Relations between Results of Different Analysis Threads

Markus Scheidgen

Department of Computer Science, Humboldt Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{scheidge}@informatik.hu-berlin.de

Abstract. Markus: TODO

1 Introduction

Complex analysis scenarios lead to a multitude of analysis results. Each results presenting its own information, but each result is also related to other results. The ability to relate different results to each other is a key aspect to interpreting the results.

1.1 The Final Goal – Visualization of Relations between Analysis Results

Different threads of analysis on the same or on related data produces results that are related to each other based on the same or on related origins. These relations between the results of different analysis threads are non-trivial and hence hard to find, yet to visualize.

Fig. 1 shows a mock-up of an screen shot of an potential data analysis visualization. The screen shows the results of different analysis threads of the same WSN: the topology, properties of links, time series on the busyness of the radio hardware, and a time reference. The different results are related to each other. The link properties are related to the links in the topology; the time series are related to the nodes are taken from; and each result is related to a point or frame in time.

1.2 The Problems

Problem 1: No Traces on Model-Level Data analysis takes an initial set of data and produces new sets of data in a sequence of analysis steps. During this process no traces between the different data items are created. This means there is no formal relationship between elements on the data-level (model-level). Formal relations do only exist on the analysis-level (meta-model-level). These relations are defined by the analysis description.

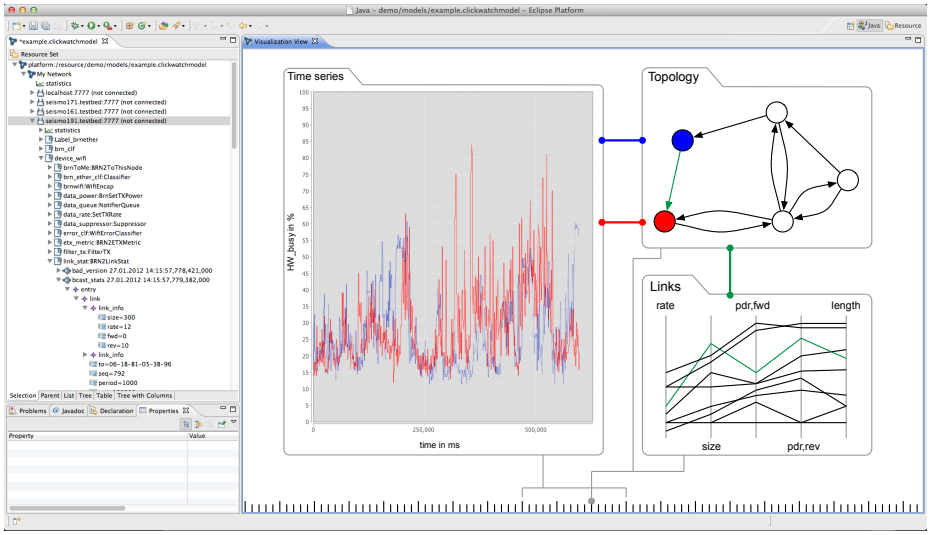

Fig. 1. A mock-up screen shot of a tool that visualizes relations in different analysis results.

Problem 2: No Feasible Data Store for Related Analysis Results If we would keep traces between analysis results, initial data, analysis results and traces would form one interconnected information model (graph, refer. to Fig. 3. This information model must be accessed (navigation along traces) efficiently.

Problem 3: Relations on Model-Level can only Be Seen In Context Of Relations on Analysis-Level Imagine an example drawn from a WSN analysis. The analysis has three threads and therefore produces three results for each node. We take a single result from one thread and are interested in the related result of another thread. We can navigate the trace of the taken result back to its origin. When we reach the origin, we are confronted with a multitude of different traces and we can only determine their meaning if we look at the analysis description. We need to know two things, which traces correspond the the second tread we are interested in, and which traces belong to the node that our original value belonged to. The analysis descriptions can tell us, how each trace was produces. Hence, if we look on the relations on both model and analysis-level, we can answer the questions.

Problem 4: Implicit Relations in the Initial Data Items in the initial data often contains string references to other data items. These references represent links that are valuable to the analysis and to our efforts to relate analysis results to each other. In order to use them, these implicit links need to be made explicit. String references must be resolved and replaces by actual model links.

1.3 The Solution

Solution to Problem 1: Model-Level Traces Typical ways to describe data analysis is using languages such as Matlab, R, Java. We can't change that. Therefore, we need a language that allows us to describe analysis as before, but in its semantics does not only realise the analysis but also creates traces and annotates the result accordingly.

Language like Scala provide enough flexibility to provide internal languages that might look like regular programming/analysis/transformation languages, but also hide additional behaviour. We will use Scala to provide an analysis language that creates traces transparently.

Solution to Problem 2: Data Store Large (EMF) models can be stored to different resources. The model is fragmented into different fragments and each fragment is stored in its own resources. References between fragments are realized with URIs identifying elements in different resources.

We use a key-value data store (e.g. HBase) to store large models. Each resource becomes a value and the resource URI the key. With clever fragmentation the model can be created and accessed fast. For example, to navigate a n-depth analysis thread to a related item in another thread, in a model with m data items, it always takes less then $2\,(\mathcal{O}n * log(m)($ to access the origin and navigate to the related data item of another thread.

Solution to Problem 3: Model-Level and Analysis-Level Relations Markus: Is this really a problem?

Solution to Problem 4: Implicit References There are obvious solutions, but we need on that operates of automatically as possible. One approach is to annotate the meta-model (which is generated in case of ClickWatch and HWL).

2 Analysis Traces (Problem 1)

2.1 The examples

All examples in this paper are taken form the domains *wireless mesh networks* (WMN) and *wireless sensor networks* (WSN). Specifically, all example are taken from the *Humboldt Wireless Lab* (HWL) test-bed. The HWL test-bed is a WSN based on WMN technology. Therefore, the HWL test-bed is both a WMN and WNS.

2.2 The nature of data and information

Terms like data or information are ambiguous and hence have potentially different and confusing means is different communities. *Disclaimer:* The following is not an attempt to provide some sort of commonly accepted information theory; it only serves the cause of this paper.

We start with the smallest pieces: *atomic pieces of data* (APD). APDs do not contain other data but themselves. An APD has an *identity*, *value*, and *data type*. The set of values of all APDs of a certain type form the *defining set* of that data type. Many data types are defined through (subsets) of real or natural numbers, such as temperature values between -30 and 100 degree Celsius. A set of labels is also a typical APD type. But data types are not limited to numbers and strings. Tuples of numbers, bitmaps, graphs, etc can also be the values of APDs, but only if these APDs are never considered to consist of multiple parts. Each APD has an identity. Two APDs from type integer, are not necessarily the same just because their values are equals.

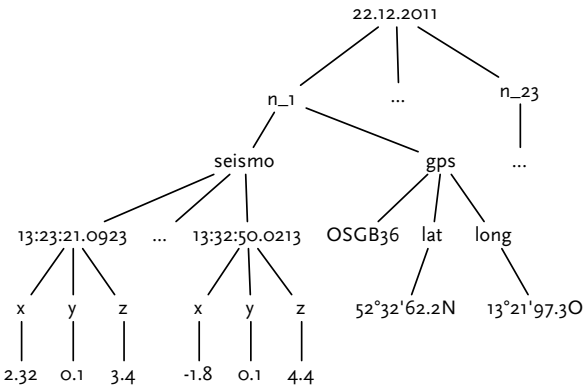

Fig. 2. Experiment data (information) from the HWL a WMN/WSN with seismo and gps sensors

Because each single APD does not have any context, it is considered *data* and not information. But APDs can be linked to each other. We consider graphs of APDs as *information*. The vertices are takes from a set of APDs and edges connect different APDs. In such a graph each APD has a *context*: that are the neighbors of an APD.

We assume that each information graph contains a designated information tree. An *information tree* is a directed and acyclic graph subgraph that covers all vertices (i.e. all APDs). Edges are directed from *child* to *parent*. One child can have multiple parents, one parent can have multiple parents, diamonds are allowed. An information graph can contain multiple information trees, but one tree must be designated and is called *the information (or composite) tree of an information graph*.

In an information tree edges provide an antisymmetric relationship between APDs. The reflexive transitive hull of this relationship induces a partial order among APDs in an information graph. We simple say an APD is *related* to another APD, iff these APDs are related with respect to this reflexive transitive hull. Note that children are only related to parents and not vice versa. The set

of all related APDs of an APD is called *ancestors* of that APD. The joint of ancestors of a set of APDs is called the *common ancestors* of that set. The set of smallest ancestors is called the *closest ancestors* of a set of APDs. Two APDs are *siblings* if they have a non empty set of common ancestors.

The information graph if Fig. 2 for example comprises of all the data from an experiment with a WSN of 23 nodes and two sensors (seismo and gps). This includes APDs with several types: date, time, node identifiers, sensor identifiers, spatial axis, accelerometer readings, degrees (long/lat), and coordinate reference systems. The ancestors of each accelerometer reading contain a axis, a time, a node, and an experiment date. Each accelerometer reading and each gps coordinate at least share the experiment date as common ancestor. An accelerometer reading and a gps coordinate of the same node share also the node as common ancestor.

Information has also a type (*information type*). Consider the WSN from the last example: all possible information graphs from experiments with this WSN share a unique structure. There are constraints for the set of possible APDs and possible edges.

There are several technical systems to create, store, and access information. XML files can contain information trees (and graphs) where each entity (or attribute) represents an APD. XML schema define types of information. Relational databases organize information in tables and references between entries of different tables. Each value (specific entry, specific column) in a relational database table is a APD. Entries and relationships form links between APDs. Entity relationship diagrams or database schemas define types of information. Models (as in OMG) are information, objects (and attributes) are APDs. Meta-models define types of information. Within programming languages data structures (classes, structs, union, arrays and primitive types) are used to represent information. Further systems are based on ontologies or RDF.

There are several abstractions for the representation of information. We may call graphs of APDs information, but these graphs are actually only one possible abstraction. Other (somewhat limited) representations are list, maps, functions, terms, different forms of trees and graphs, algebras, vectors, matrices, tables, etc. Of cause different representations can be combined.

2.3 Analysis and Traces

The information graphs considered in this paper (e.g. the information graphs defined and exemplified in section 2.2) are to be analyzed. Information is usually created during experiments. These experiment results are on low level of abstraction and we want to extract information on a higher layer of abstraction. The term *analysis* describes the process in which new information graphs are created from existing information graph. An analysis can be divided into analysis steps. In each *analysis step* a distinct information graph is created. An analysis can have multiple steps and hence produce intermediate information graphs, i.e. intermediate results. Fig. 3 shows the information graphs of an example analysis.

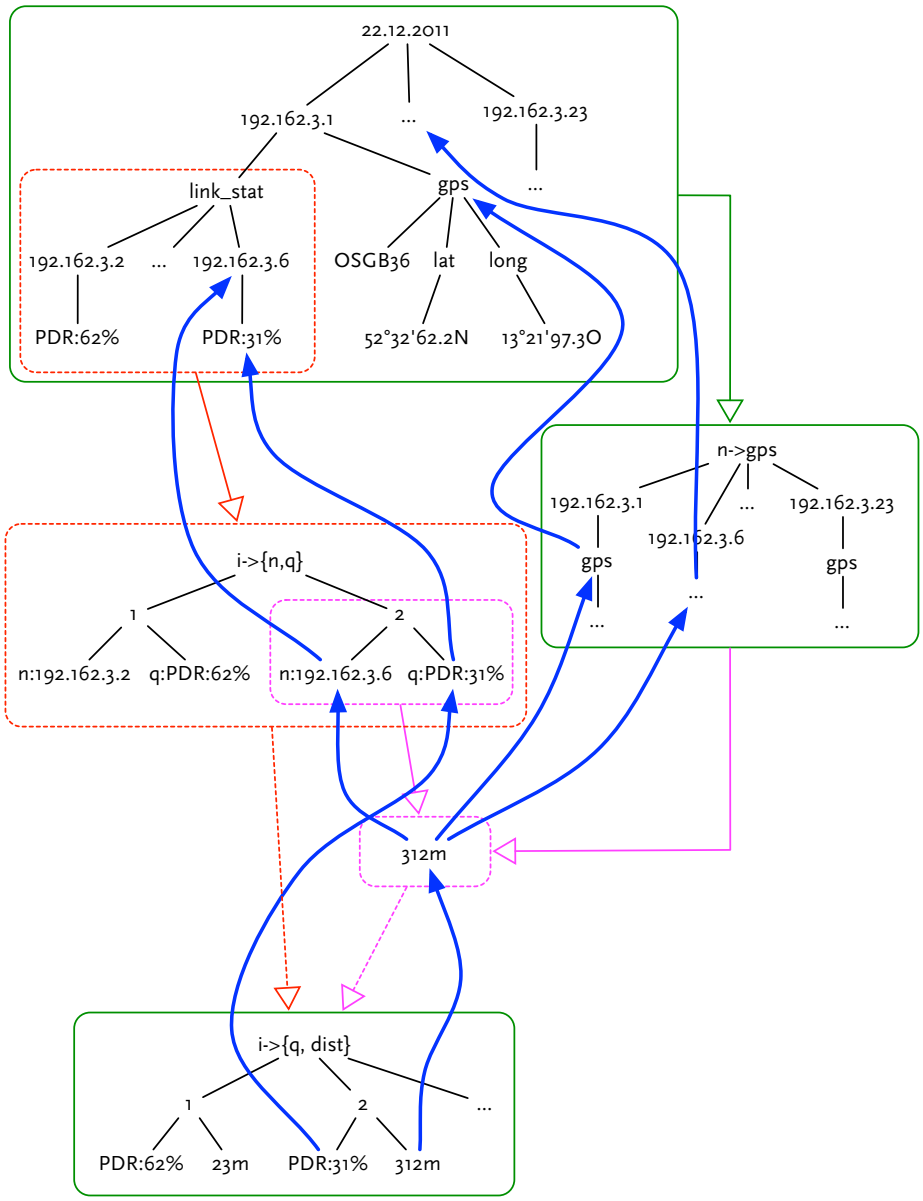

Fig. 3. An example for Information in different analysis steps and traces between its APDs. The example is taken from an experiment with the HWL network. The information obtained within the network includes information about links and link quality (packed delivery rate, PDR) and the position of the nodes (via global positioning system, GPS). The colored boxes mark distinct information graphs. The thin colored arrows mark analysis steps, the arrow point marks the created information graph. Green boxes mark information graphs that cover the whole experiment, other colors mark information graphs that only cover single samples. In the example, the red box describe the links of one network node, the purple boxes describe a single link. The thick blue arrows mark traces between APDs: the arrows show which APD is based on what other APDs.

When a new information graph is created during an analysis step, APDs of the new graph can be assigned to APDs of existing information graphs: new APDs are either directly taken (copy) or are computed from one or more existing APDs. This creates a directed relationship between APDs of different information graphs. This relationship is antisymmetric. The reflexive transitive hull links one APD to all those APD that participate in its creation (including the APD itself). We denote this relation *trance relation*. The trace relation induces a partial order. The ordered set of APDs that participate in the creation of an APD (including the APD itself) is called the *trace* of that APD.

Original information graph, intermediate and final results of an analysis as well as the trace relationship can be used to form a union information graph. The trace relationship and all information trees compose the composition tree of that new information graph. We call this information graph the *analysis information graph* and the original information graph the *experiment information graph*.

Analysis information graphs can be further extended. Depending on the type and semantic of APDs, we can define (usually equivalence) relationships on the values of APDs. For example: to denote close proximity GPS coordinates can be designated equivalent, timestamps can be designated equivalent if they shell denote the same point in time, etc. This way, two APDs can be *weak siblings* if they share ancestors with different identity but (similar or) same meaning.

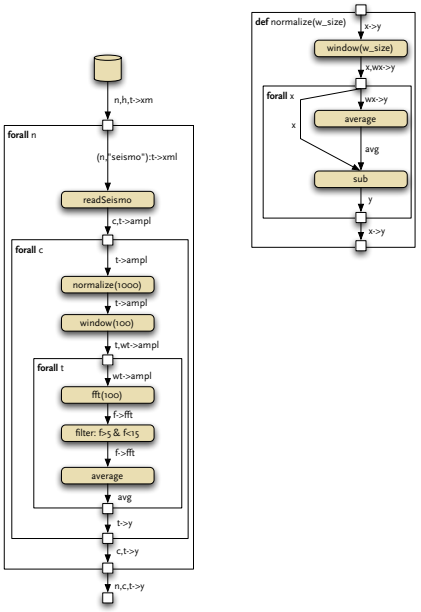

Fig. 4. Seismo analysis composition.

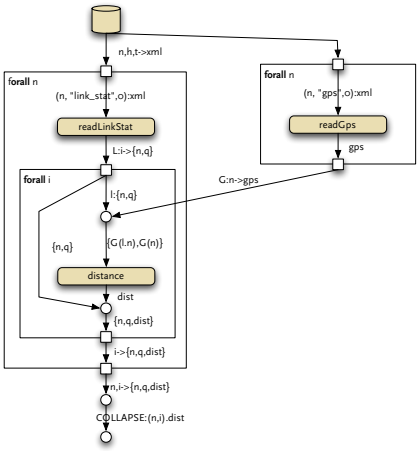

Fig. 5. Link distances composition

Examples taken from experiments with wireless mesh networks, the experiment pictured in Fig. 3. The gps coordinates of two wireless nodes are siblings with the experiment time as closest common ancestor. The gps coordinates of two wireless nodes that form a link are siblings to the link quality of that link with the link as closest common ancestor; further ancestors are the nodes of the link and the experiment time.

Fig. 4 and Fig. 5 examples for the analysis of HWL ClickWatch data. These activity diagrams choreograph analysis steps (activities). Both are supposed to be performed on HWL ClickWatch data. This is are information graphs (trees). In those nodes have handlers and handlers have XML values at different points in time: hence n, h, t, xml as starting information graph. XML values are part of the information graph.

3 Storing Large Models (Problem 2)

Please refer to the paper *Model Fragmentation For High Access Performance of Models Persisted in Key-Value Stores.*