2016 IEEE 24th International Requirements Engineering Conference

AnalyticGraph.com: Toward Next Generation Requirements Modeling and Reasoning Tools

Joseph Gillain, Corentin Burnay, Ivan Jureta, Stéphane Faulkner
Data Analysis Decisions Advice (DA^2) Research Unit
Department of Business Administration & PReCISE Research Center, University of Namur

***Abstract*—Graphical Requirements Modeling (GRM) consists of representing requirements in diagrams: requirements (and other relevant information) are represented as nodes, and relationships between them as edges. Relationships can show, for example, that one requirement refines another, that some are in conflict with others, that they are more or less desirable, and so on. Various software tools have been proposed over the years as a support to doing GRM, some capable of performing computations over diagrams, such as searching for text strings, or determining if a requirement is satisfied (and how much). We present yet another tool, available at AnalyticGraph.com. The tool departs from much of prior work in the following ways. (i) The tool is a web application, is available on-demand, and requires no installation of specialized software. (ii) Each model made with the tool gets its own permanent and unique URL, so that models can be linked in research papers. (iii) If a model on Analytic Graph is linked in a paper, then any reader can click on the link, open a free account, and edit and run a copy of the linked model. (iv) The tool supports the definition of various requirements modeling languages. (v) Models are stored in a graph database, and standard graph queries (such as find the shortest path between two nodes) are included by default. (vi) It is possible to combine models made with various languages, and do computations over the resulting mixed models.**

I. Introduction

The aim of this paper is to revive the old, but critical research topic of software tools for Graphical Requirements Modeling (GRM). We use the term GRM, to refer to the widespread practice in the RE field of representing requirements and their relationships in diagrams, and of doing reasoning (computation) on these representations. Typically, nodes in such diagrams hold information about the requirements, environment, or the system to build or change, while edges are labeled by relationships holding over the connected nodes; such relationships can say, for instance, that two requirements are in conflict, that some set of more concrete requirements refines a less concrete requirement, they may convey the relative desirability (preference) or importance (priority) of requirements, and so on. Various Requirements Modelling Languages (RMLs) [1], [2], [3], [4], [5], [6], [7] rely on GRM in the sense that models made with these languages are visualized as diagrams.

To this aim, we present and argue for a series of features that next generation GRM tools could have, and we present an early version of a new software tool, freely available at AnalyticGraph.com, which implements these features. We use the tool to illustrate the discussion in the paper.

Our aim is not to propose a definite list of features for GRM tools, but instead to stimulate discussion and hopefully renew interest of colleagues in coming up and validating the relevance of new features for software tools which have received little attention and benefited from little innovation in the past years.

The paper is structured as follows. Section 2 presents some existing GRM software tools and their respective features. Then, in Section 3, we discuss strengths and weaknesses of existing tools. We introduce Analytic Graph's architecture in Section 4. In Section 5, we show with examples how this architecture allows Analytic Graph to mitigate weaknesses. We finally conclude and discuss future work in Section 6.

II. Brief Tour of Existing GRM Software Tools

While vector drawing software can be used to do GRM, this is rarely, if ever advocated. It is common to see GRM being done with generic diagramming tools, which may or may not include graphical primitives corresponding to specific RMLs. We consider such tools as DIA, Microsoft VISIO, draw.io, yEd or OmniGraffle to be examples of generic GRM tools. Their limitation is that they offer little support to users in terms of syntax-checking and computation over the models (diagrams) made.

More comprehensive tools have been developed in RE to deal with such limitations. For instance, Tropos comes with the java-based GR-Tool, which proposes forward and backward reasoning on goal models [8], [9], [10].

In line with the Tropos methodology, TAOM4E supports a model-driven, agent oriented software development, and has been designed to respect the Model Driven Architecture (MDA) recommendations [11].

RE-Tools [12] is another modeling tool that supports notations such as i*, the NFR Framework, KAOS, Problem Frames, and UML. Among other things, RE-Tools supports the combination of previous notations, enabling engineers to combine functional and nonfunctional requirements, agents, goals, soft-goals, formal goals, and objects into one single diagram.

DesCARTES [13] is a Computer-Aided Software Engineering (CASE) Tool, which also supports i*, NFR models, UML models, and I-Tropos developments. It takes the form on a Eclipse IDE (Integrated Development Environment) plug-in.

2332-6441/16 $31.00 © 2016 IEEE
DOI 10.1109/RE.2016.33

RE 2016, Beijing, China
RE@Next! Paper

jUCMNav [14] is another example of Eclipse plug-in which permits the modeling of requirements based on the User Requirements Notation.

MetaDONE [15] is a tool supporting domain specific modeling languages (DSML); it is aimed at helping engineers in the more effective implementation of software systems, based on the production of generative methods [16].

III. Strengths and Weaknesses of Existing Tools

Although existing GRM tools clearly differ in the type of language they support or the nature of reasoning they enable to perform on requirements models, they all seem to support requirements engineers in at least one of the following complementary ways:

- They offer symbols and visuals for a given notation, keep data and meta-data related to these elements, and ensure these elements are combined in a way that complies with the syntax of the modeling language.
- They offer reasoning capability about a model to identify solutions, discover alternatives or resolve conflicts.
- They offer capability to design their own notation and behavior capabilities, fitted for the actual domain.

These are the three main pillars of GRMs, on which we continue to build Analytic Graph.

Beside those strengths, we see a number of improvements which could be made on existing GRMs, that would further help requirements engineers in the modeling and analysis of the requirements. Those improvements, which we discuss with more details in the remainder of the paper, could be summarized as follows:

- *Portability*: models defined in one GRM, on one computer, are difficult to transfer to other GRMs/computers.
- *Collaboration*: existing GRM are not designed to ease collaboration between several engineers on same models.
- *Navigability*: models can get very large, and existing GRM offer no support to navigate effectively in the many nodes and edges that constitute a requirements model.
- *Reasoning*: current GRM offer predefined computations on models, leaving no room for custom reasoning.
- *Extensibility*: existing GRM are usually designed to support one or more preset RMLs, without the possibility for users to add new RMLs.
- *Flexibility*: it is normally not possible to make models made by bridging several models from different RMLs.

Our overall aim in making Analytic Graph is to consolidate the strengths and address the limitations of existing GRM tools. Although Analytic Graph is not fully developed, it is available for use and already illustrates many of its key features.

IV. Analytic Graph Architecture

Existing GRMs are predominantly desktop tools, which hurts portability and collaboration. GRMs do not store and treat requirements models as graphs, which does not help navigability and custom reasoning. In this section, we describe how the architecture of Analytic Graph differs from classical GRM tools. Next section discusses how that architecture helps in relation to GRM limitations.

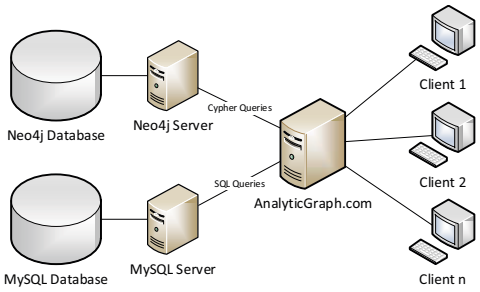

Fig. 1. Architecture of AnalyticGraph.com

The architecture of Analytic Graph is illustrated in Figure 1. Our system is composed of one web-server on which AnalyticGraph.com is hosted. The web-server runs queries against a Neo4j server, which manages the storage of graphs. Neo4j is a graph-oriented and open-source database management system. It receives Cypher queries through a web-service from AnalyticGraph.com and returns json files with the result of these queries. AnalyticGraph.com also runs SQL queries against a MySQL server, to store or retrieve any meta-data on the graph (graphID, owner...) as well as data that is not related to a model; for example, requirements notations, logs, user account data, etc. Clients interact solely with Analytic Graph using a web browser, as explained in next Sections of this paper.

A. AnalyticGraph Stores Models as Directed Graphs

A limitation we emphasized in our introduction is that existing tools do not enable users to define and execute their own queries/reasoning algorithm on their models. To deal with this limitation, we designed Analytic Graph as a self-service system; we intend to let users define themselves the queries they want to run against a model, to define or import the requirements notations they want, with as little constraints as feasible on how the model is stored. The only constraints are those imposed by the RML which the user wishes to use.

This approach led to important constraints on the way RE models should be stored in Analytic Graph. Namely, we need to store the model in a way that is amenable to flexible querying and computation, while at the same time reflect the structure of graphical requirements models. We therefore store models as graphs. Each concept and link of a model (for example, a goal, a task or a contribution link in i*) is stored as a node of a graph. Under this scheme, the directed links in our graph are simply used connect two nodes of the model, and carry no other information than the direction. For example, a model in which a goal decomposes into two sub-goals will be stored in Analytic Graph as a graph composed of four nodes: one goal node, two sub-goal modes, and one decomposition node, with a link from each of the sub-goals

to the decomposition node, and a link from the decomposition node to the goal.

The Techne model [7] already adopts such view on modeling. It represents a relationship between two nodes (goals, tasks, etc.) as other nodes (inferences, conflicts, preference), so that links (edges) themselves carry no information other than their direction. Techne is already implemented in Analytic Graph and we use it in the rest of this paper for illustration.

B. Analytic Graph is a Web Application

Another limitation we emphasized in our introduction is the difficulty for users to share and interact with models. We consequently built in collaborative features, namely to let users share the models they created in Analytic Graph, or to access and interact with others' models in few steps. We wanted a tool that does not require specialized software to be installed on users' devices, and we therefore opted for an online platform. Analytic Graph is a web application, in which users can create, edit, save and load models and RMLs. By associating a unique URL to each model and notation, it is also possible for users to share their own models, link models in documents, research papers, presentations, and elsewhere, and to access and/or import in their own library, the models or RMLs created by others.

V. Candidate Features for Next Generation Tools

In this section, we discuss features which we consider important for next generation GRM tools. We do this by presenting how Analytic Graph addresses common limitations of GRM tools. For each limitation, we present a feature of Analytic Graph, and we illustrate and explain how that feature works.

A. Portability

If a GRM tool is desktop software, then it needs to be installed on the users' devices, and models will be stored by default on these devices. *Our tool is a web application; it means that it is available on-demand anywhere, and requires no installation of specialized software*[1].

The main requirements for accessing Analytic Graph is to have access to an Internet connection, and have a web browser installed on the device[2]. Once logged-in, users are able to save their models. The models are stored on the Analytic Graph.com server, which means that the users can access the model on any device, using their credentials. Models are by default private, so that a model created on one user account will not be visible to other accounts, unless he gets the URL. We also leave the possibility for users to design models without having an account, in which case the model can be exported as an image or transferred to an existing account, or lost (not saved) otherwise.

[1]The tool is accessible at analyticgraph.com/app. Tutorials are provided at analyticgraph.com/category/tutorial/

[2]At this stage, Google Chrome is the recommended browser for accessing AnalyticGraph.com.

B. Collaboration

With desktop GRM tools, a model is stored as a file and shared by sending the file among collaborators. If the model is shown in a research paper or other document, a reader needs to find the model source online, download and setup the relevant GRM tool, and only then work on the model, and use it in own research.

Analytic Graph was designed to simplify the process from seeing the model in a publication to being able to edit and run it. In the caption of Figure 2, we included a URL. Clicking on the URL should bring up the reader's web browser, and allow the reader to view, edit, and run on Analytic Graph the model shown in that Figure. It is a Techne model, of simple requirements for music streaming software.

Linking models for editing is made possible with two features of Analytic Graph: (i) *each model made with the tool gets its own permanent and unique URL*, so that models can be linked in research papers or other document types, and (ii) *any reader who has access to the URL of a model can click on the link, open a free account (or access as a visitor), and edit and run a copy of the linked model*.

Putting aside the graphical representation in Figure 2, any non-interactive visualization of the model – just as the one in that Figure and on any printed page and on any non-linked model – suffers from many drawbacks. For example, readers cannot run themselves computations on the model. They cannot reuse and enrich that model for their own purposes. They cannot navigate in the graph. And so on. Analytic Graph aims to facilitate the interaction with requirements models, by letting users access to and use the graph. As an example, we invite the reader to click the URL shown in the caption of Figure 2.

C. Navigability

We use the term navigability to refer to how one can search for and find sub-models of a given model. Sub-models may correspond to, for example, candidate solutions (specifications) to requirements, alternative refinements of a given requirement, all requirements which may be involved in a conflict, and so on.

Our approach to navigability in Analytic Graph is to *store requirement models in a graph database, use well known algorithms to search the graphs, and enable users to define their own queries on models / graphs*. A series of queries are predefined in Analytic Graph. Users open a model, select and click on the query to execute it against that model. Some queries are generic, in that they can be run on any graph, in an RML. Others are specific to an RML.

Queries we have been exploring so far do not add or remove nodes or edges in a model, but return a sub-graph. By combining several queries, users are then able to identify patterns in the model, select only a part of the model, identify a certain type of nodes, or identify the shortest path between two nodes. We see in this feature a first step toward more sophisticated queries and an interface for custom queries.

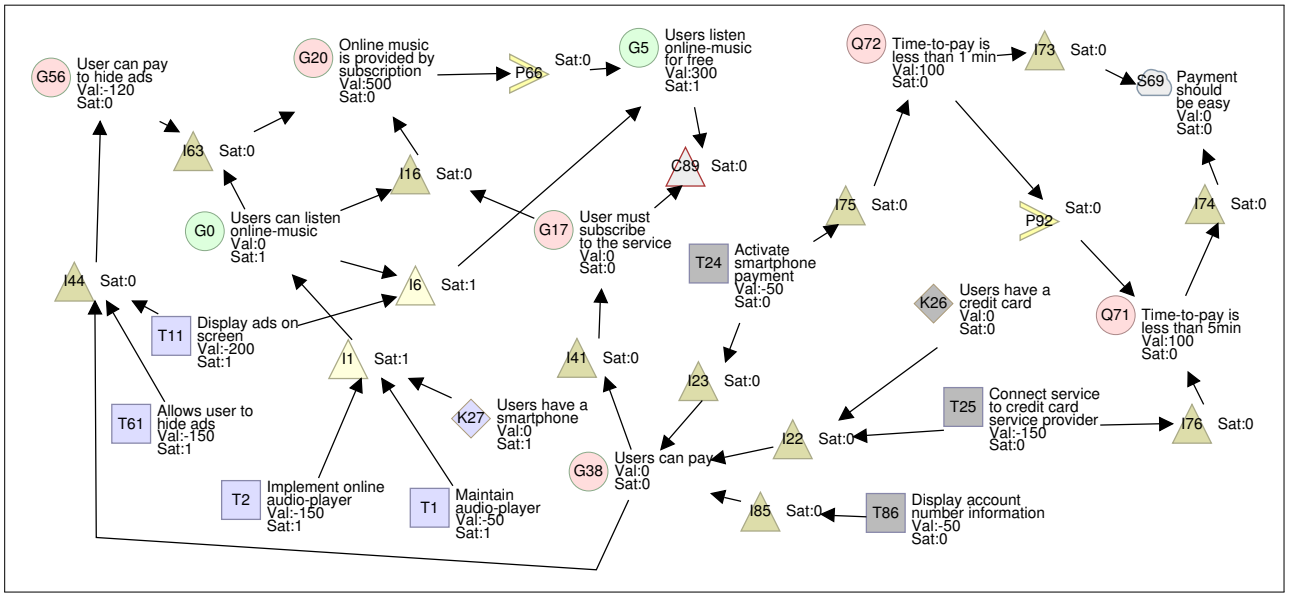

Fig. 2. Example of a Techne Graph in Analytic Graph- analyticgraph.com/app/?g=5u3lNJVddv

Consider our running example from Figure 2. The number of nodes is still limited, yet it might already be difficult to detect one particular set of nodes. Say for instance that you want to browse the graph in the refinment direction. You then need to locate sink goals in order to start the reading of the model. Using a simple search query, you could detect them instantly. The same search without a query would likely take more time, even if the number of goals is limited. Moreover there always is a risk you miss some of them. Similarly, one might run a query to count the number of conflicts, to select all soft-goals, to determine the entire sub-graph associated with one goal, etc.

D. Reasoning

Reasoning on a model allows engineers to answer questions, which the language was designed for. This is common to all RMLs, ranging from early requirement languages such as i* or Techne to later requirement languages such as BPMN, features diagrams and so on. For example, in feature diagrams, one question is to find some subset of interest, among all possible configurations of the modeled system. Without a proper tool capable of reasoning, the diagrams themselves are of limited use. Most GRM tools provide such features, but tend to be limited to one or more predefined reasoning ways on models.

Since Analytic Graph is based on a graph database, it comes with a manipulation language (called Cypher[3]) which has been used to design behavior on diagrams. Use of this language enables several ways of defining a behavior (transaction, node behavior), and leaves that decision to the user. For example, Techne mechanism of inference has been implemented by defining for each node a particular behavior. Each inference is associated with a query that evaluates if premises (i.e. incoming nodes) are satisfied, in which case it sets the conclusion node (i.e. the outgoing node) as true. This behavior is propagated from source nodes to sink nodes.

[3]http://neo4j.com/developer/cypher-query-language/

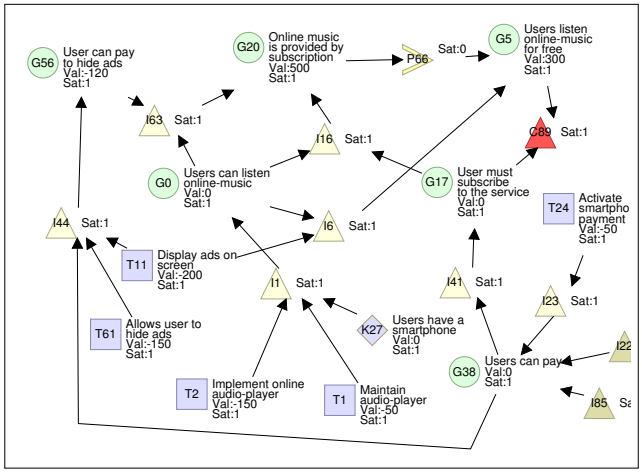

Fig. 3. Example of Reasoning on Techne Graph in Analytic Graph- Illustration of the propagation of satisfying task T24

Consider our running example as illustrated in Figure 2. The number prefixed with **Sat** below the name of a node is its satisfaction; 0 means that the node is not satisfied, 1 means that the node is satisfied. Note the conditional formatting of nodes, depending on their satisfaction (another customizable feature). In our example in Figure 2, the goal G17 *"User must subscribe to the service"* is unsatisfied. During requirements analysis, engineers might want to test the effect on the realisation of some tasks on this goal. Since satisfying goal G38 "Users can pay" should have an impact on G17, they just need to ensure the satisfaction of underlying tasks of G38. So, they set value

of T24 “Activate smartphone payment” to 1 (true), and run the model again (we invite the reader to open the model and run node behavior in Analytic Graph app). The result is reported in Figure 3; the figure shows that satisfying T24 enables to satisfy the goal G38 of the model, yet introduces a conflict with G17 (symbolized by C89). It also satisfied among others G56, G0 and G20. This is a simple example of how Analytic Graph implements reasoning on goal models.

As discussed earlier, other reasoning or treatment of the graphs are possible. Consider now the case where utility values can be associated to each node in the model. It is represented by the numbers prefixed with **Val** next to each node. Positive value means a revenue while negative value represents a cost. Having this information, it is possible to map the model to a mixed-integer mathematical program and execute an optimization on this [17] (we invite the reader to open the model and run optimization in Analytic Graph app). Given these values, the optimal solution allows to reach a value of 350 with following node satisfied: G0 G17 G20 G38 Q72 K27 T2 T24 T1 I1 I41 I16 I23 I75.

E. Extensibility

GRM are often designed for a specific RML. For example, GR-Tool implements the Tropos notation, jUCMNav implements the User Requirements Notation, etc. Analytic Graph has a language management module, which allows users to define their own RMLs, both in terms of graphical primitives and their properties used for queries and computation. Users are also free to build requirements models using the notation they prefer. In case users need other notations, they can either design that notation or import it.

Although **Techne** is the only built-in RML at the moment, our aim is to increase the number of built-in RMLs available to users. The tool will soon allows users to define relatively simple notations of their own. They will do it using the *Language Management* (LM) tool[4], which enables users to define virtually any language, as long as its models can be represented as directed graphs. The meta-model behind language definition is depicted in Figure 4. A *Language* has a name and is composed of several *NodeType*. Each of them is described by a name, a shape (graphical representation) and a behavior (currently a cypher query). The user can define how and how many *NodeType* can be linked together. Moreover, each *NodeType* comes with a set of *PropertyType* which are characterised by a name, a type and a default value. Reasoning in a language is introduced by defining functions over values of node properties.

F. Flexibility

Different requirements notations have been defined to deal with different requirements engineering concerns. For example, Tropos [18] focuses on the agents from early requirements to specifications and implementation, i* [6] on intentions of stakeholders, BIM [19] on Business Intelligence aspects. It is

[4]More information about how the *Language Management* will work can be found at analyticgraph.com/create-your-own-language/

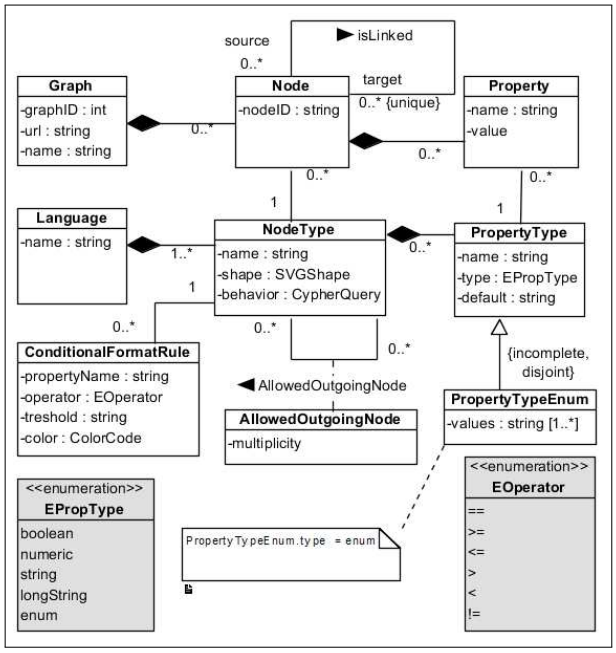

Fig. 4. The Meta-Model of Analytic Graph Language Management Module

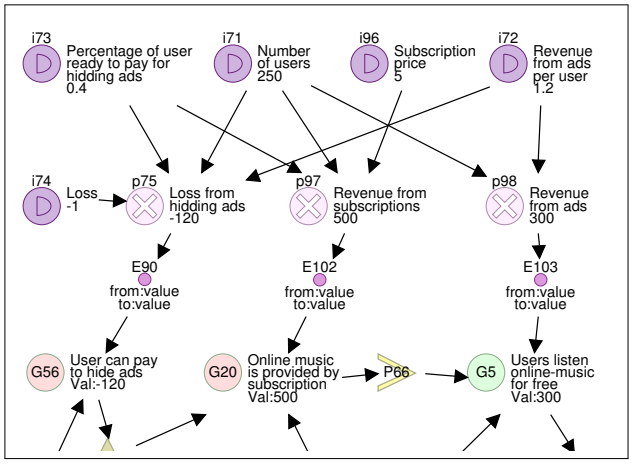

Fig. 5. Example of Mixing 2 Notations in a Model - Application on G5, G20 and G56 from Example Presented in Fig. 2

usually not the case for a GRM tool to allow the building of combined models, that is, of graphs where sub-graphs are models in different languages, and where new types of relationships are used to bridge models of different languages. However, several approaches combining different notations have already suggested. For example, Metzger et al. suggested to distinguish two types of variability by relating orthogonal variability models (OVMs) to feature diagrams [20]; two different languages.

We have made early advances in Analytic Graph to allow users to combine different languages in a same graph. Results coming from computing such sub-graphs are then used in other sub-graphs. For example, Analytic Graph comes with

an Algebra notation capable of performing simple calculations such as addition, multiplication and so on. This notation has been used in the "Music streaming software case" in order to compute value of some goals. This application is depicted in Figure 5. Basic data such as the number of users, the price of subscription and so on, are entered by the user in order to compute value of goals G5, G20 and G56. These basic data are represented by nodes i71, i72, i73... The nodes are then used as inputs of multiplication (represented by cross nodes p75, p97 and p98) for computing advanced values (labelled next to the nodes). Results of the multiplications are next used as inputs of *Transfer nodes*. The latter are special nodes used to extract the value of one property of the incoming node and transfer it in a specified property of the outgoing node (the value of goals in this case). After running behavoir of this graph, one can run the optimization model. This example shows how the Algebra notation can be used with Techne to perform "what-if" analysis. Engineers will be able to check if the optimal solution is still the same if the expected number of users drops by 50%. In the future, users will be able to import data from external sources (a database, a web-service...)

Notice that the present discussion raises some important questions about the validity of using mixed RMLs; namely, does the combination of several different models – designed based on different assumptions and intended for different purposes – make sense? While this question is out of the scope of this paper, it is worth noticing that part of our ongoing work goes into this both theoretical and empirical discussion.

VI. Conclusion and Future Work - Toward Analytic Graph 2.0

Analytic Graph is still a prototype. This means that we are still working on the consolidation of our tool, to ensure it is fully functional and can be used properly by requirements engineers, as intended. Next steps includes the involvement of several users, as a way to collect feedback on Analytic Graph features, and improve/revise them, if relevant. Our future work will also investigate the feasibility and implementation of following improvements:

- Connecting Analytic Graph to external data sources such as databases, webservices;
- Providing classic diagram tool features such as undo, redo, auto diagram layout;
- Completing the user interface for defining custom languages, queries and algorithms;
- Improving real-time collaboration with features allowing users to work on the same model (and not only on a copy).

Despite all limitations of Analytic Graph today, we hope it illustrates interesting directions for the development of next generation GRM tools.

Acknowledgment

This work was partially funded by the FNRS - FRS.

References

[1] S. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing more world knowledge in the requirements specification," in *Proc. 6th international conference on Software Engineering*, 1982, pp. 225–234.

[2] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, and A. Rifaut, "A knowledge representation language for requirements engineering," *Proceedings of the IEEE*, vol. 74, no. 10, pp. 1431–1444, 1986.

[3] J. Hagelstein, "Declarative approach to information systems requirements," *Knowledge-Based Systems*, vol. 1, no. 4, pp. 211–220, 1988.

[4] J. Mylopoulos and A. Borgida, "Telos: Representing knowledge about information systems," *ACM Transactions on Information Systems*, vol. 8, no. 4, pp. 325–362, 1990.

[5] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, pp. 3–50, 1993.

[6] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proc. 3rd International Symposium on Requirements Engineering*, 1997, pp. 226–235.

[7] I. J. Jureta, A. Borgida, N. a. Ernst, and J. Mylopoulos, "Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Proc. International Conference on Requirements Engineering*, 2010.

[8] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," in *Proc. 21st International Conference on Conceptual Modeling (ER'02)*. London, UK: Springer-Verlag, 2002, pp. 167–181.

[9] ——, "Formal Reasoning Techniques for Goal Models," *Journal on Data Semantics I*, vol. 2800, pp. 1–20, 2003.

[10] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models," *Advanced Information Systems Engineering*, vol. 3084/2004, pp. 675–693, 2004. [Online]. Available: http://www.springerlink.com/content/kllxamyqbw61npxq/

[11] D. Bertolini, A. Novikau, A. Susi, and A. Perini, "TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process," Tech. Rep., 2006.

[12] S. Supakkul and L. Chung, "The RE-Tools: A multi-notational requirements modeling toolkit," in *Proc 20th IEEE International Conference on Requirements Engineering Conference (RE)*, 2012, pp. 333–334.

[13] M. Kolp and Y. Wautelet, "DesCARTES Architect: Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems," *Louvain School of Management, Universite catholique de Louvain, Louvain-la-Neuve, Belgium. http://www.sys. cl.ac.be/descartes*, 2007.

[14] "University of Ottawa: jUCMNav. http://softwareengineering.ca/ jucmnav/ (2011)."

[15] V. Englebert and P. Heymans, "Metadone, a flexible metacase to support evolution."

[16] S. Kelly and J.-P. Tolvanen, "Visual domain-specific modeling: Benefits and experiences of using metacase tools," in *International Workshop on Model Engineering, at ECOOP*, vol. 2000. Citeseer, 2000.

[17] J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck, "Product portfolio scope optimization based on features and goals," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 2012, pp. 161–170.

[18] J. F. B. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: the Tropos project," *Information systems*, vol. 27, pp. 365–389, 2002.

[19] J. Horkoff, D. Barone, L. Jiang, E. S. Yu, D. Amyot, A. Borgida, and J. Mylopoulos, "Strategic business modeling: representation and reasoning," *Software & Systems Modeling*, oct 2012.

[20] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis," *... , 2007. RE'07. 15th ...*, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=4384187