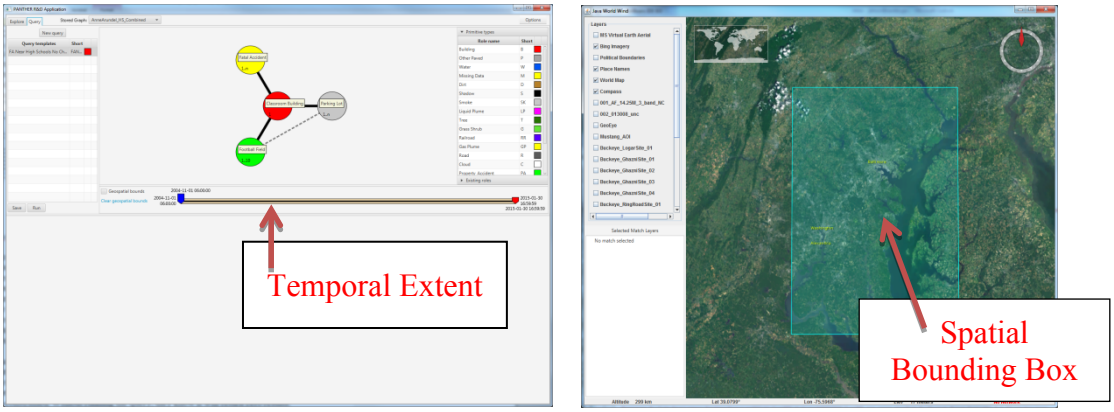

Figure 4: Spatial and Temporal Extent of Graph

A second step toward this capability allows the user to explore node parameter distributions as they are constructing queries. Query templates are very flexible, allowing users to enter exact values or min/max ranges for node parameters, such as the area, perimeter, or eccentricity of a region. However, it is not always obvious which values should be entered to locate particular features. What is the perimeter of a warehouse? What is the eccentricity of a football field? To make this process easier, analysts can view a histogram showing the distribution of parameter values for all nodes in the stored graph. Users can adjust the min and max values for the parameter and update the histogram, allowing them to narrow in on parameter value ranges where they know data exists.

A final step toward this capability allows users to view a dynamic node counter based on the set of current constraints based on a particular node in the query template. This counter gives analysts immediate feedback on the number of nodes that obey the constraints. More importantly, if the counter goes to zero, analysts know immediately when the node has been over-constrained. Without running the query, they know it will return zero results.

These initial features are only small steps toward a full graph exploration capability. Additional work is needed to determine what information is most useful and how best to overcome performance issues related to the amount of data in the stored graph.

Rendering Query Templates

Query templates can contain an arbitrary number of nodes and edges. It is important that the user interface present the template to the user in an understandable fashion. To accomplish this goal, we have simplified the rendering of nodes and edges on the query canvas.

Query Nodes

A QueryNode is a user interface object that combines a role node, its interrupt node (if present), and any of the three edges that might be defined among them. These three edges are:

1) A recursive edge from the role node to itself
2) An edge from the role node to the interrupt node
3) A recursive edge from the interrupt node to itself

A recursive edge indicates that more than one instance of node type can take place in succession.

Query Edges

A QueryEdge is a user interface object that combines the up to four edges that might exist between two QueryNodes:

1) Role A to Role B
2) Role A to Interrupt B
3) Interrupt A to Role B
4) Interrupt A to Interrupt B

In a Star Graph Search, the role node to role node edge always exists (from hub to spoke), however, the interrupt edges can only exist if the user has defined an interrupt node for that role. The QueryEdge drawn on the canvas becomes slightly thicker with each edge that is defined within it, providing an indication to the user of how many edges are hidden within that QueryEdge definition.

By default, a distance edge constraint of 10m (max) is assigned for the role to role edge. This constraint is required by GeoGraphy to run the search. The user can update the default value at any time.

See the figure below for a comparison of the typical rendering of two nodes, two interrupt nodes, and their respective internal and external edges (left side) vs. the simplified rendering of the same information as shown in the user interface (right side). A single QueryNode encompassed a role node, its interrupt node (if present), and the up to three internal edges described above. A single QueryEdge encompassed the up to four external edges that can exist between two QueryNodes. The user can click on either a QueryNode or a QueryEdge to view the details of the hidden information.

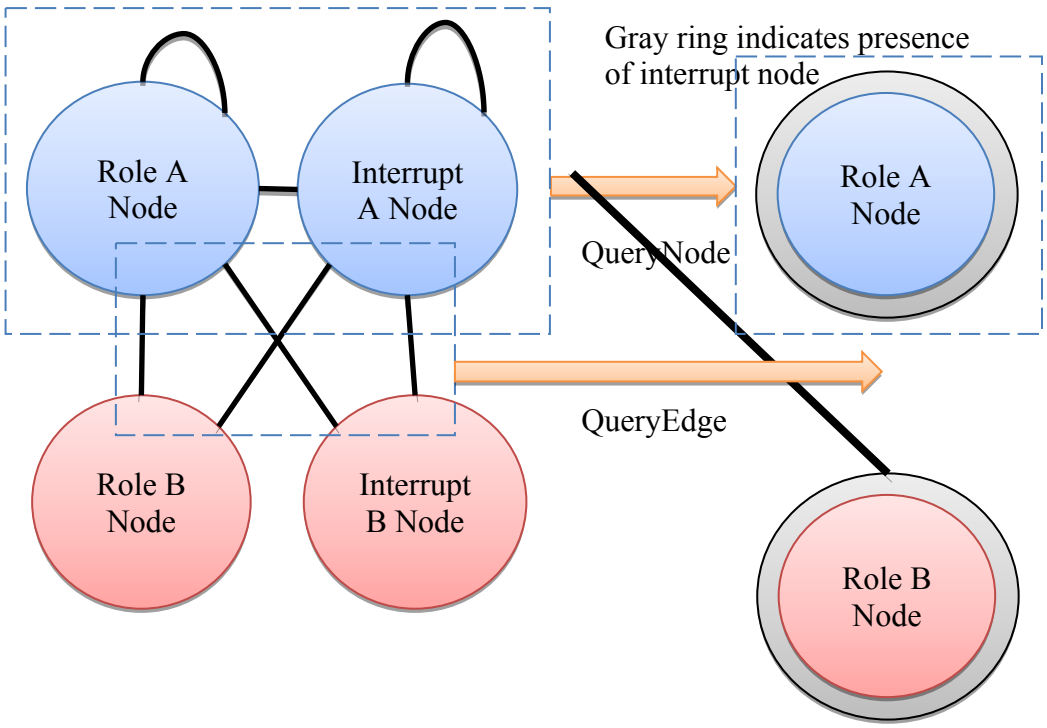

Figure 5: QueryNode and QueryEdge Rendering

Given that a search template typically contains many nodes, it is easy to imagine how cluttered the rendering of search templates might become. Collapsing the multiple edges into a single edge and the role, interrupt, and the three internal edges into a single node greatly simplifies the rendering of complex search templates.

Saving Query Templates

The user can save a search template with or without running the query. This capability allows the user to work on the definition of a complex query template over time, without losing work from session to session. When a query template is saved, it is written to a file in a directory specific to the current user.

The file is written and read using Java's Serialization capability. All classes that are involved in specifying a query must be Serializable in order for the query can be saved. A disadvantage to this approach is that any significant change to any of the Serializable classes renders the saved question unloadable and so it must be re-constructed from scratch in order to be used with the new version of the code.

Note that saving a query template in this fashion has nothing to do with GeoGraphy or its database. Saving a query template is performed at the user interface level only. This operation is different from the concept of saving a set of query results after executing a query. Saving results will write the information back to GeoGraphy's database. To keep query templates and query results in synch, we automatically save the query template any time a set of results from the template is saved to the database.

A side effect of saving query results back to the GeoGraphy database is that the associated query template is no longer editable. This is necessary to maintain consistency between the saved results and the template that created them. Otherwise, the user could not be certain if two sets of results from the same query template are comparable.

Reusing Roles across Query Templates

Even though a query template becomes read-only once results have been saved, it is important to remember that roles can be reused across multiple queries. Furthermore, the underlying node specifications can differ for each use of a particular role. This means that a Football Field node specification, for instance, can be defined completely differently from one query template to another, even though the appearance of the Football Field role (e.g., name, color, explanation) remains the same.

Once results are saved for a query template that uses the Football Field role, the Football Field role becomes read-only – meaning that the name, color, etc. cannot be changed. Likewise, the underlying node specification for the query template with saved results also becomes read-only. However, the underlying node specification can still be edited for other query templates that make use of the Football Field role and do NOT have saved results.

See Figure 3 for a summary of this behavior. There are three queries: Q1, Q2, and Q3. A single role is reused in all three of them.

Initially, none of the queries have any saved results (Figure 3, top left). Therefore, the façade information for the role (name, color explanation…) can be modified. If modified, only the shallow façade information is affected – this is the only information that is shared across all three query templates. The underlying node specification, or "deep" portion of the definition, evolves separately for each individual query template. Modifying the node specification in Q1 does not affect Q2 or Q3.

Once Q1 query results have been saved, the shallow role definition becomes read-only. However, only Q1's node specification becomes read-only; the node specifications for Q2 and Q3 remain editable (Figure 3, middle right).

When Q2 query results are saved, its node specification also becomes read-only (Figure 3, bottom left).

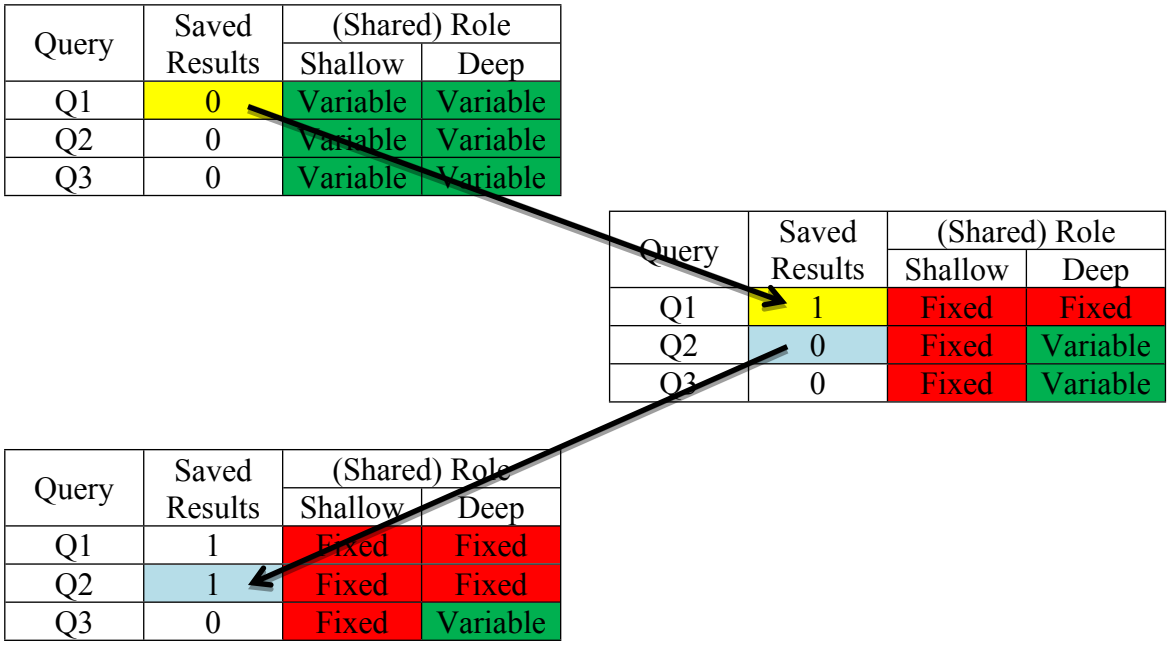

Figure 6: Shallow vs. deep copy for role definitions

One reason why a node's specification can be changed independently across multiple query templates is to eliminate unintended interference between queries. If a node's specification was shared among multiple query templates, then editing one query could break or modify the second query unintentionally. The disadvantage of this design decision is that each role node must be configured independently for each query template that makes use of it. The user interface attempts to make this easier by indicating to the user when a role is added to a query template if that role has been configured previously in another template. If so, the user can choose either to reuse the role with one of its previous configurations or to reuse the role but configure it from scratch.

Reset Database Utility

The purpose of the Reset Database Utility is a Java tool developed to support software testing. It facilitates resetting an individual user's version of the GeoGraphy database back to a known state. This is useful in two key cases:

1. After executing a series of operations via the UI that modified the data in the database (e.g., saving query results)
2. After changes to the underlying code base have rendered existing query templates obsolete

The limitation described in case 2 occurs because questions are saved via serialization of classes. When those classes change as a course of normal development, the saved questions cannot be re-read. When this happens and there are saved results associated with the question, the database must be reset to a known state (e.g. as it was originally built). Otherwise, there would be results in the database that do not have a reference question.

The GeoGraphy database is created and populated during stored graph construction, and the name of that database is normally StoredGraph_search.db. To prevent contention caused by multiple users reading from and writing to the same database, the application checks for the existence of StoredGraph_search_userid.db (where "userid" is the user ID of the person using the application). If this user-specific file is missing, then the application attempts to copy StoredGraph_search.db into it. It is this copy of the database that the user modifies as s/he saves results or creates new roles.

There are a set of "canned" query templates defined for each stored graph that are used for R&D Application testing. These canned questions are defined in the Java code, and the Reseat Database Utility writes those questions out to disk. Each user has his/her own questions directory in the stored graph area, named with their user ID. An individual user's questions for a particular stored graph are contained in this area, which prevents different users from corrupting one another's questions.

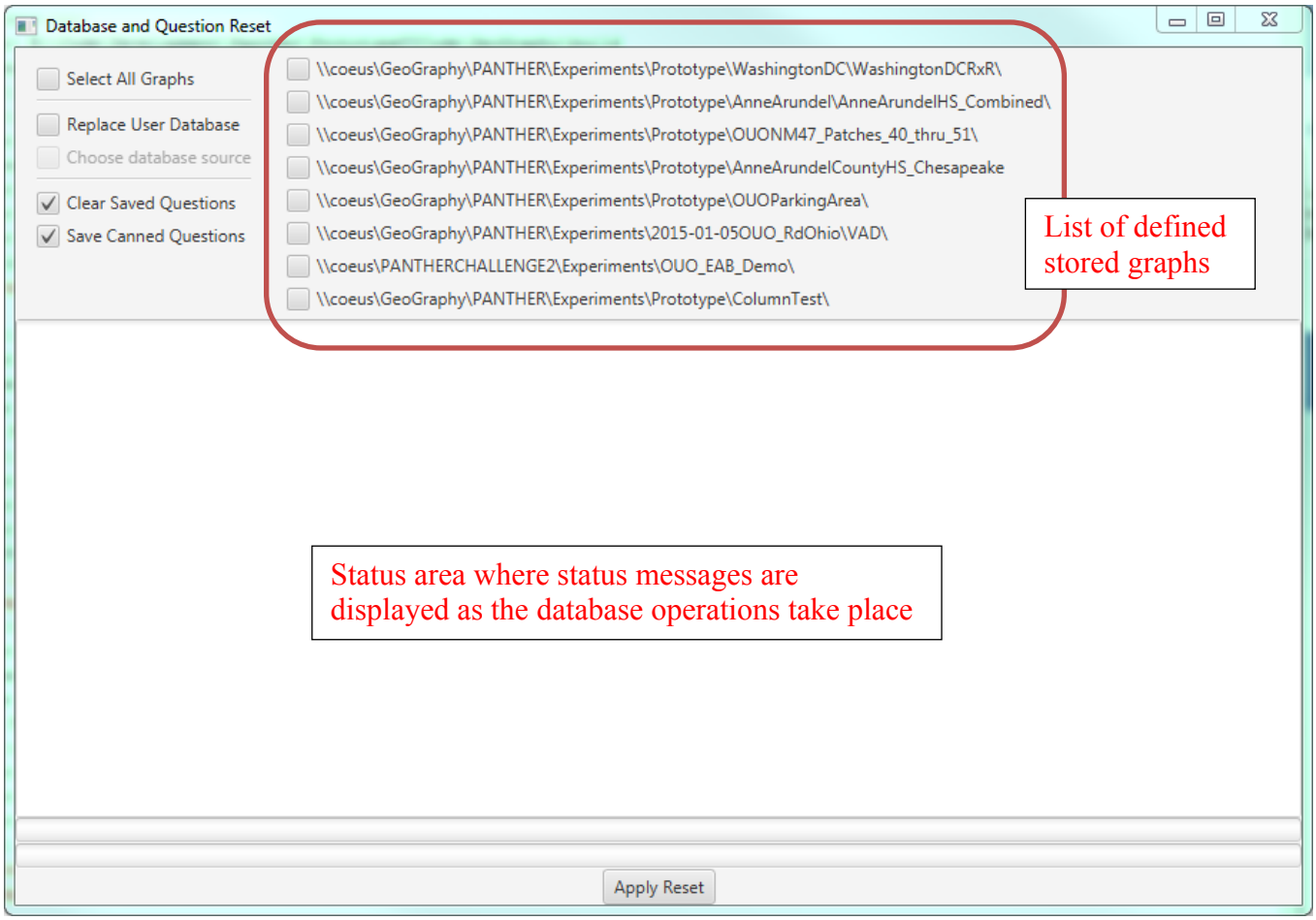

Figure 7: Reset Database Utility

The *Select All Graphs* checkbox is a convenience method to select/clear all of the stored graph checkboxes. Alternatively, the user can select a subset of the stored graphs. All other commands apply only to the selected stored graphs.

The *Replace User Database* checkbox indicates whether the user's database (StoredGraph_search_userid.db) should be overwritten. The default source file that will be copied into the user-specific database is StoredGraph_search.db.

The *Choose database source* checkbox is only enabled when the *Replace User Database* checkbox is selected. This gives the user the option to select a different source database other than StoredGraph_search.db when overwriting the user-specific database.

The *Clear Saved Questions* checkbox gives the user the option to remove saved questions in their user-specific directory. This would typically be done if the questions no longer load due to changes in the underlying code base.

The *Save Canned Questions* checkbox gives the user the option to re-initialize canned questions for the selected stored graphs

Development Environment

The R&D Application code base is version controlled via a Git repository (\\coeus\GeoGraphy\Admin\Repositories\UICode.git).

The code base is organized into the following NetBeans projects:

- CoreGUI: Project contains code to implement the graphical user interface.
- EyeTracking: Contains code to interface with the EyeWorks eye tracker to enable eye tracking analysis of using this interface. This mainly consists of connecting to the EyeWorks over the network, pinging it periodically (<= 1Hz) to get a timestamp, and writing the application time and EyeWorks time to a log file. This enables synchronizing of our logs with the eye tracking data.
- MapInterface: A generic map-independent interface meant to be able to support more than one map – for example, WorldWind and Google Earth. This was abandoned early in development.
- MapWW: The WorldWind map implementation.
- PrototypeUtilities: Common utilities needed across projects.
- ResetDatabase: A utility to manage the database – see chapter on this.

4. USER INTERFACE LAYOUT

The user interface consists of two windows: the main editor window and the map window. Within the editor window, there are two key areas:

1) A query construction area where query templates are constructed and displayed
2) A query results display and filter area

The Query Construction area is where query templates are constructed and displayed. The spatial-temporal constraint specification area, below the query canvas, allows the user to define bounding boxes in both space and time against which the query is run. This capability allows the user to select a portion of the Stored Graph to run the query against.

The Results Display and Filter area is where query results are displayed and where they can be filtered. There are several temporal views available that highlight different temporal aspects of the results set. The Filter pane on the right allows the user to apply a range of filters to the results set to narrow in on patterns and trends of interest in the data.

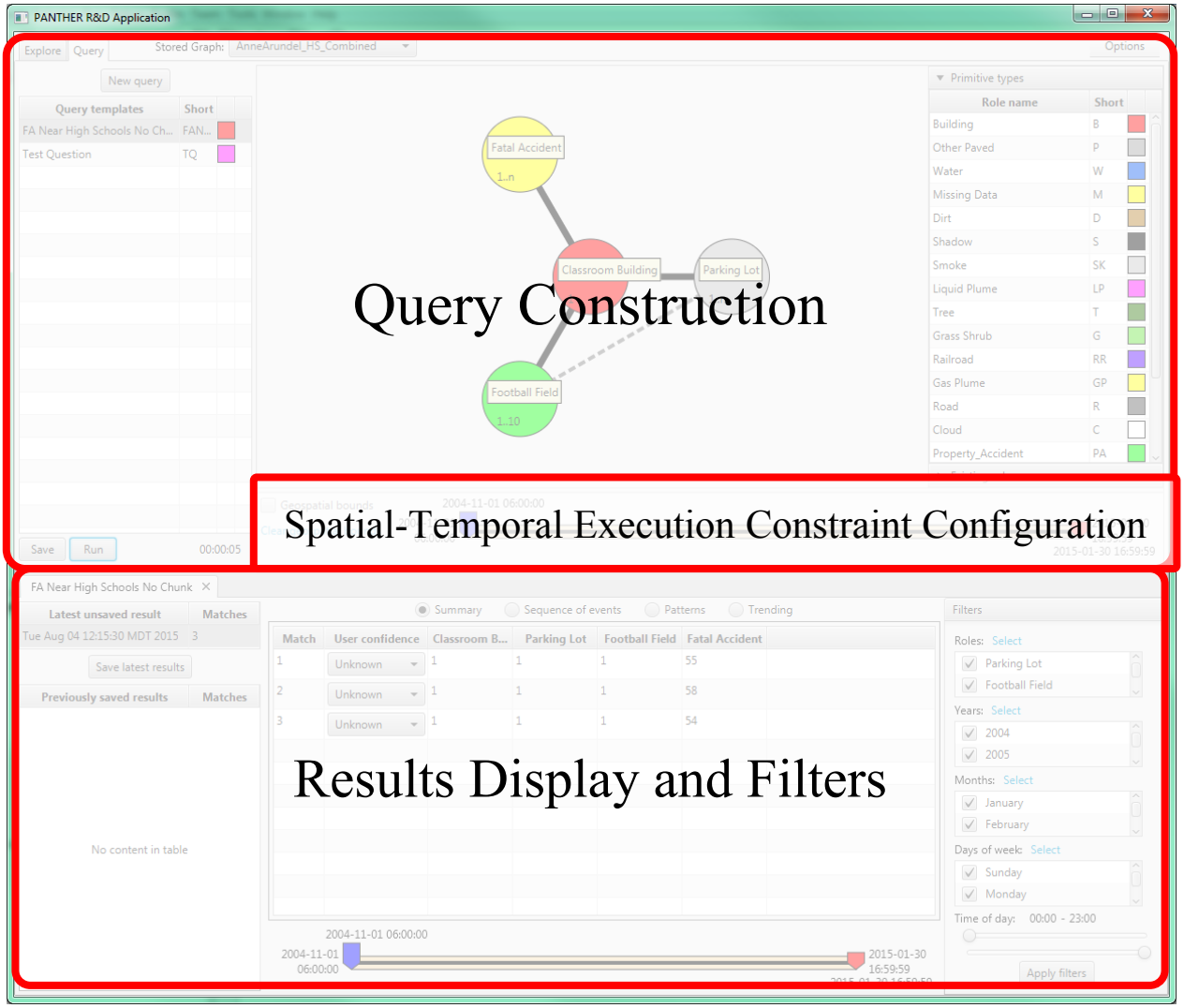

Figure 8: R&D Application editor window

Query results are also displayed geospatially in the map window. The map and query results displays are synchronized in terms of selection and filtering.

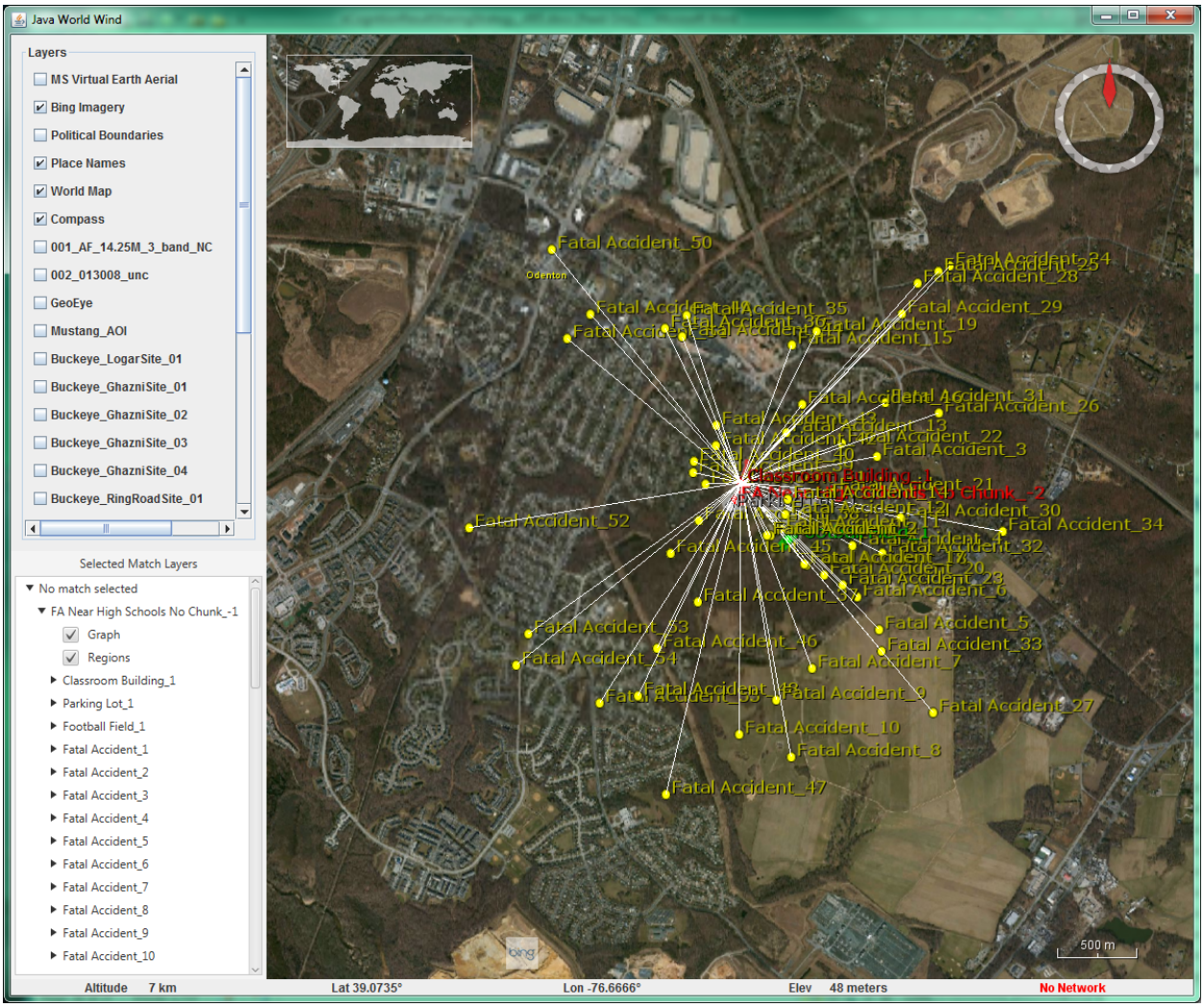

Figure 9: R&D Application map window

Constructing a Query

Users construct queries by defining a search template made up of nodes, edges, and constraints. The first step in constructing a query is to select a stored graph from the drop-down list at the top of the window. Once a stored graph is selected, saved query templates associated with that stored graph are displayed in a list on the left. The details of the selected query template are displayed on the query canvas in the middle of the window.

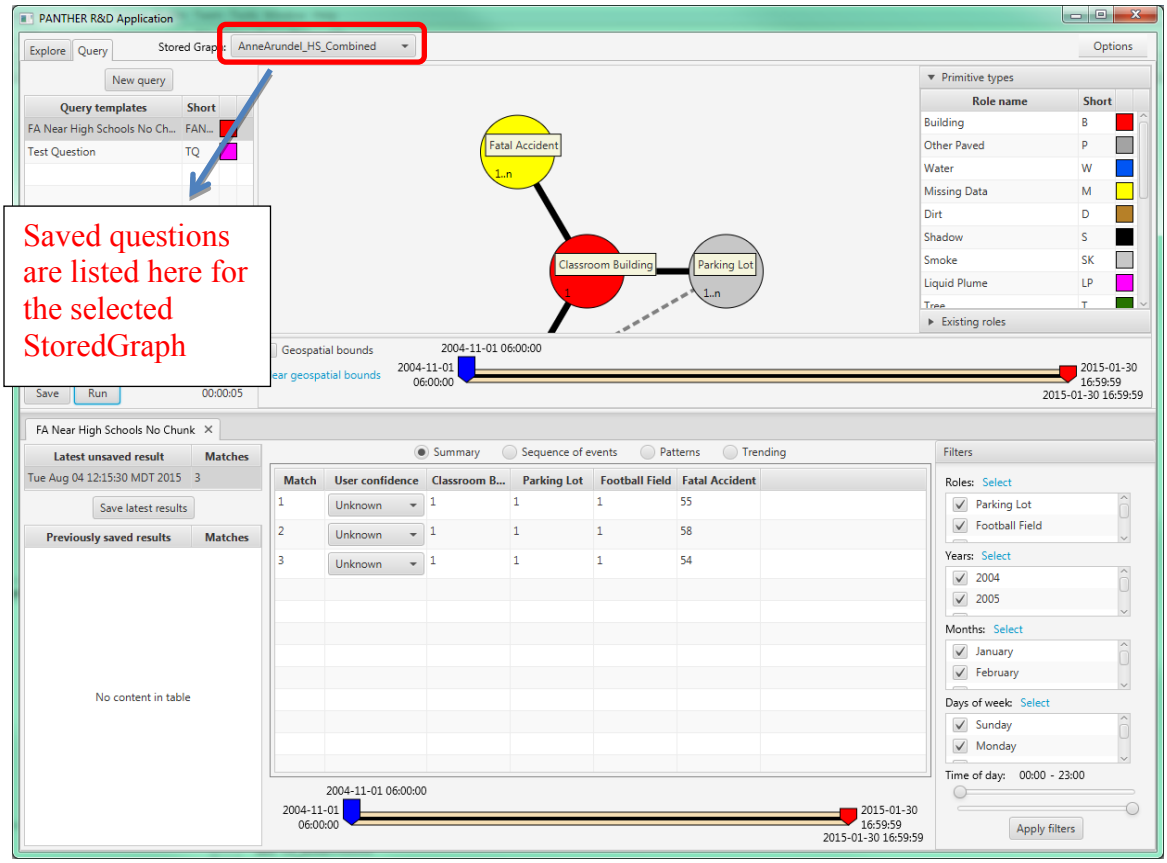

Figure 10: Selecting a Stored Graph

Each query template is assigned a top-level role that is unique to this question. In the above example, the role is "Test Question." This name also serves as the name for the saved question (which is written externally to a .qst file).

Currently, the only search graph algorithm exposed through the user interface is the Star Graph Search. Eventually, other algorithms, such as Connected Component Search, will also be supported. The two buttons displayed in the query canvas above provide a hook for future selection of the desired graph algorithm by the user. For now, press the "Star Search" button to create a new star graph search.

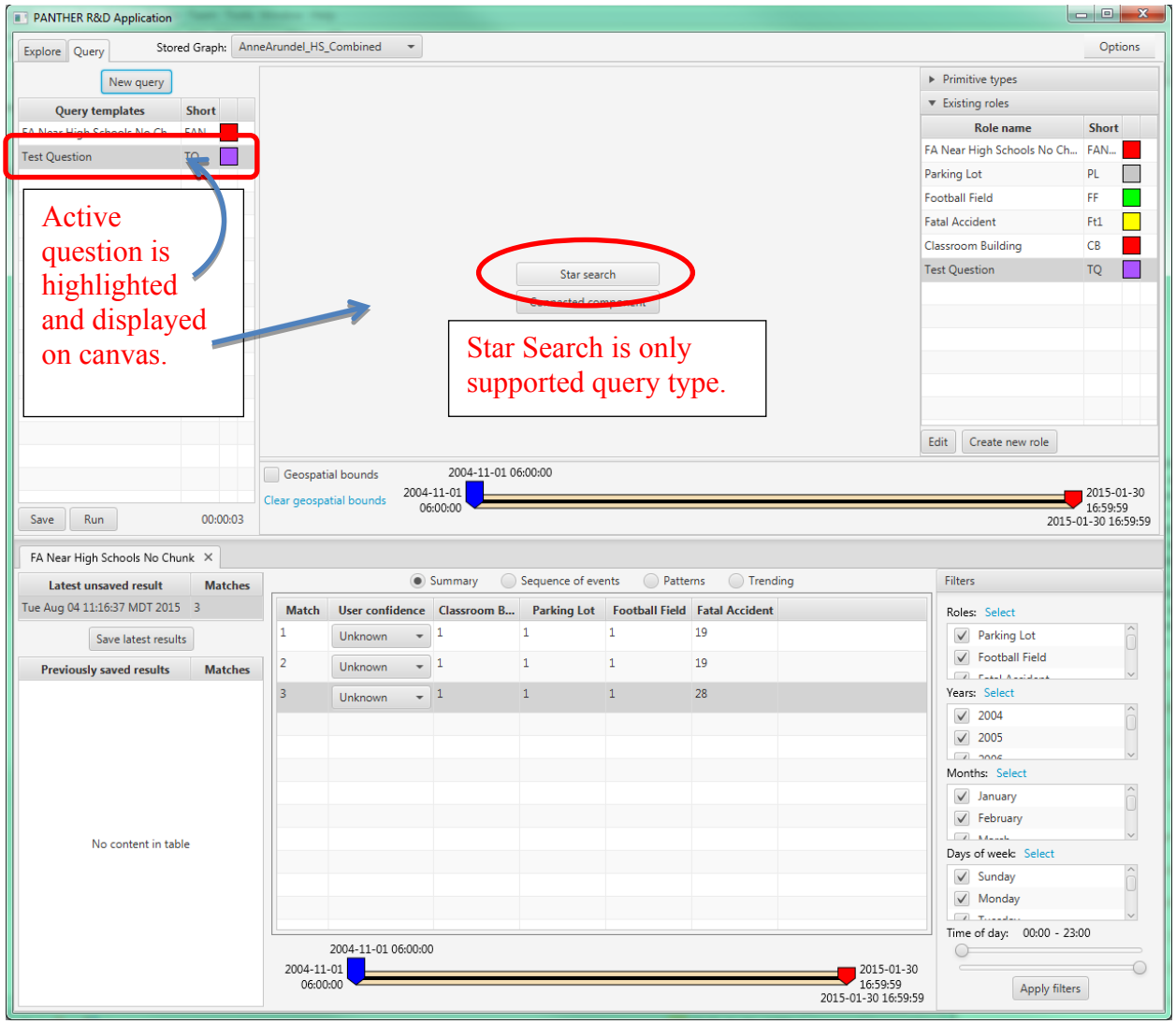

Figure 11: Query templates

A star search is made up of a single hub and a variable number of spokes. Each spoke node is automatically connected to the hub node via an edge.

To define the hub node, drag a primitive type or an existing role from the lists on the right. The list of available primitive types is determined by the data types that exist in the selected stored graph. After dragging a primitive type onto the query canvas, the user is prompted to select a role, or semantic label, for that type.

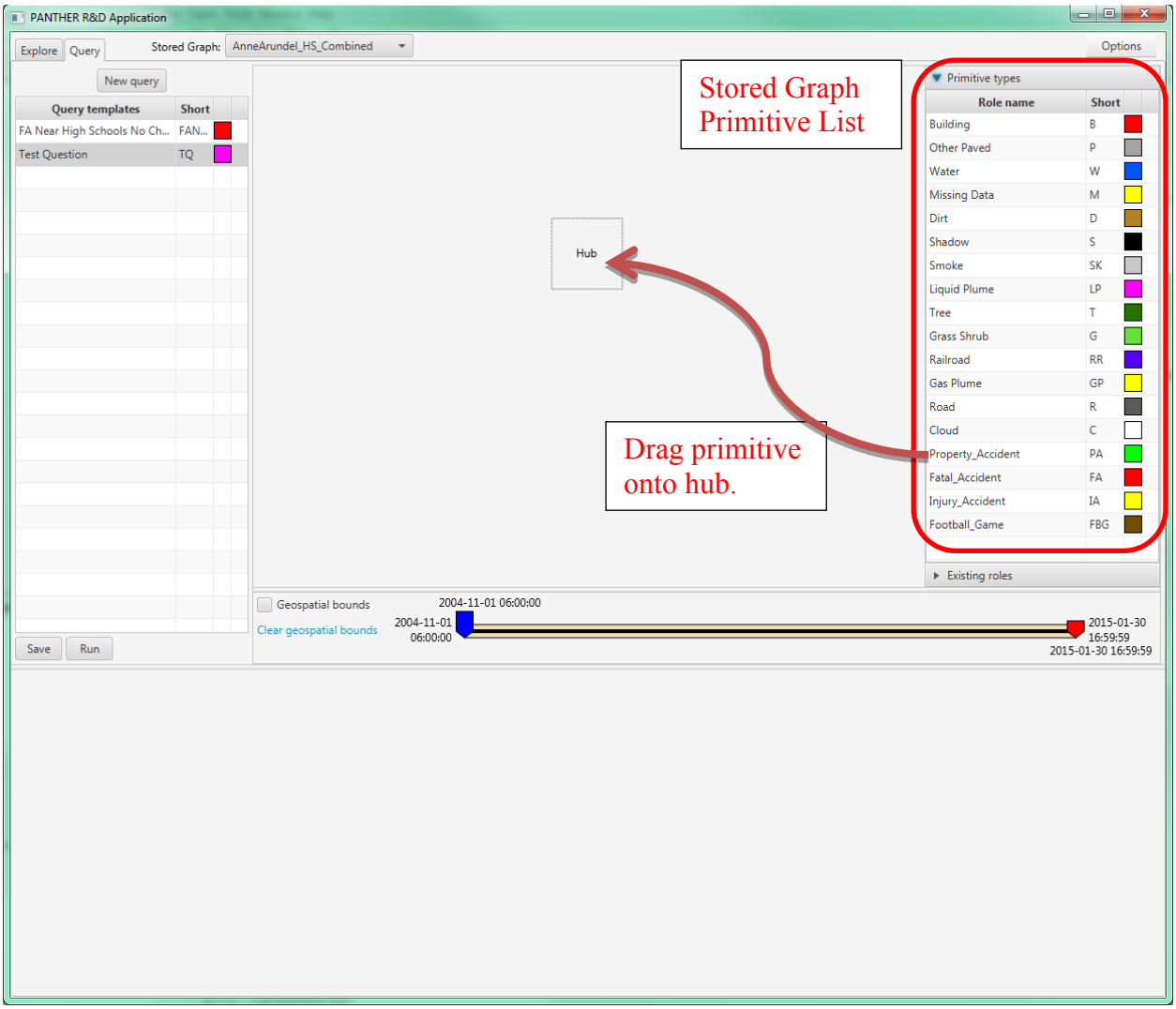

Figure 12: Adding primitive data types to a query template

Alternatively, the user can drag an existing role onto the query canvas. Existing roles are displayed in the Existing roles list below the list of primitive types. To conserve space, one list collapses when the other is expanded. Users can create new roles or edit existing roles using the buttons below the existing roles list.

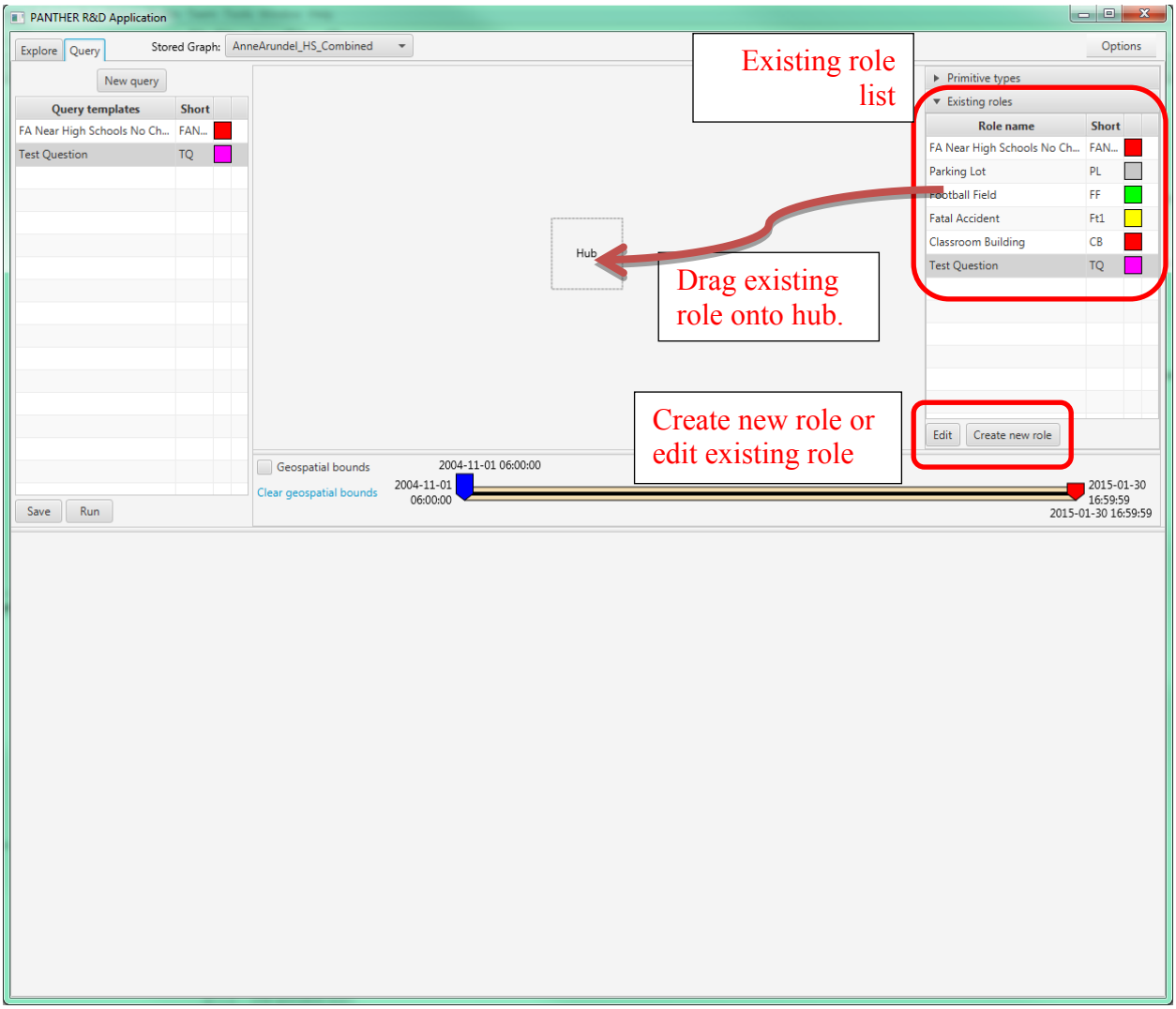

Figure 13: Adding existing roles to a query template

If the existing role has been used in other queries, the user is given the option to reuse the node specification for that role from another query.

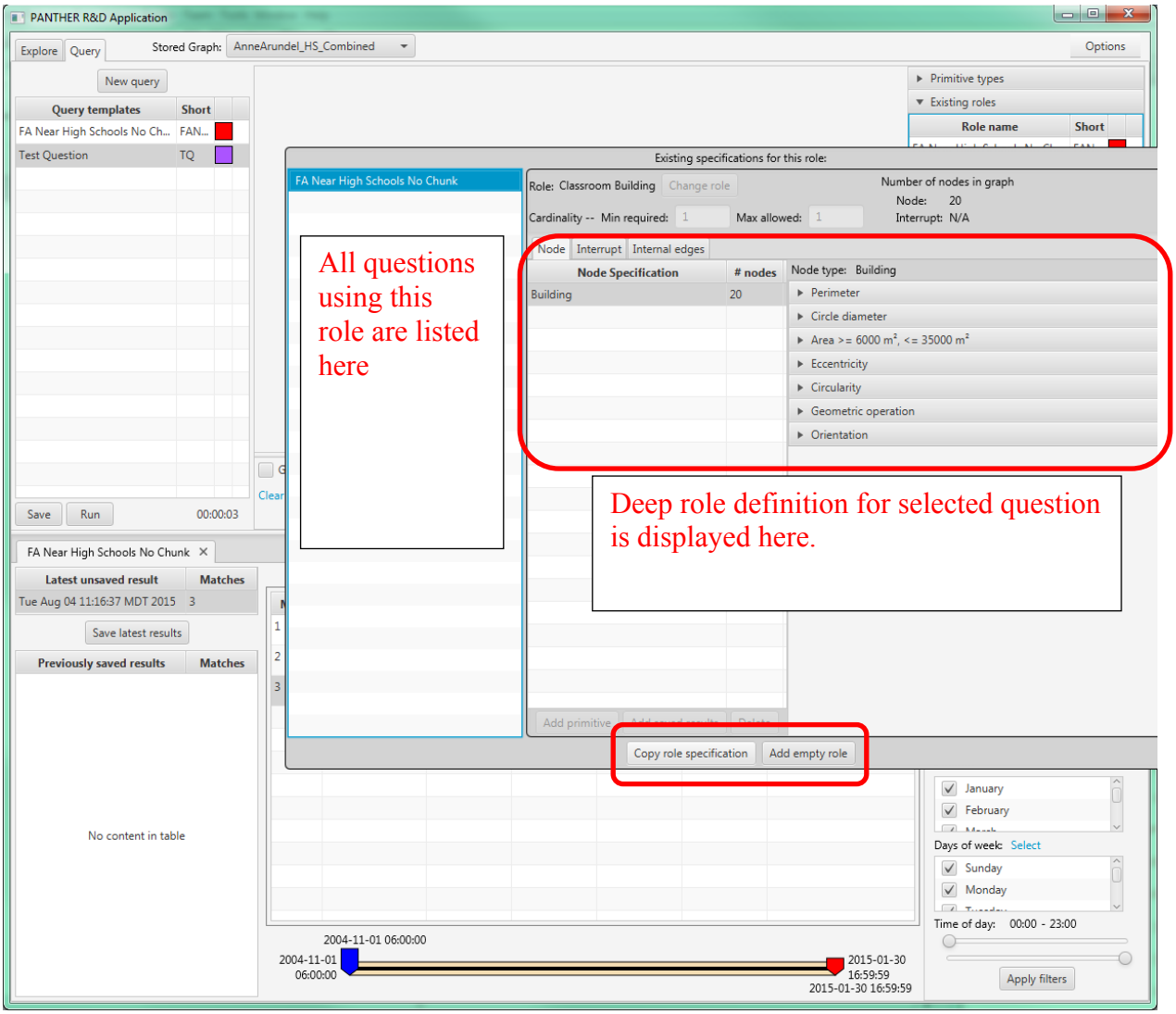

Figure 14: Reusing existing role definitions from previous queries

Or, the user can choose to use the empty role, in which case the role will be rimmed in red on the query canvas to remind the user that a node specification is needed. Clicking on a red-ringed node in the query template displays an empty list of node specifications.

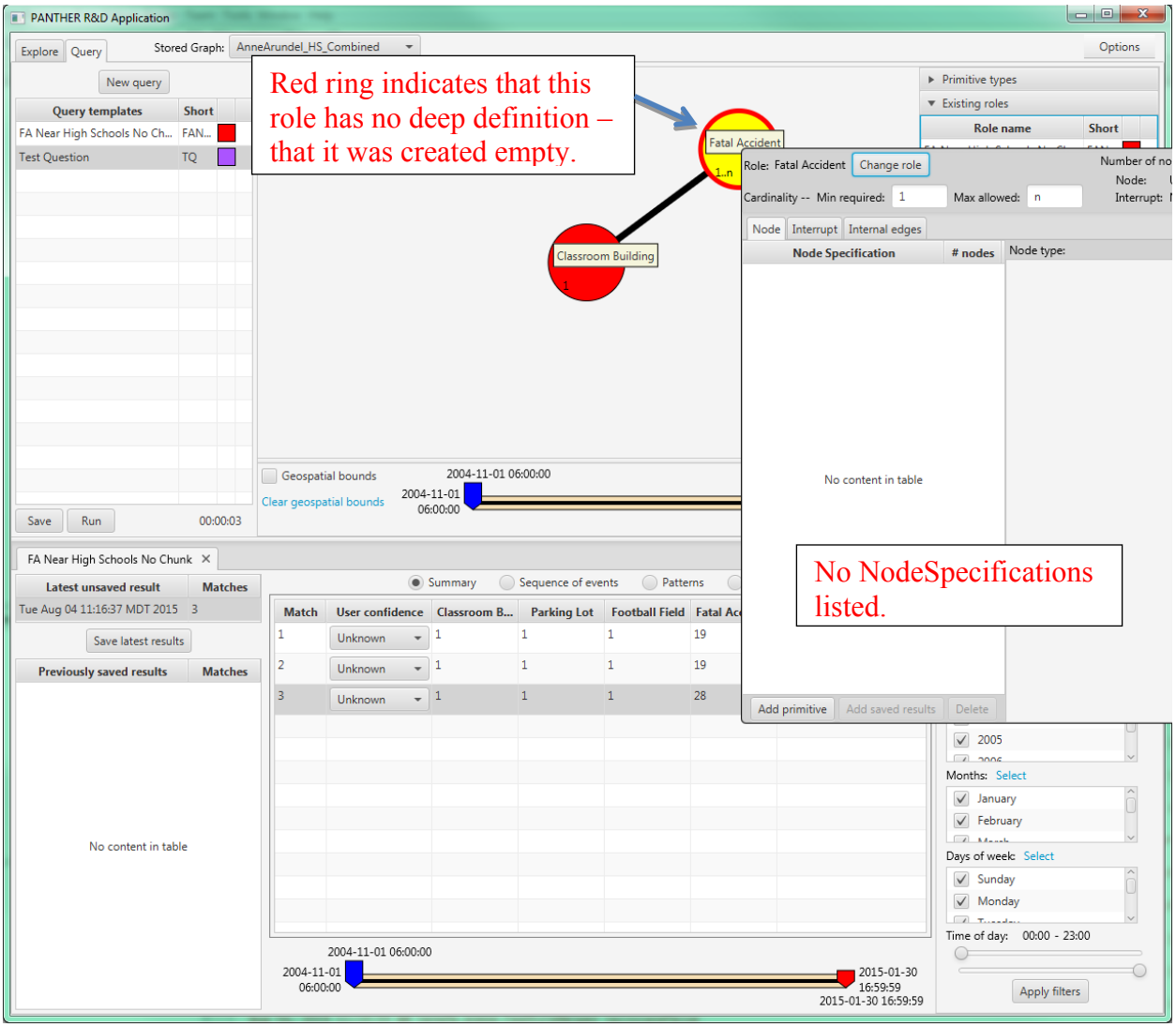

Figure 15: Indicating empty semantic roles in a query template

Users can create a new query template by pressing the New Query button above the query template list. Alternatively, users can make a copy of an existing query and then edit it. Right-click on an existing query template in the list and choose Duplicate to create a copy of the query.

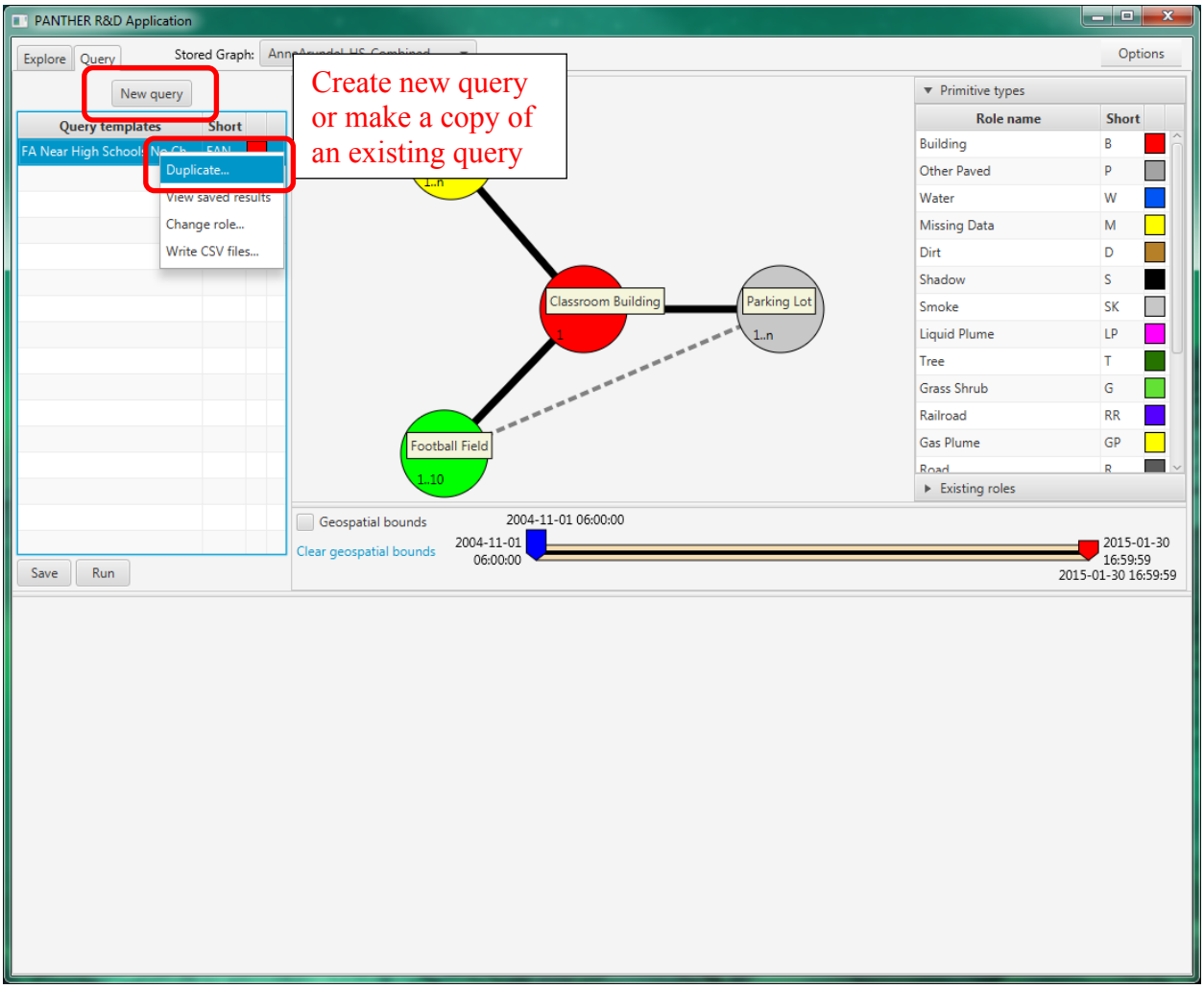

Figure 16: Creating a new query or duplicating an existing query

The user is prompted to choose a unique role for either a new query or a duplicate. Choose an existing role from the list or choose to create a new role. To create a new role, enter a unique role name, a short name (abbreviation), and a brief explanation of the role. Select a color from the palette to represent the role (e.g., green for grass).

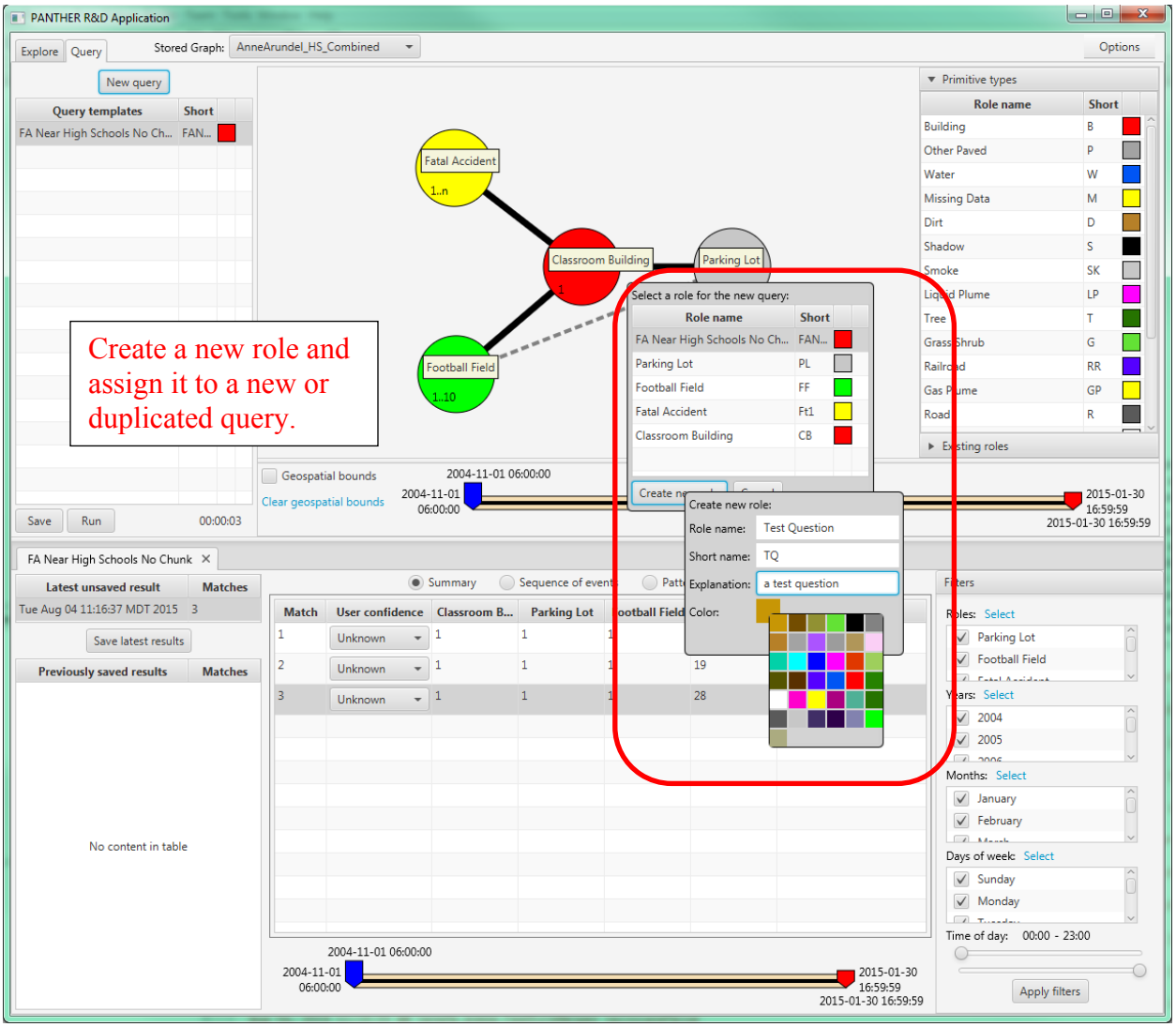

Figure 17: Creating a new role

The user can click on a node in the query template to view the details about that node's configuration. Configuration options include the node's role, cardinality requirements, and details about the node, interrupt node, and internal edges between the node and the interrupt. The Node tab (shown by default) shows the node specifications that define this node's role. In this case, the Fatal Accident role is defined by the Fatal Accident primitive type. In some cases, it makes sense to include more than one node specification in this list (e.g., a parking lot could be either paved or dirt). A node specification can be either a primitive type or a set of query results that has been saved back to the database. To add a new node specification to the list, press the Add primitive or Add saved results buttons and choose from the available list.

Select a node specification in the list to configure its parameters. Parameters for the selected node specification are shown in the accordion to the right of the list. Expand each parameter in the accordion to view/edit configuration options.

A node counter for each node specification is shown in the table. This node counter updates dynamically to reflect the number of nodes of that type in the stored graph that meet the current set of parameter constraints configured in the accordion.

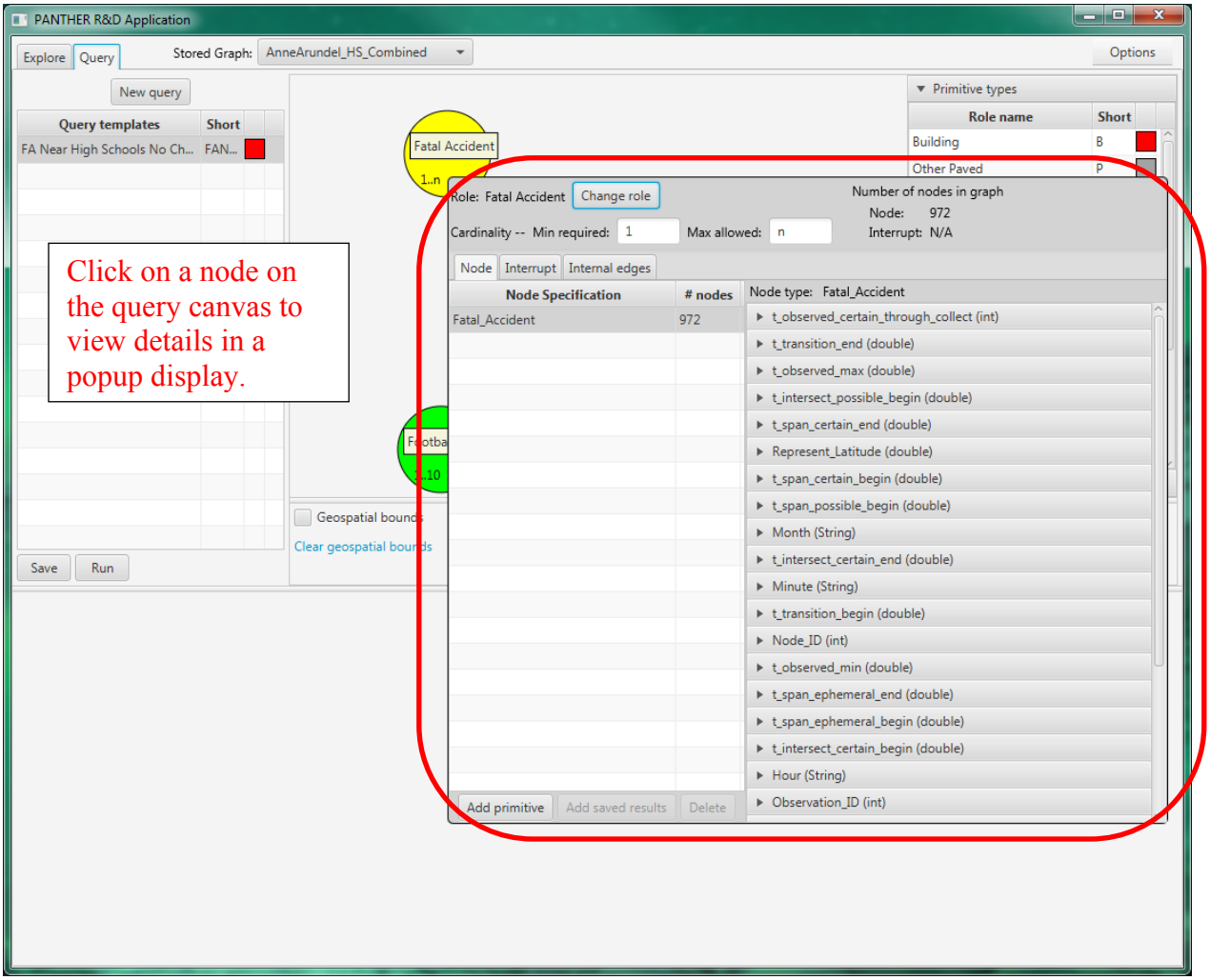

Figure 18: Viewing node specification details

After configuring a node, the user can view the node specification as SQL. This is useful for troubleshooting if a query is not running or not producing the expected results. To view the node specification, right-click on the node and select 'View specification.' The specification is display in a popup window, as shown below.

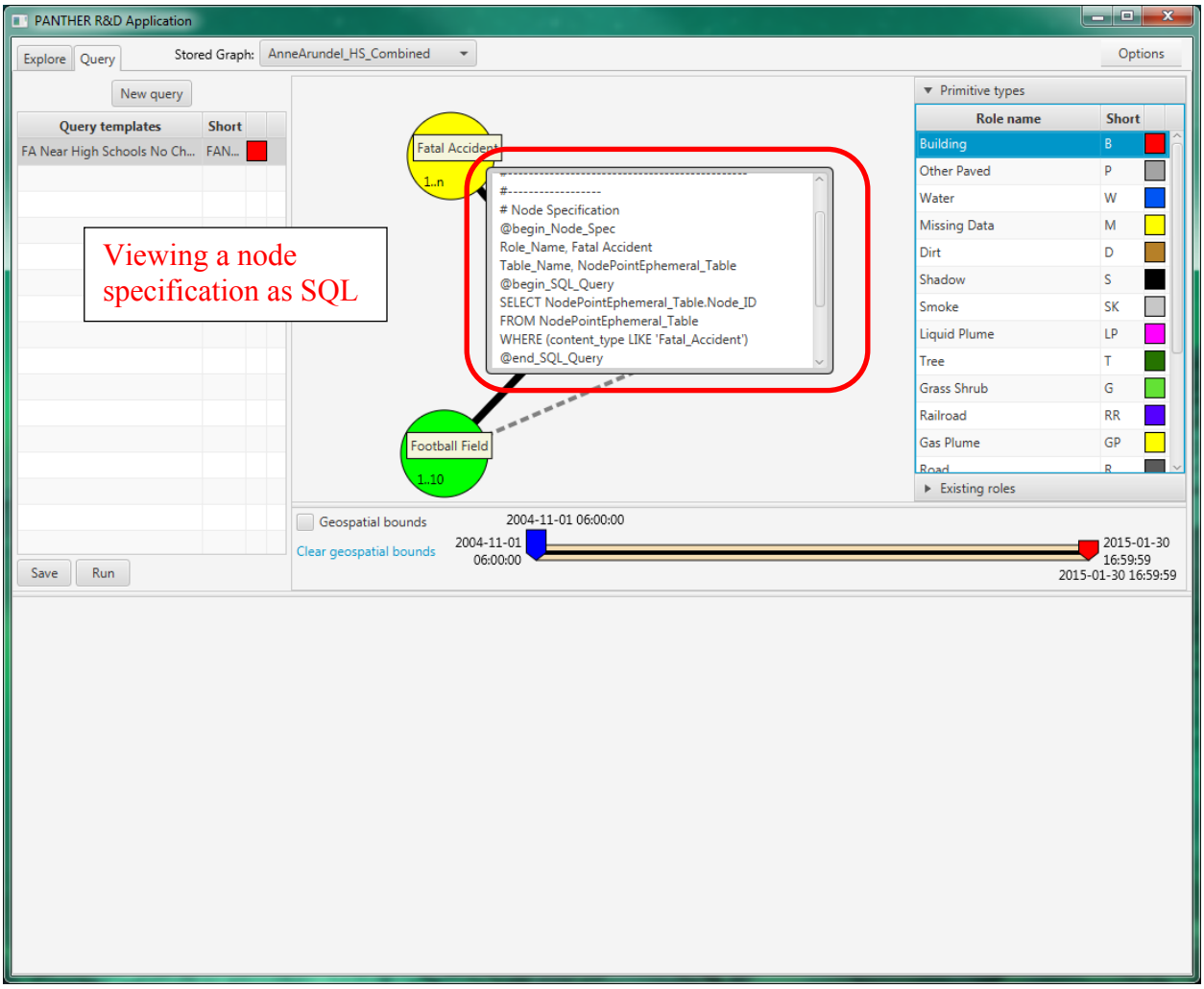

Figure 19: Viewing node specification as SQL

Users can define interrupt nodes for any node in a query template. If a node has an interrupt defined, it is displayed as a gray outline around the node in the query canvas.

To define an interrupt, click on the node to view the node specification details. Select the Interrupt tab and add items to the Interrupt node specification list. In this case, the Classroom Building node has been assigned an Interrupt of primitive type Shadow. Interrupt node specifications can be configured in the same manner as regular node specifications.

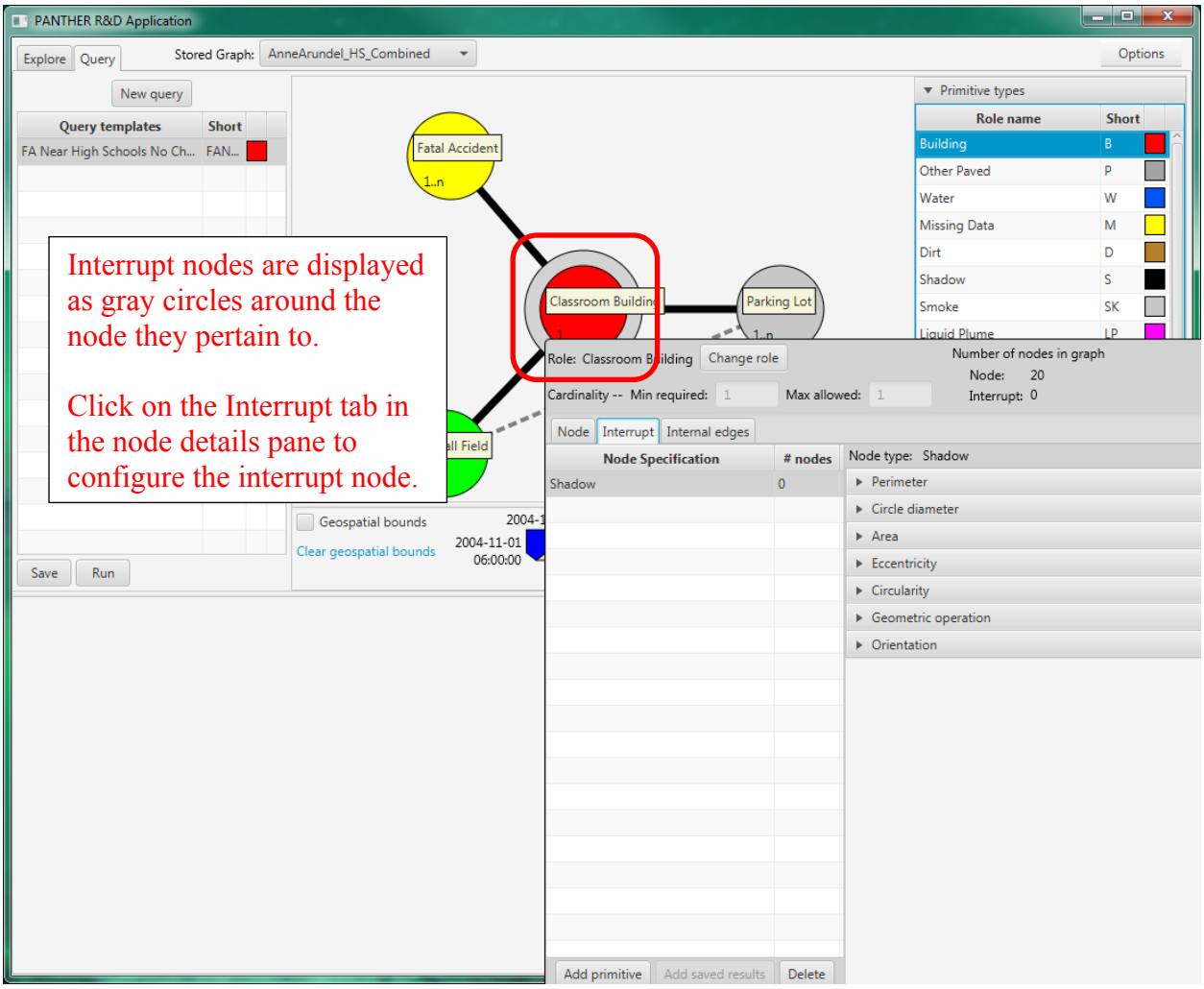

Figure 20: Configuring interrupt nodes

A single node in a query template can have up to three internal edges, which are managed via a third tab in the node specification details pane. All nodes have a node-to-node edge. If a node has an interrupt defined, then there is also a node-to-interrupt edge and an interrupt-to-interrupt edge. The user can select the checkboxes to include individual edges in the query definition. Define constraints on the selected edge by clicking on the Add constraint button and choosing a constraint from the available list. The selected constraint is added to the constraints list for the selected edge. Select a constraint in the constraints list to view/edit its minimum and maximum values.

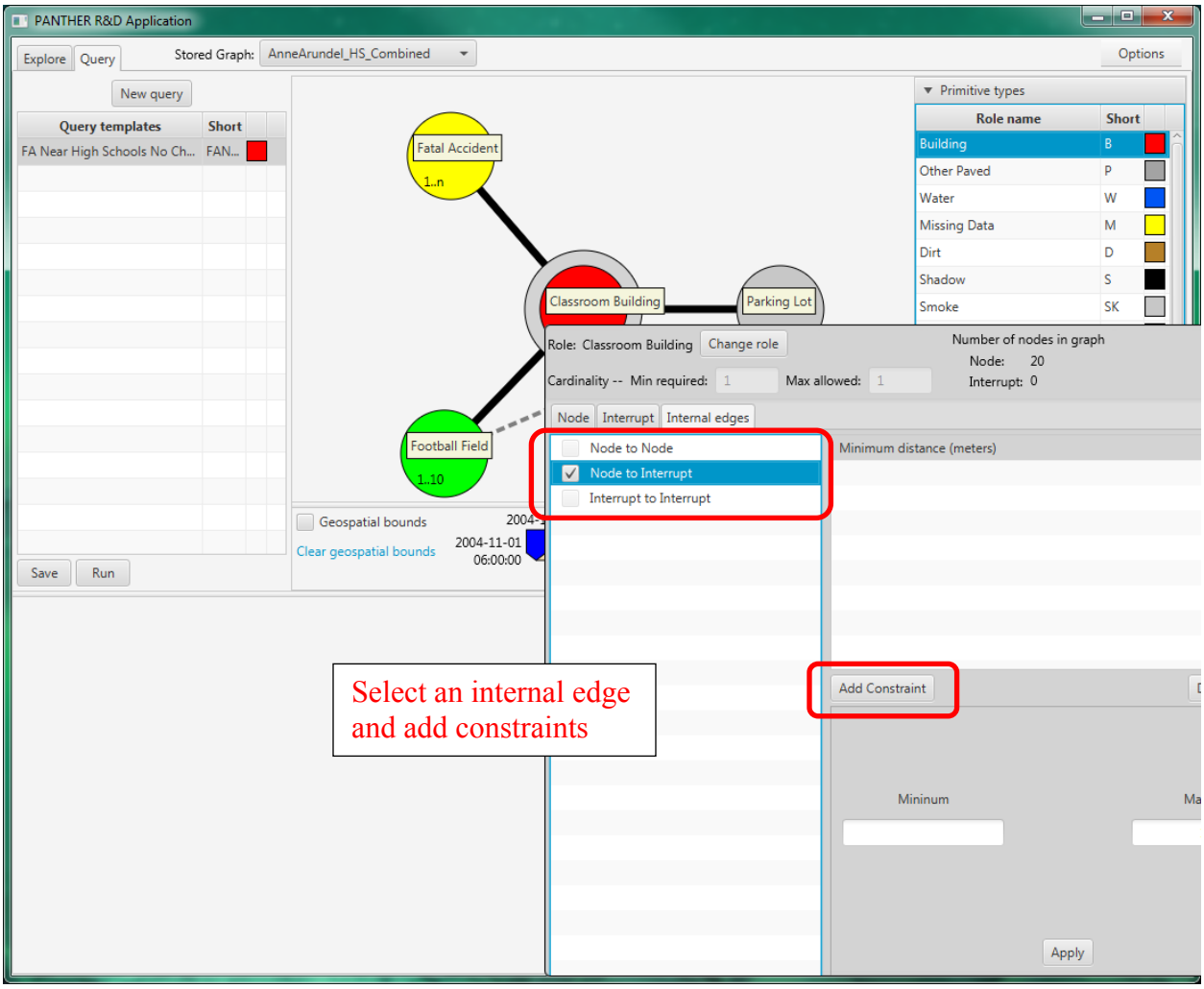

Figure 21: Configuring internal edges

Users can also constrain hub-to-spoke edges in the query template. These edges are configured in a similar manner to internal edges.

A hub-to-spoke edge can actually contain up to four individual edges, if both nodes have interrupts defined. To simplify the query canvas, a single edge is drawn between two nodes. Click on the edge to view/edit individual edges and their constraints.

Like node parameters, edge constraints can be complex and non-intuitive to define. For some edge constraints, such as the Directed Center Direction constraints shown below, a graphical depiction of the constraint makes its purpose more intuitive. In this case, the user can enter aim angle and angle span values directly into the text fields, or, alternatively, can click and drag the blue wedge to move and resize it for the desired angle and span.

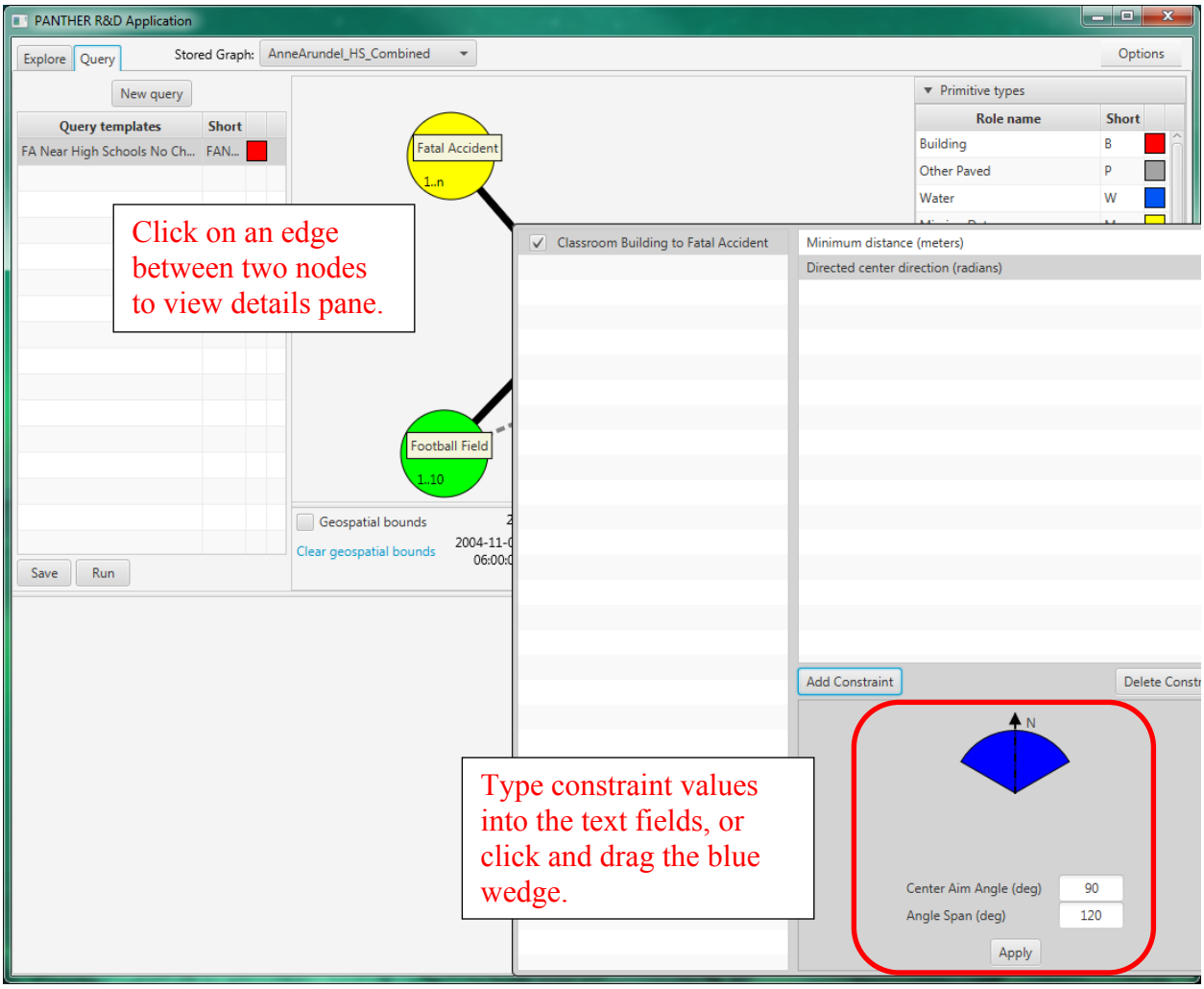

Figure 22: Configuring hub-to-spoke edges

Users can also define edges between two spokes in a query template. These edges are called Spoke-to-Spoke Constraints and are drawn as dotted lines on the query canvas. Click on a spoke-to-spoke constraint to edit the constraints in the same way as a hub-to-spoke edge. Spoke-to-spoke constraints are drawn differently on the query canvas to indicate that they behave slightly differently than hub-to-spoke edges. Spoke-to-spoke constraints are post-processed after a potential match is found for the rest of the query template.

To add a spoke-to-spoke constraint, right-click on one of the nodes and select the 'Add spoke-to-spoke constraint' option in the menu. Then, click on the second node.

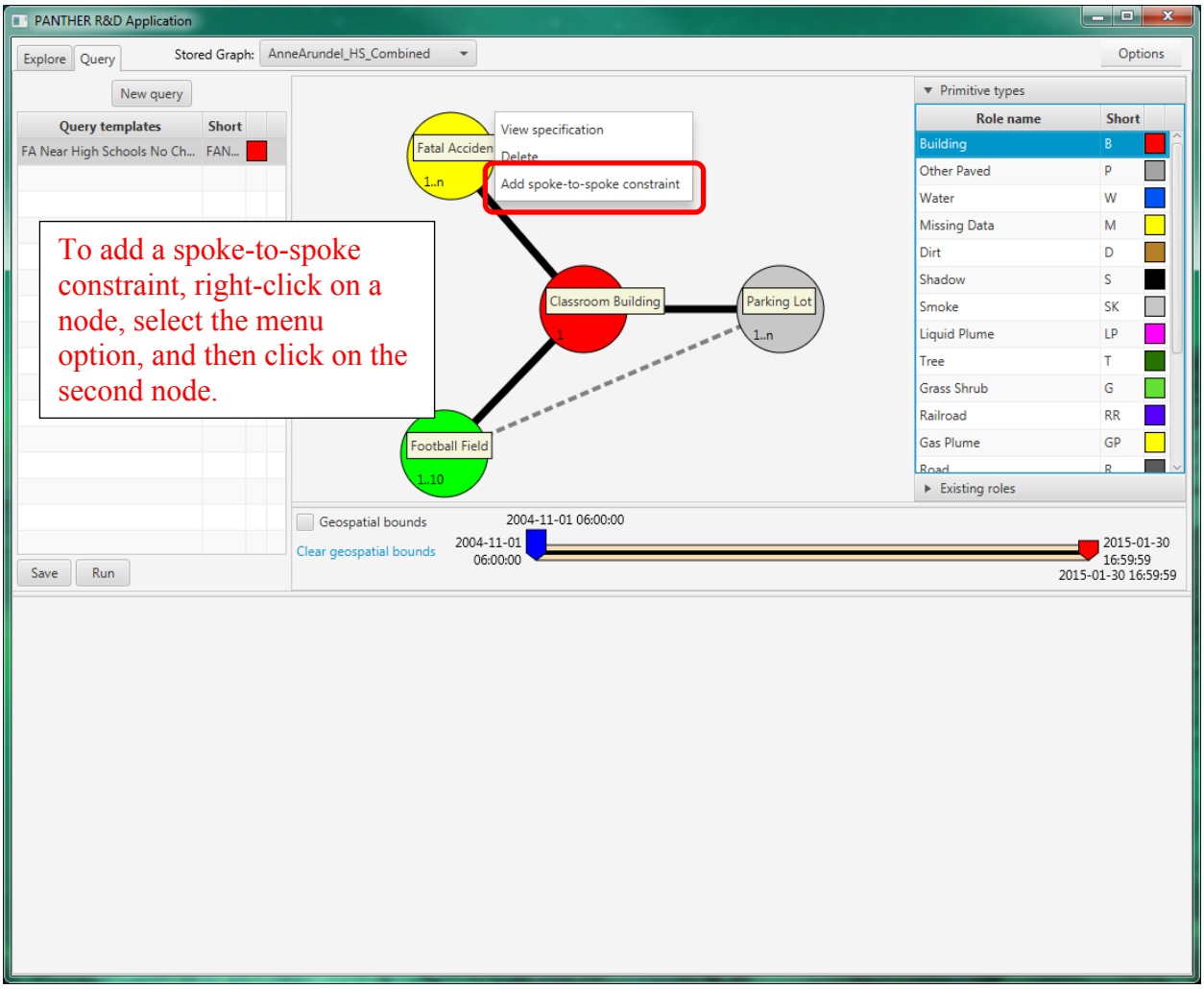

Figure 23: Configuring spoke-to-spoke constraints

Running a Query

Once the user is satisfied with the query template, it is time to run the query. The user can optionally choose to bound the query temporally and/or geospatially before running the query. Temporal and geospatial bounds are not stored with the query template, but are, instead, optional filters that can be applied as needed.

To set temporal bounds for a query, click and drag the blue and red tabs in the timeline below the query canvas. The date/time labels on the left and right indicate the full temporal extent of the stored graph. The date/time labels above/below the tabs indicate the temporal bounds for the query.

To set geospatial bounds for a query, select the Geospatial bounds checkbox below the query canvas and then click and drag on the map to draw a bounding box. Click 'Clear geospatial bounds' to remove the bounds from the query.

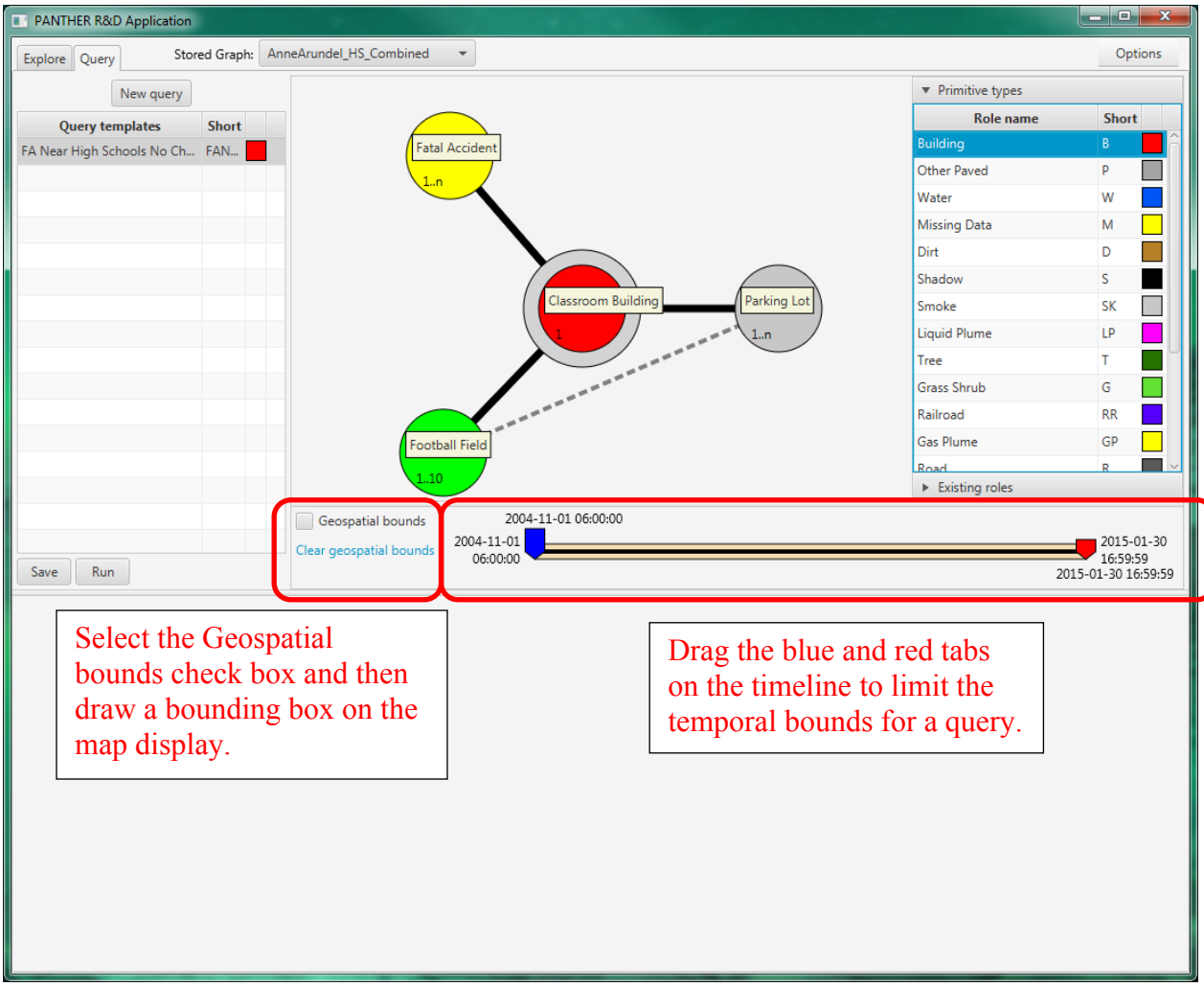

Figure 24: Configuring geospatial and temporal bounds

Press the 'Run' button to run the query. A progress bar and timer are displayed next to the Run button while the query is running.

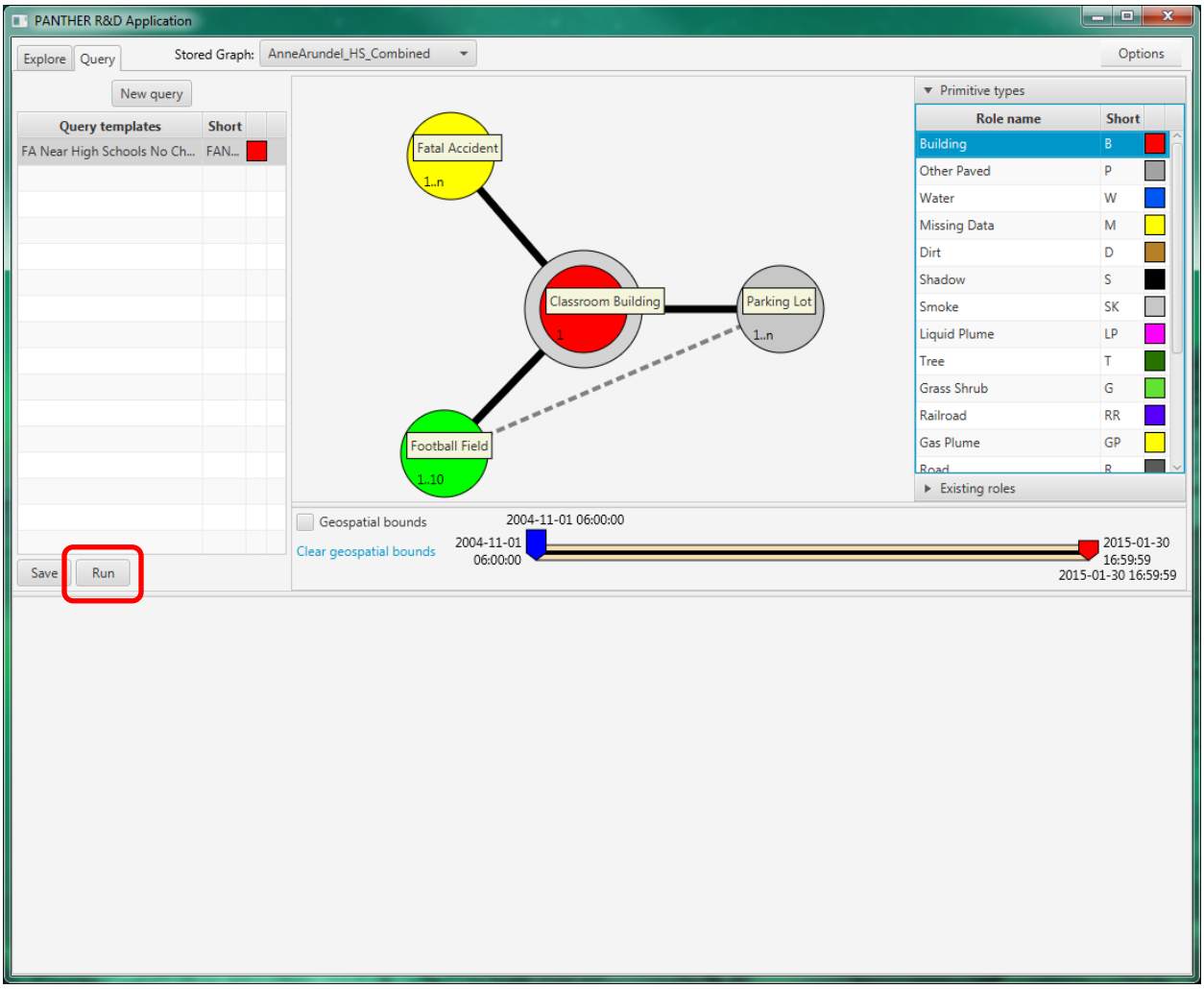

Figure 25: Running a query

Analyzing Query Results

When the query is complete, the results are displayed in the Results and Filters area below the query construction pane. For each query that is run, the results are populated in a separate tab. Click and drag a tab outside of the main window to "tear it off." Once torn off, the results window can be resized to make use of more screen space to analyze the results.

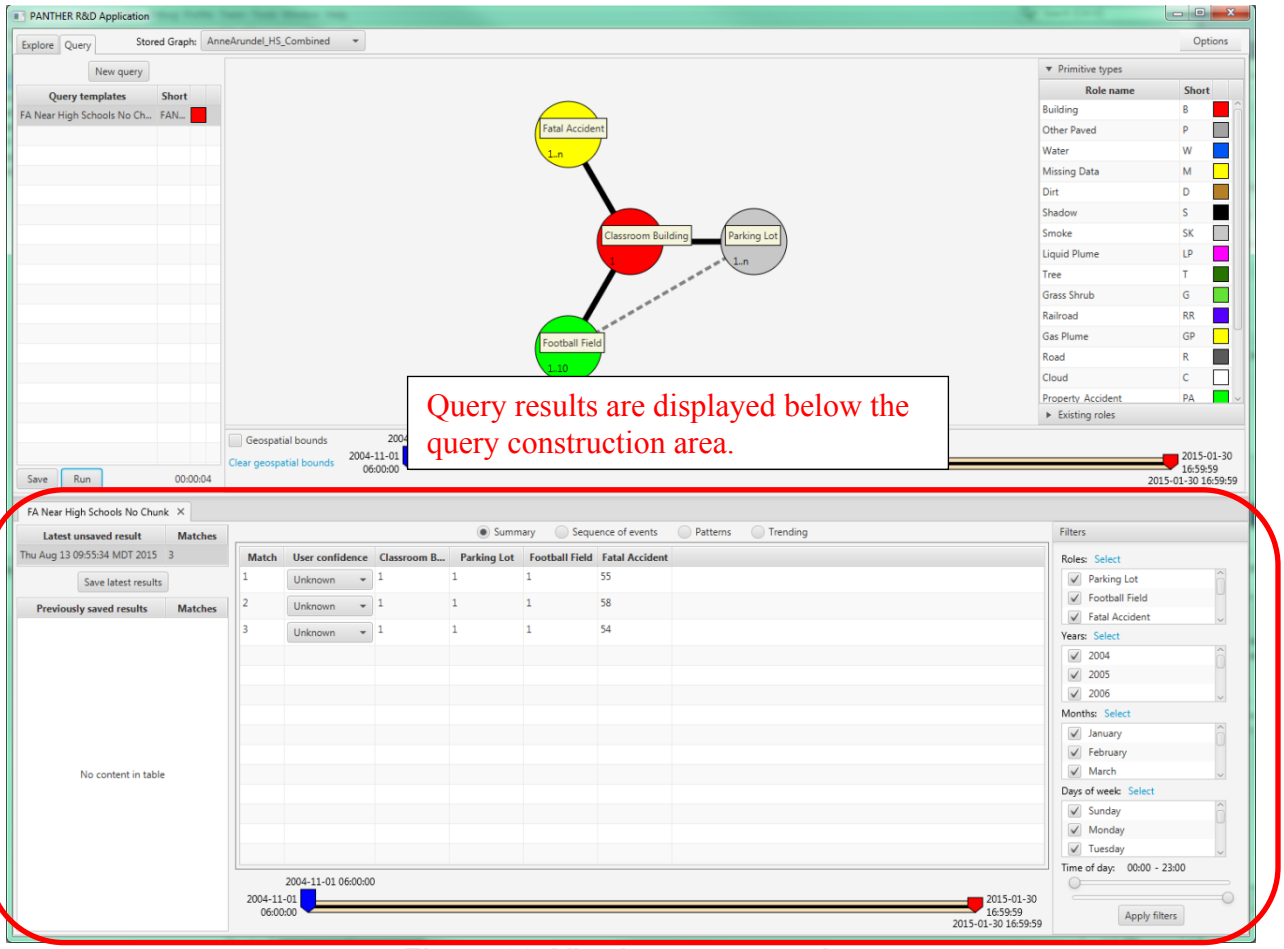

Figure 26: Viewing query results

The timestamp for the latest run of this query, plus the match count, are displayed in the 'Latest unsaved result' area in the top left. If the results of any previous runs on this query have been saved, those timestamps and match counts are listed in the 'Previously saved results' table.

To save the latest results, press the 'Save latest results' button. Once results have been saved, they can be reused in a subsequent query. Drag and drop the saved result from the table back onto the query canvas to create a node that represents those saved results.

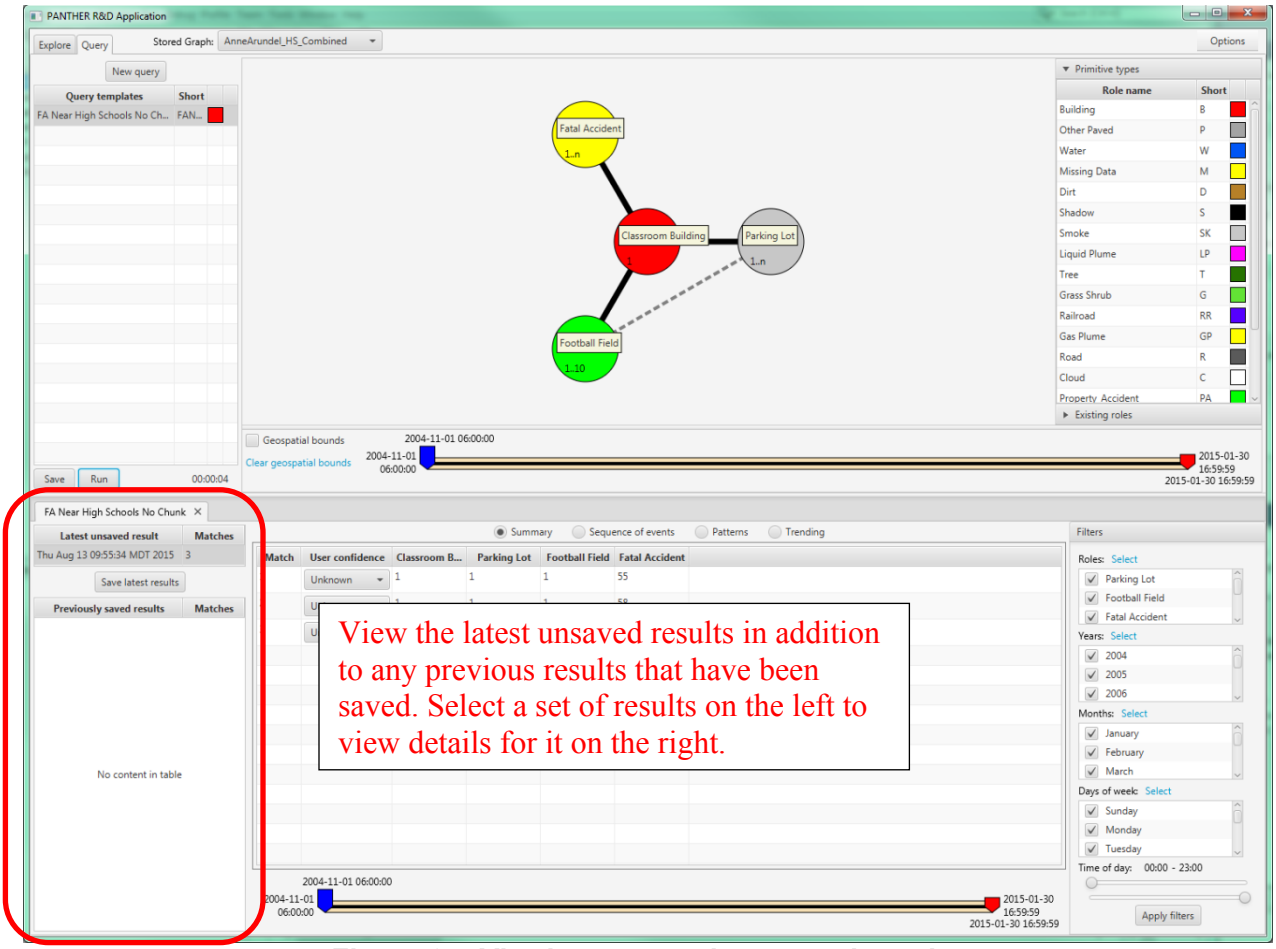

Figure 27: Viewing unsaved vs. saved results

Select the latest unsaved results set or any of the saved results set to view details on the right portion of the display. Several different temporal views are provided to help analyze the results and isolate patterns of trends of interest.

By default, a Summary table of the results is shown. Each row in the table represents on match to the query template. Each match is given a unique identifier – a negative integer for unsaved results and a positive integer for saved results.

The User Confidence column in the table allows the user to set a score for how confident they are that each match is an example of what they query template represents. By default, the User Confidence score is set to Unknown for each match. To change the User Confidence score, select a new value from the drop-down list. Values range from Very Low to Very High confidence. If a set of results is saved, the User Confidence scores are saved with them. Once the results have been saved, the User Confidence scores are no longer editable.

The remaining columns in the table represent node counts for each role that was defined in the query template. These counts give the user an initial indication of the types of matches that were found.

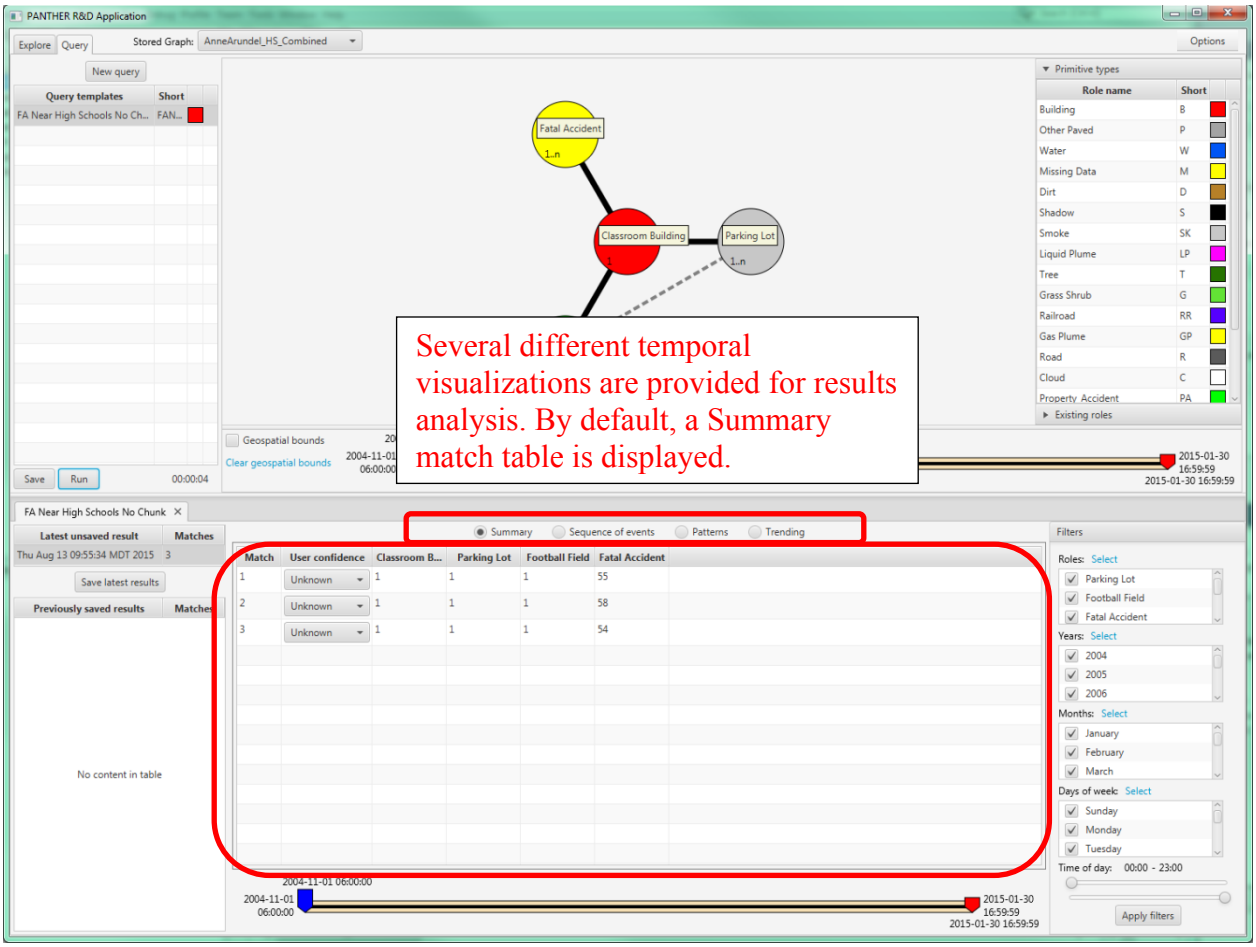

Figure 28: Results summary view

Results are also displayed in the map window. The temporal visualizations and the map display are synchronized to help the user simultaneously analyze results both temporally and geospatially. By default, a single placemarker is added to the map for each match in the selected results set.

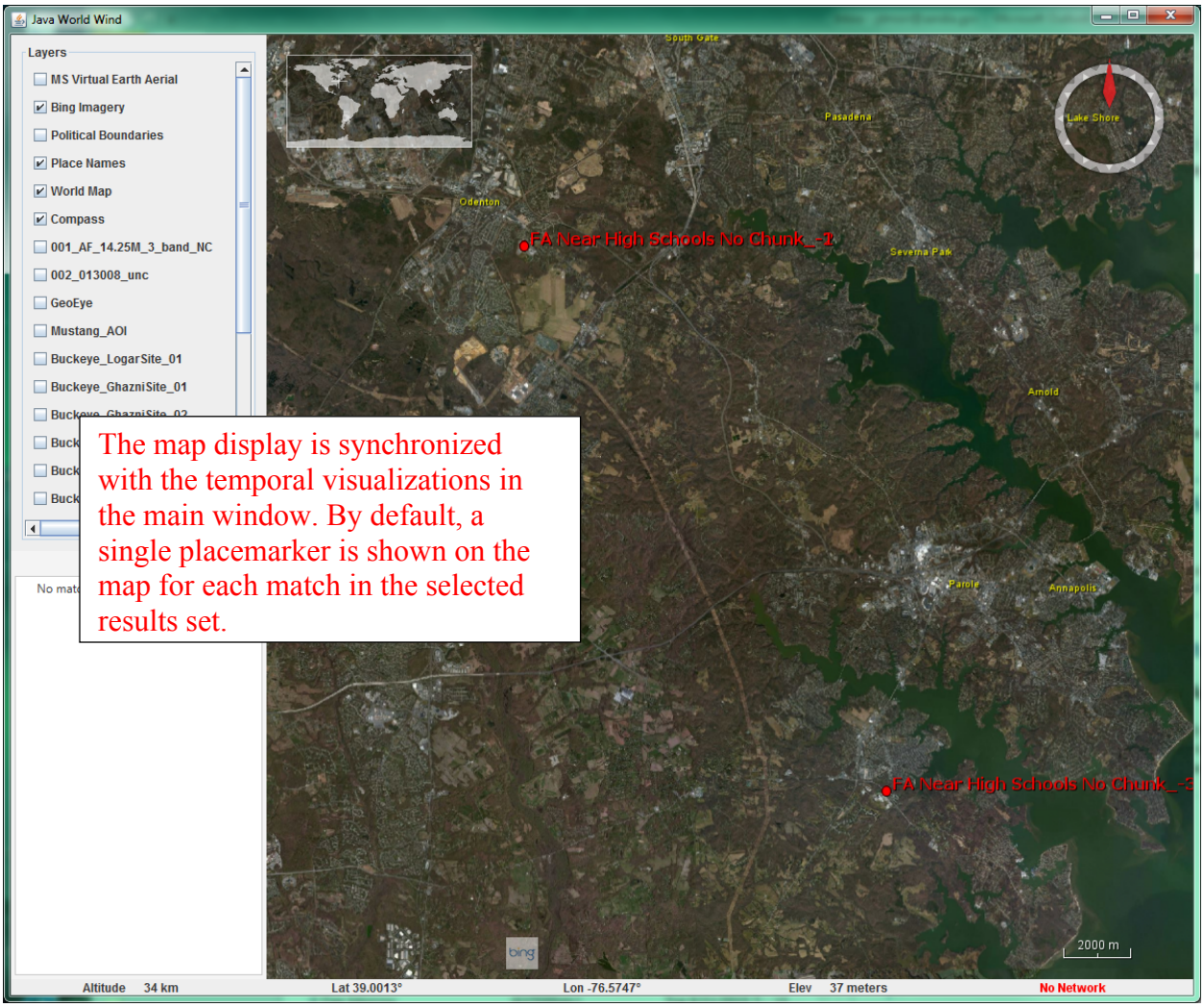

Figure 29: Viewing results on the map

Select a row in the Summary table to view details about that match on the map.

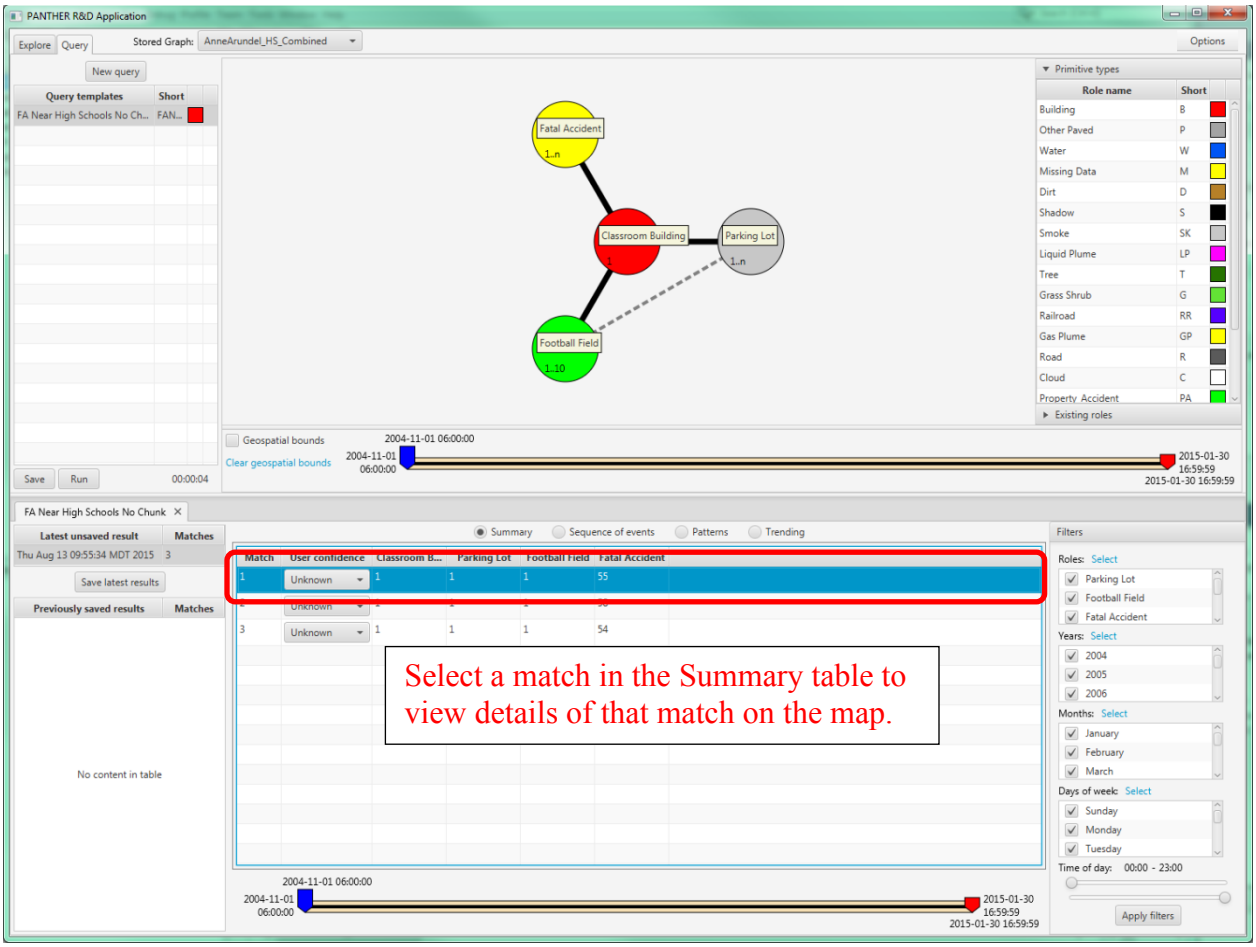

Figure 30: Selecting a match

Selecting a match expands the single placemarker into multiple placemarkers – one for each node in the match. In the example below, there is one placemarker for the Classroom Building (red), one for the Football Field (green), one for the Parking Lot (gray), and one for each of the many Fatal Accidents included in this match (yellow).

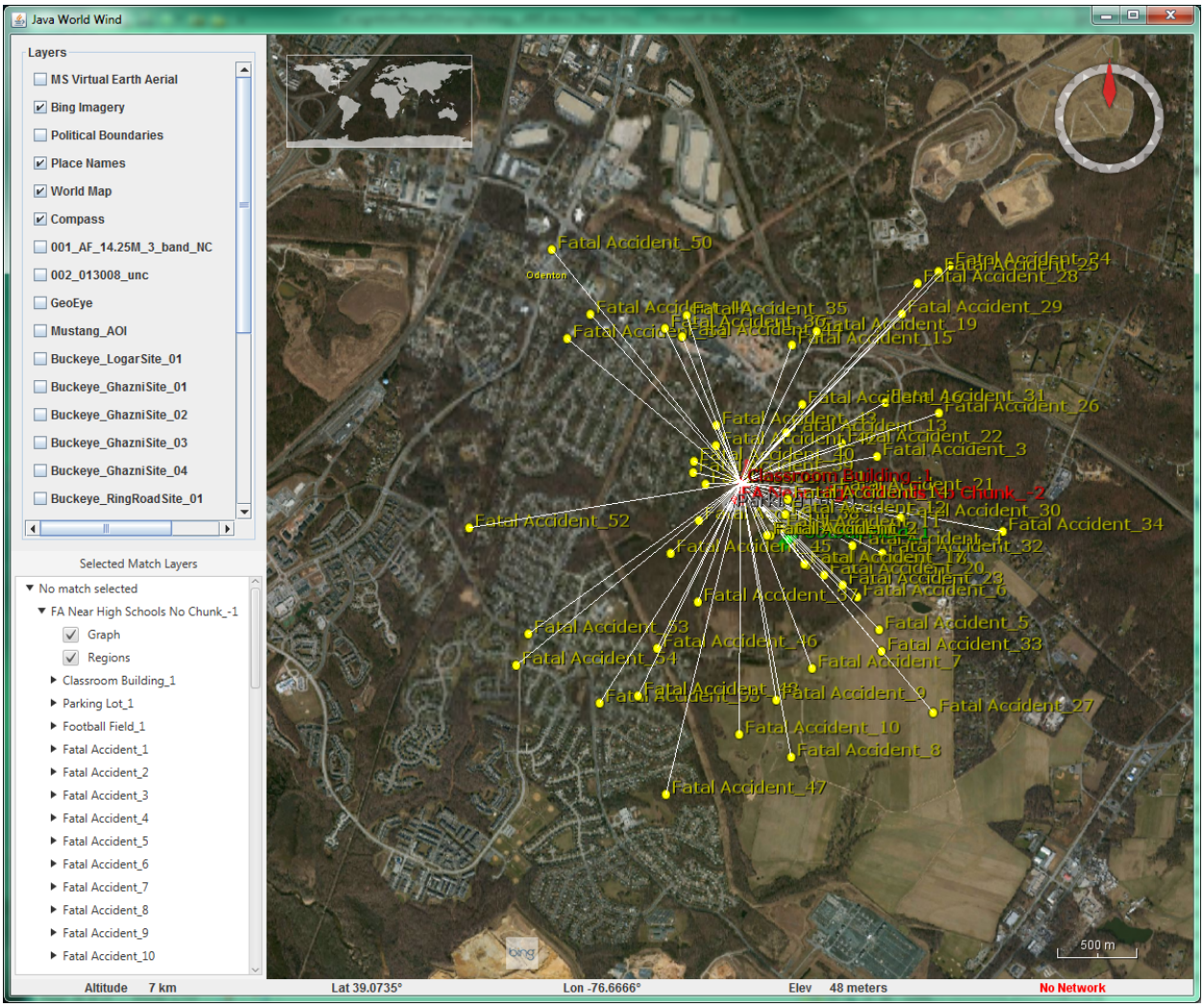

Figure 31: Viewing match details on the map

The user can zoom and pan the map using the mouse to view the details of the match.

For the region nodes (Classroom Building, Football Field, Parking Lot), a region of the same color is overlaid on top of the default optical imagery.

The user can toggle on and off the graph and region overlaid layers to declutter the display. The user can also expand the node entries in the tree to the bottom left of the map and choose to overlay available source imagery from the stored graph.

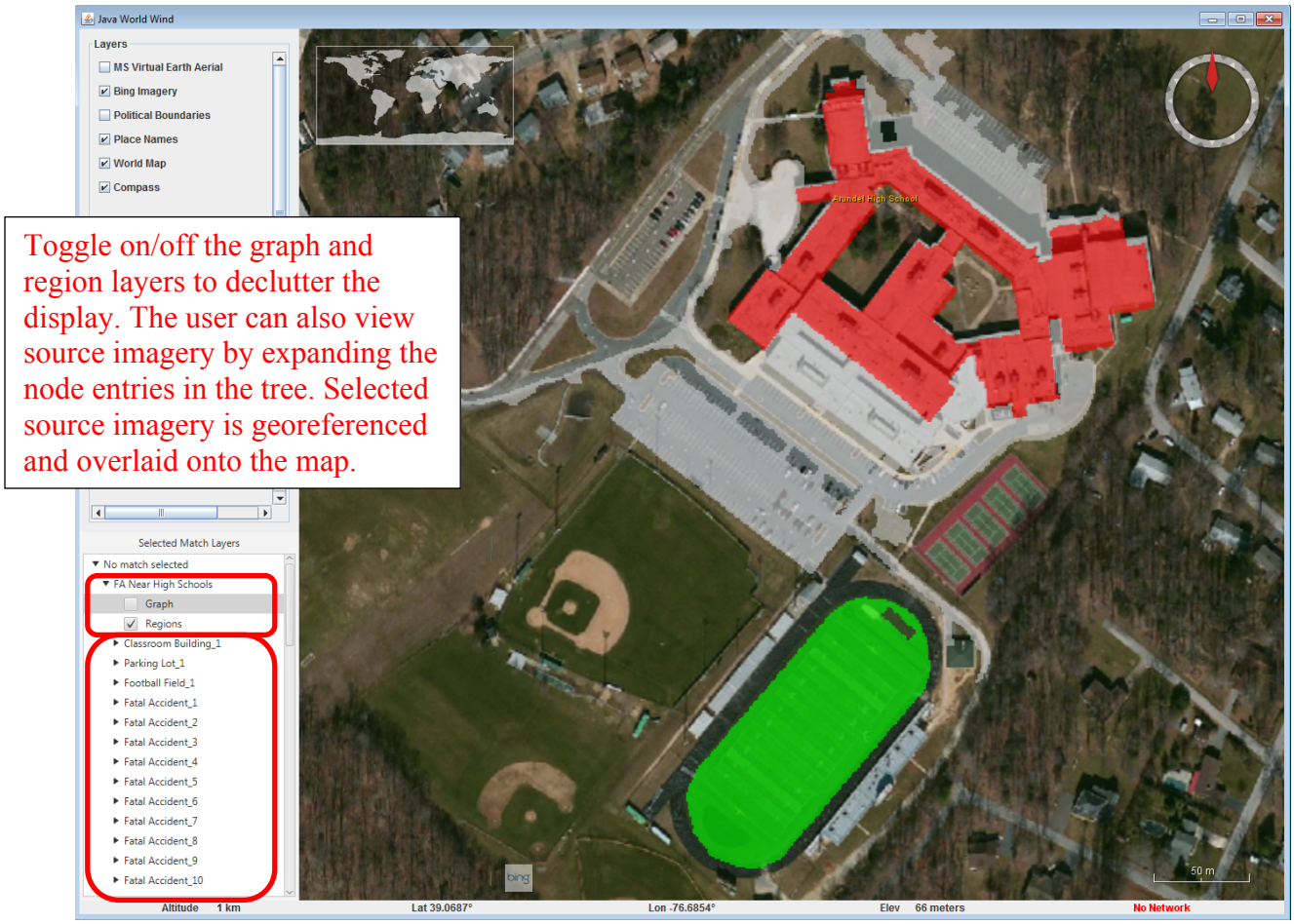

Figure 32: Toggling map layers

Back on the main display, click on the Sequence of Events radio button to view a more detailed depiction of how the nodes in each match are distributed through time. In this view, there is again one row per match, and the horizontal axis is a timeline. Colors in this view also match the colors defined for each role in the query template. In the example below, the red line represents the Classroom Building; the green line represents the Football Field, etc.

The vertical lines represent the start and end times of observations of each node. A solid horizontal line represents the portion of time when that particular node was known to exist – based on observations. For durable nodes, such as the Classroom Building, a dotted horizontal line indicates the assumed existence of that node. If the data set contains no follow-on observation where that node did NOT exist, then it is assumed to exist in both the past and future.

For ephemeral point data, such as the Fatal Accidents shown in yellow, the vertical "tick mark" represents the moment in time when that accident occurred. Depending on the data, ephemeral nodes can also have non-instantaneous time ranges, which would be displayed in the same manner as durable nodes (but without the dotted line).

The user can adjust the timeline below the temporal views to zoom in on a smaller portion of time. Click and drag the blue and red tabs to adjust the time range. The four temporal views, along with the map display, are all updated to reflect the current time range.

Alternatively, drag the blue and red tabs close together and then press Ctrl+Right-Click on the blue tab to bind the two tabs into a single draggable piece. Drag the piece through time to "play" the data. Data on the map is synchronized with the data shown in the four temporal views.

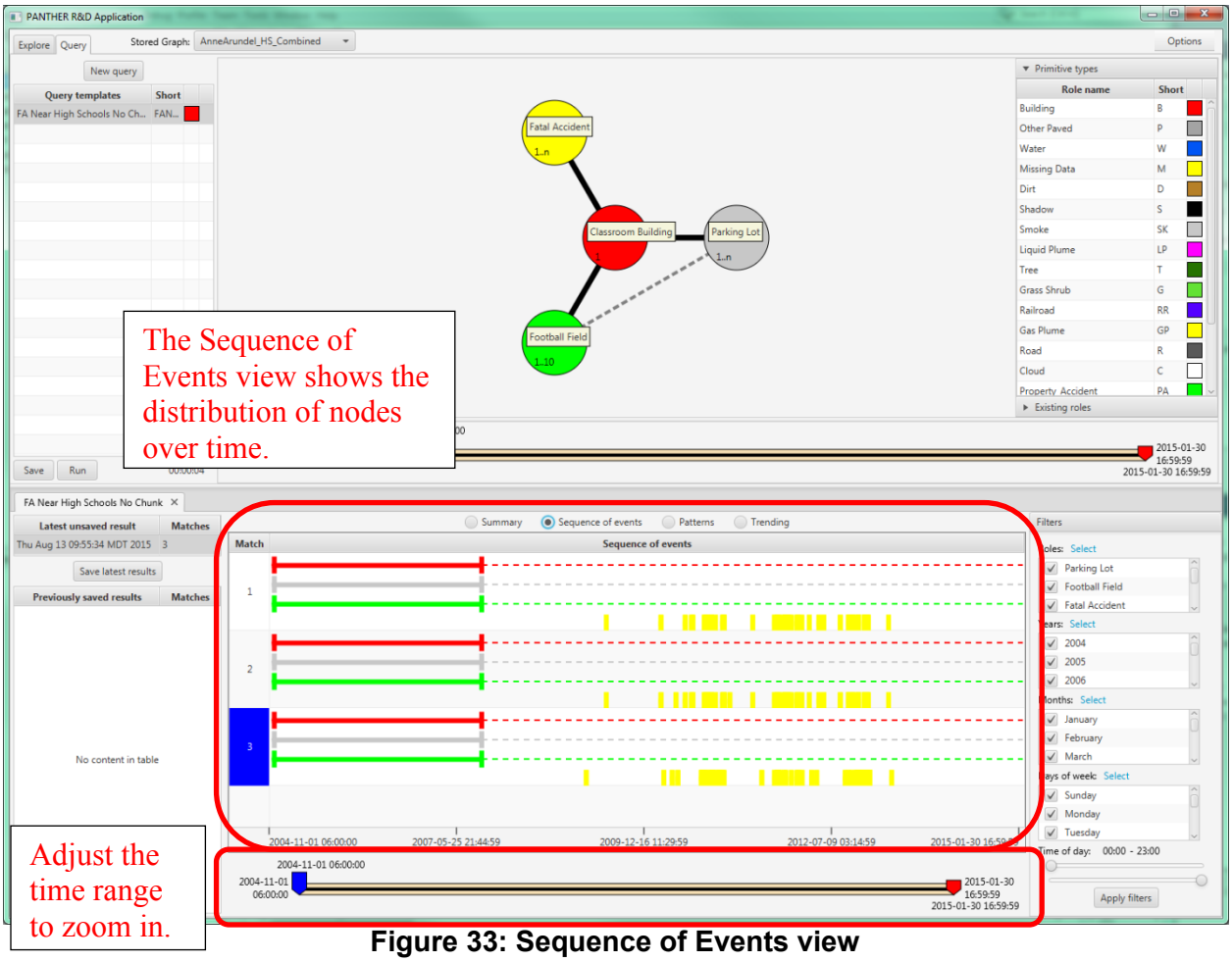

Figure 33: Sequence of Events view

Select the Patterns radio button to view a calendar-based view that highlights temporal patterns in the data. The rows in the table are divided into 6-hours chunks, and the horizontal axis shows days of the week. A bar chart, with one bar per node, displays node counts for each cell in the table, i.e., for each 6-hour chunk of each weekday. This view highlights days of the week and times of day when certain types of nodes are occur more or less frequently.

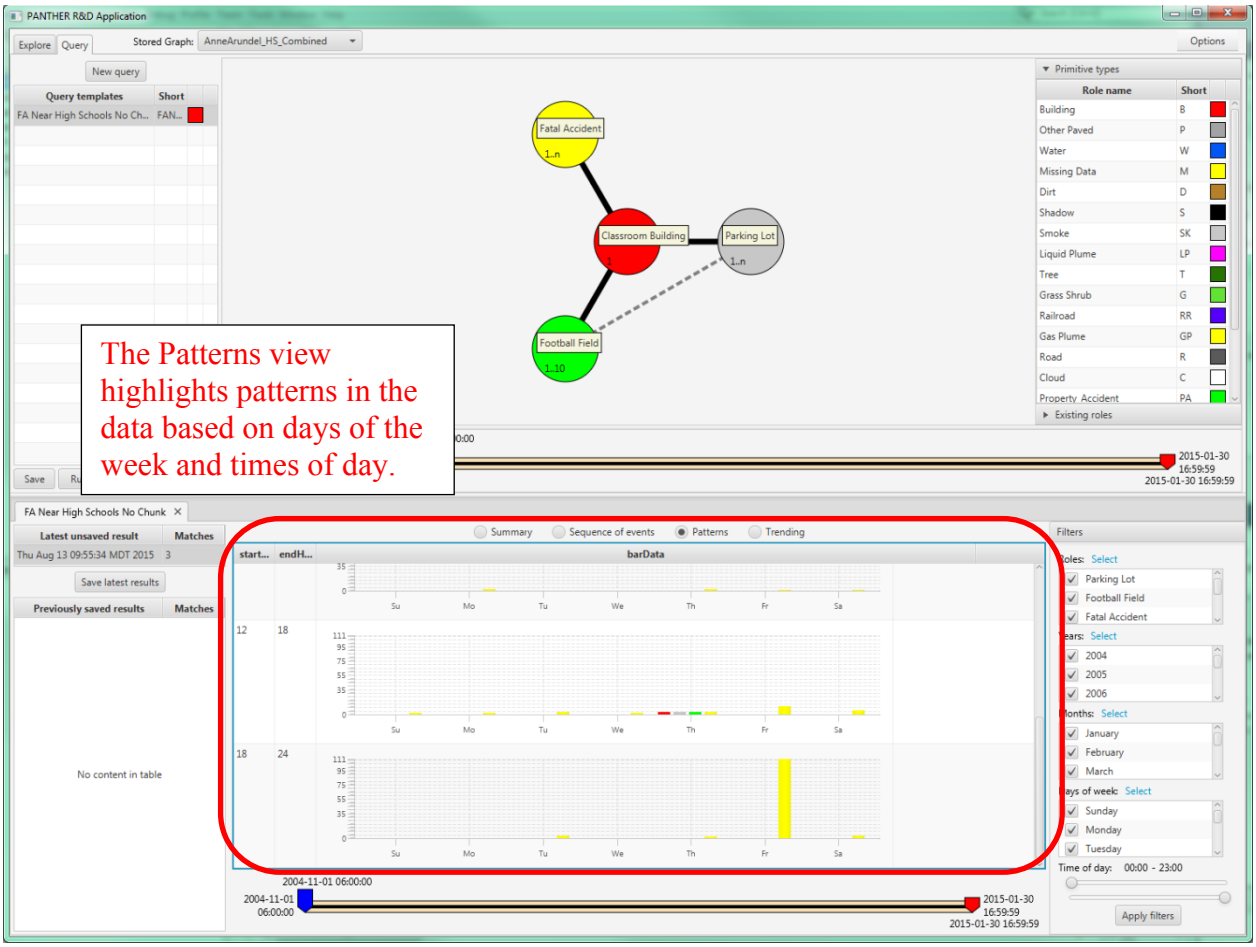

Figure 34: Patterns view

Select the Trending radio button to view data trends over time. The top portion of the Trending display is a heat map. Again, there is one row in the table per match, and the columns are binned based on configuration time frames. To adjust the granularity of the bins, use the Distribution text field and the dropdown menu. By default, the data is binned by years. Other options include months, quarters, weeks, and days.

The table cells are colored according to a color gradient. The darker the blue color, the mode ephemeral nodes that took place within that bin of time. The Total column indicates the total number of ephemeral nodes that were binned for that match.

Below the heat map, a line chart shows per-node trends for a single match selected in the heat map table. Again, colors match the role colors defined in the query template.

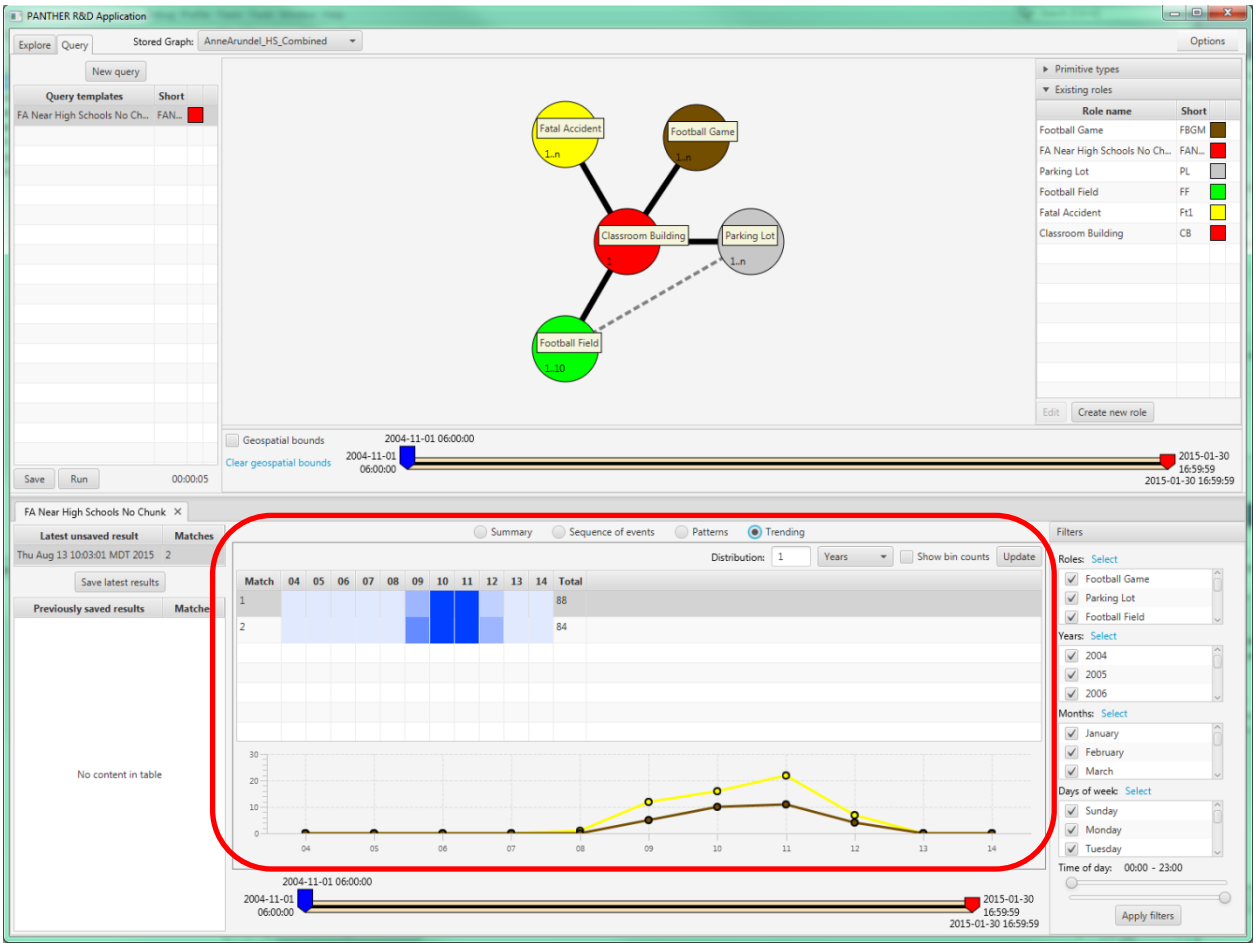

Figure 35: Trending view

The user can adjust the filters on the right side of the display to narrow in further on particular patterns or trends in the data. Use the filters to limit the roles that are plotted in the temporal displays and on the map. Or, limit the data based on temporal patterns. For example, only show data from the selected years, or for all years but only for the selected months, days of week, or times of day.

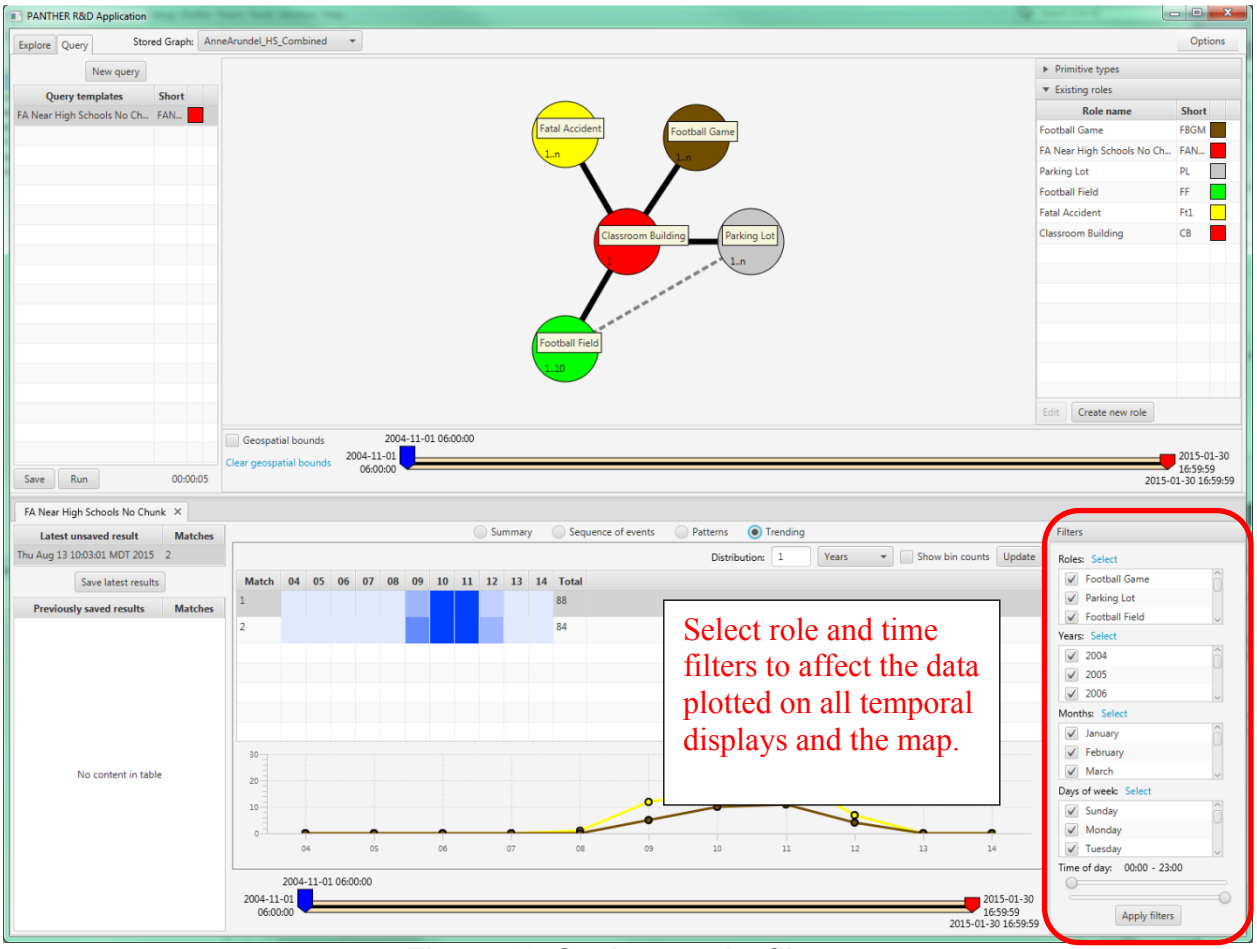

Figure 36: Setting results filters

5. REFERENCES

1. Google Earth, https://www.google.com/earth/, Accessed 21 August 2015.
2. Java FX Oracle Documentation, http://docs.oracle.com/javase/8/javase-clienttechnologies.htm, Accessed 21 August 2015.
3. Java Native Access (JNA), https://github.com/java-native-access/jna, Accessed 21 August 2015.
4. NASA World Wind, http://worldwind.arc.nasa.gov/java/, Accessed 21 August 2015.

DISTRIBUTION

1	MS0899	Technical Library	9536 (electronic copy)
1	MS0359	D. Chavez, LDRD Office	1911


Sandia National Laboratories