Supporting End-Users in Defining Complex Queries on Evolving and Domain-Specific Data Models

Thomas Reschenhofer, Florian Matthes
Technical University of Munich (TUM)
Munich, Germany
Email: reschenh@in.tum.de, matthes@in.tum.de

***Abstract*—To define queries on domain-specific data-structures, end-users have to be familiar with the underlying data model. This is challenging if the data model evolves continuously and through collaborative model management. While visual languages address this issue by providing strong guidance for non-technical end-users, they suffer from a limited expressiveness due to their focus on usability. Therefore, they are not applicable in cases where domain-experts have the need to define complex queries.**

This paper proposes an interactive and textual query editor which focuses on the formulation of complex queries and the familiarization of the end-user with the underlying data model during the query definition process. We demonstrate our approach by this query editor's application in the domain of Enterprise Architecture Management, where tech-savvy end-users need to define complex metrics based on an evolving enterprise architecture model.

I. Introduction

End-user development (EUD) [1] is a means to empower domain-experts to autonomously adapt certain parts of a software to changes which affect their domain [2]. In data-driven business applications, adaption of software often refers to the adaption of the data structures, and the adaption of queries which are based on these data structures [3] and which empower end-users to define dynamic views, metrics, data visualizations, etc.

In recent years, many approaches emerged to support end-users in defining queries in data-driven business applications. For example, visual languages provide guidance [4] for non-technical end-users and ensure that generated queries are syntactically valid as well as consistent with the underlying pre-defined and domain-specific data structure [5], [6], [7], [8]. However, due to the focus on usability, visual language approaches typically suffer from a limited expressiveness compared to textual ones [6], [9]. On the other hand, textual query languages allow technology-savvier domain-experts to formulate more complex expressions as long as they have explicit knowledge about the language's syntax and semantics.

Regardless of the approach, the end-user has to be familiar with the underlying data structure in order to be able to define respective queries [7]. In cases of continuously evolving data models, this is a challenging task, which becomes even harder if the data model is defined collaboratively by a group of domain-experts and not by the user who is performing the query. For example, in the domain of Enterprise Architecture Management (EAM), multiple stakeholders collaboratively and iteratively define a conceptual data model for the enterprise architecture (EA) [10], on the basis of which enterprise architects define EA metrics by formulating respective model-based queries [11], [12]. Because of the evolving nature of the EA model, architects have to familiarize themselves with the data model each time they are defining a metric.

In the paper at hand, we address this issue by proposing an interactive query editor which provides a means to familiarize the end-user with the underlying data model during the query formulation process. For this purpose, the editor displays a relevant excerpt of the current domain-model (cf. [6] and [13]), whereas the relevance is determined by the end-user's input. In contrast to visual language approaches, we focus on a textual query interface which does not impose usability-driven restrictions to the expressiveness of the query. Thereby we target tech-savvy domain-experts which have the need to define complex queries based on a continuously evolving data model. As a concrete example, we demonstrate how our approach can support enterprise architects in defining complex metrics based on a collaboratively managed EA model [12]. In Section II we describe the key features of the query language and technical platform which form the foundation for the interactive query editor. The query editor as this paper's main contribution as well as its evaluation are presented in Sections III and IV respectively, while in Section V we discuss related work and how our approach compares to existing ones.

II. The Model-based Expression Language

Our query editor builds on the foundations of a Web 2.0-based concept for content and model management—Hybrid Wikis [14]—and a corresponding query language named model-based expression language (MxL) [15].

The Hybrid Wiki approach enables end-users to iteratively and collaboratively enrich wiki pages with structure, e.g., types, attributes, and relations. Thereby, wiki pages represent data objects, whereas the wiki page metaphor enables user-friendly and intuitive data management. Through collaboration among editors of those wiki pages as well as data modeling experts, domain-specific models emerge and continuously evolve over time. The Hybrid Wiki approach was already successfully applied in different domains in which data models cannot be defined beforehand, but emerge over time through collaboration, e.g., in Enterprise Architecture Management (EAM) [16] and New Product Development (NPD) [17].

978-1-5090-0252-8/16/$31.00

Based on the evolving data models and an increasing amount of data managed in the Hybrid Wiki system, domain-experts use the model-based expression language (MxL) to formulate queries, e.g., to define metrics, derived attributes, or constraints [18], [15]. MxL is a functional language and is inspired by the Object Constraint Language (OCL) [19]. OCL was already discussed in related research as an obvious choice for a model query language [20]. While the Hybrid Wiki system also provides an intuitive search interface to empower non-technical end-users to find specific information, MxL allows tech-savvy end-users to define more complex queries by applying different kinds of operations, e.g., standard query operations [21] or arithmetic and conditional operations.

In EAM, enterprise architects as representatives for tech-savvy end-users [22] use MxL for the definition of complex metrics [12]. For example, based on an enterprise architecture model capturing *Functional Domains*, their *Business Applications*, and *Databases*, an enterprise architect can define a metric to calculate the heterogeneity of used database systems per domain [12]. With MxL, it can be formally defined as:

```
find 'Functional Domain'.select(
  entropy(Applications.selectMany(Databases)))
```

The *find* construct retrieves all functional domain objects and maps them to an entropy measure [23] based on the databases used in each domain's business applications. In this example, *entropy* is a reusable function which is implemented in MxL. In order to be able to define this metric in an ad-hoc manner, the enterprise architect has to have at least knowledge about the model elements *Functional Domain*, *Applications*, and *Databases*. This can be a challenging task considering that enterprise architecture models usually contain many types, attributes, and relations [13], that those models continuously evolve over time [16], and that metrics can be more complex than the demonstrated example [12].

III. Interactive Query Editor for Domain-Models

To support end-user in the definition of complex queries based on an evolving domain-specific data model, we extended the existing interactive MxL code-editor [15].

The web-based MxL code editor implements useful UI features, e.g., syntax highlighting, error localization, and auto-completion support. With regard to the latter feature, the editor provides a list of hints including applicable operations (e.g., query functions), language constructs (e.g., *find*), and elements from the underlying user-generated data model (e.g., types like *Functional Domain*, attributes like *Function points*, or relations like *Applications*). However, the application of the MxL code editor, e.g., for the definition of EA metrics [12] or data-driven views in NPD [17], revealed that the auto-completion feature is not enough to provide a holistic perspective of the data model. This makes it difficult for end-users to understand how they can navigate through the model, and to determine starting points for the definition of queries. To address this issue, we extended the MxL code editor by an augmented view of the data model. We have chosen the Unified Modeling Language (UML) [24] as form of representation because of this notation's widespread adoption in related EUD research areas [13], [7], [25], and because of MxL's similarity to OCL [19] which is also based on UML.

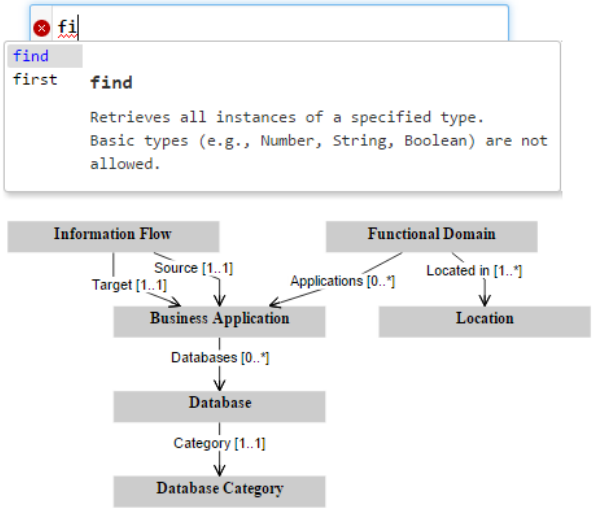

Fig. 1. The query editor with support for auto-completion and with an augmented view of the underlying data model. If the user has not yet entered any input, the model view shows a holistic perspective of the data model.

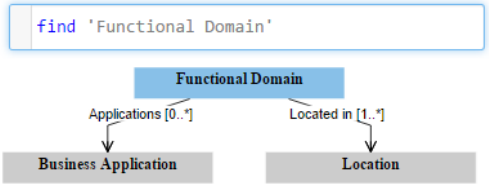

Fig. 2. As soon as the user provides a (partial) expression which refers to a data model element, the augmented model view focuses on related model elements and only shows a local excerpt of the data model.

As long as the user does not provide any input to the query editor, the augmented model view provides an overview of the data model as shown in Figure 1. This means that it shows all domain-specific types as well as relations between them. The attributes of the classes are hidden to reduce the visual complexity. In this way, the model view provides a holistic perspective on the data model and thus a potential indication for the user of how and where to start the query.

If the user enters a MxL expression which refers to an element of the data model, the model view is automatically updated and only shows a related excerpt of the data model. This is enabled by MxL's static type-safety, i.e., the static semantics of MxL expressions can be validated at build-time. This includes the resolution of references to data model elements. Based on this, the code editor creates a local perspective of the model view which only includes directly referenced model elements as well as adjacent elements and sub-elements which improves the navigability through the data model. For example, if the expression refers to a type (cf. Figure 2), the model view also shows all related types as well as attributes of

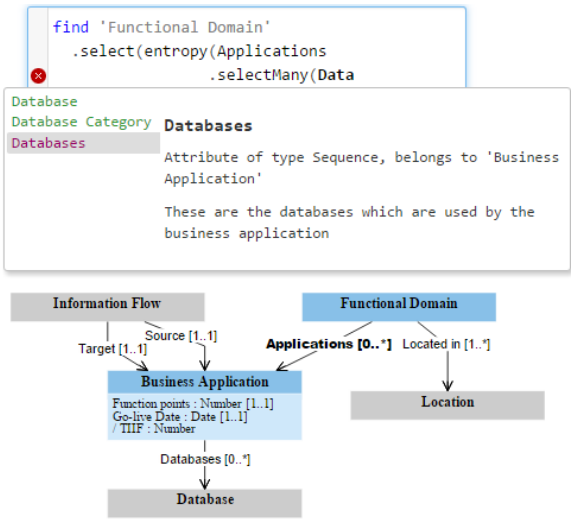

Fig. 3. The model view provides a holistic perspective on the currently relevant excerpt of the data model and thus provides contextual information to the hints provided by the auto-completion mechanism.

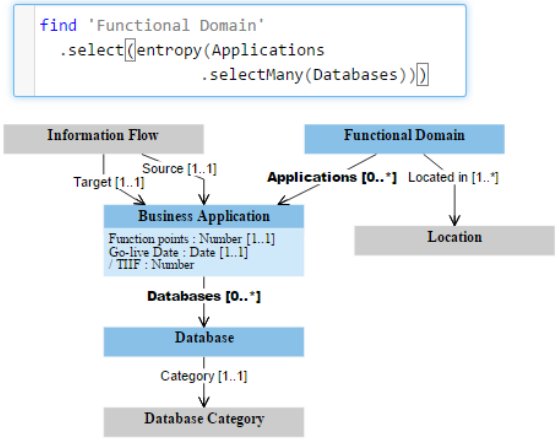

Fig. 4. Model elements which are referred within the current expression are highlighted within the augmented model view.

the directly referenced type (cf. Figure 3). Referenced model elements are visually highlighted, e.g., classes representing directly referenced types are colored blue, while the names of directly referenced attributes and relations are bold (cf. Figure 3 and 4). The augmented model view responds to each change within the code editor, i.e., as soon as the user provides an expression referencing a different set of model elements, the model view is automatically reduced or extended by respective elements. A similar feature is implemented by Störrle [13] for a visual query editor. In this sense, the model view improves the usefulness of the auto-completion hints by enriching them with contextual information, i.e., hints representing model elements can be localized with the data model (cf. Figure 3).

Let us consider again the example of an enterprise architect who wants to define a heterogeneity metric [12] as defined in Section II. Based on the overview of the data model as provided by the model view (cf. Figure 1), the enterprise architect identifies the model elements of interest, namely *Functional Domain*, *Business Application*, and *Database*. Due to the need of calculating the database heterogeneity metric for each functional domain, the enterprise architect enters an MxL expression to gather all entities of type *Functional Domain*. Based on this input, the model view is automatically regenerated to only show the referenced type as well as adjacent types, namely *Business Application* and *Location*. If the type *Functional Domain* would have attributes (which is not the case), they would be shown too. In this way, the model view provides a local perspective of the data model. It is minimal in the sense that model elements are hidden which are not reachable from the current point within the data model, but sufficient enough to determine the model elements which can be referred to within the MxL code editor. As depicted in Figure 3, the model view automatically adapts based on the query provided by the user. Therefore, new model-elements become visible within the view, including its attributes as well as the adjacent types of *Business Application* (*Information Flow* and *Database*). Figure 4 shows the final MxL expression for the heterogeneity metric [12] and the corresponding local model view.

IV. Evaluation

To evaluate the augmented model view, we performed an interactive online experiment with four information systems researchers, three computer science students, one enterprise architect from a Swiss Bank (10,001+ employees), and one enterprise architect from a German IT services provider (5,000 - 10,000 employees). All of them already know the expression language MxL in different levels of detail.

We defined two scenarios for which the experimentees had to define 4 queries each, namely a sales scenario (including customers, products, orders) as well as an EAM scenario (including business applications, functional domains, business supports). The data models and queries for those scenarios are isomorphic, i.e., they only differ in names of types, attributes, and relations. In a first step, test persons had to define four queries for the sales scenario, in a second step they had to formulate four analogous queries for the EAM scenario. Half of the experimentees used the augmented model view only for the first step, the other half only for the second step. After those steps, they were asked about the usefulness, weaknesses, and potential improvements of the augmented model view.

The key findings of the evaluation are as follows:

- Six respondents explicitly stated that the augmented model view accelerates query formulation, particularly if the user is not familiar with the underlying data model, or if the data model is subject to frequent changes.
- At the same time, six respondents expressed the opinion that showing the full data model would be as useful as showing only a local view. However, they admitted that this might be no longer true for larger data models.
- Three respondents noted that the model view should be interactive, e.g., that types could be expanded manually

by clicking on them, or by inserting the model element to the textual query editor on selecting them.
- One respondent mentioned that there might be a scaling issue with the augmented model view for large models.
- Four respondents highlighted that the augmented model view (obviously) only gives an overview over the underlying data model, but that it does not help users to learn the query language and its syntax and semantics.

V. Related Work

There are many approaches to visual query languages and systems based on user-definable ontologies or conceptual data models [5], [6], [9], [13], [26], [27]. Thereby, domain-specific ontologies are a means for defining an end-user-friendly representation of a domain-specific data model as well as a mapping to the physically stored data. They form the basis for visual query user interfaces which empower non-technical end-users to formulate queries in an end-user-friendly way. Due to their focus on usability, visual query languages usually suffer from limited expressiveness [6], [9]. In contrast, the paper at hand focuses on a high expressiveness and targets different use-cases, e.g., the definition of complex metrics. Nevertheless, there are visual design principles which were inspired by visual query editors and adopted to our approach, e.g., the responsive view of the user-model by Störrle [13].

Valencia-García et al. [7] propose a natural language and a corresponding query editor named *OWLPath*. A query expressed in natural language is translated to SPARQL as the de-facto standard for querying ontologies. Similar to the query editor as described in our work, *OWLPath* provides helpful features which improve the usability of the editor, e.g., grammar checking and auto completion. However, while using a natural language lowers the hurdle for non-technical end-user to define queries compared to formal languages, there is still the need to familiarize the end-user with the underlying data model. Therefore, we think that applying our approach of automatically generating a relevant excerpt of the data model or ontology based on the current textual user input would be a valuable addition to OWLPath.

Cunha et al. [28], [8] describe a spreadsheet-based query approach. Thereby, end-users define the structure of a spreadsheet by using ClassSheets [29]. They choose ClassSheet models to present the data model which has to be queried instead of ontologies, which allows end-users to formulate queries in their preferred environment—namely within the spreadsheet [30]. Although they follow a visual language approach, the integration into spreadsheets and thus the possibility to apply further spreadsheet formulas to the query results potentially enables more complex calculations. Nevertheless, they do not discuss the challenge of familiarizing the user with the underlying ClassSheet model during the query formulation process. Again, integrating the approach by Cunha et al. with the approach of the paper at hand—which addresses the familiarization issue—might yield interesting results.

VI. Conclusion

This paper presents our work on improving the usability of a textual query editor for end-users with a focus on the user's familiarization with the underlying data model during the query formulation process. It is based on previous research on Hybrid Wikis [14] for collaborative information management on the one hand, and on the model-based expression language MxL [15] as functional query language on the other hand. The improvements of the corresponding query editor by the integration of an augmented model view are motivated by the application of Hybrid Wikis and MxL in different contexts, e.g., EAM [12] and NPD [17].

The code editor as proposed in this work integrates an augmented model view which provides a holistic perspective to the data model which has to be queried. The model view automatically responds to the user's query input in the sense that it only shows model parts which are currently relevant for the user. The proposed code editor is particularly helpful in domains with continuously evolving data models, e.g., EAM. It is not only applicable in other domains, but also to other query languages, provided that the language's static semantics can be analyzed and thus references to data model elements can be resolved at build-time.

Based on the findings of a first evaluation, our work on supporting the familiarization of the end-user with the data model during the query formulation process requires further improvements and elaboration. Particularly an application of the interactive code editor in a practical environment and a corresponding evaluation would encourage our claim of improving the usability of query editors. To this end, we are establishing a network of industry partners which plan to prototypically implement the Hybrid Wiki system, MxL, and the query editor. In this context, we plan to conduct empirical studies on the usefulness and usability of the query editor.

As revealed by the evaluation as described in Section IV, there are further conceptual and technical design aspects which should be improved. For example, although the model view helps users to familiarize themselves with the underlying data model, formulating complex queries is still a difficult task. Consequently, there still has to be research about how to make the definition of complex queries easier, particularly for non-technical end-users. On a more technical note, by strengthening the interrelation between the auto-completion feature and the model view, users could be enabled to insert references to model elements by clicking on the respective element in the model view. Another suggestion by a respondent was to make the model view manually explorable, i.e., to allow the manual and explicit expansion of a local data model view by clicking on certain model elements. Moreover, showing and highlighting recent changes of the data model within the data view would improve the awareness of which model elements changed in which way. Finally, as rightfully mentioned by one of the respondents, there still has to be research about large-scale data models and the performance of the augmented model view with this kind of models.

REFERENCES

[1] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*, ser. Human-Computer Interaction Series, H. Lieberman, F. Paternò, M. Klann, and V. Wulf, Eds. Springer, 2006, pp. 1–8.

[2] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, "Meta-Design: A Manifesto for End-User Development," *Communications of the ACM*, vol. 47, no. 9, pp. 33–37, 2004.

[3] M. Burnett and B. A. Myers, "Future of End-User Software Engineering: Beyond the Silos," *Proceedings of the International Conference on Software Engineering*, pp. 201–211, 2014.

[4] J. Hvorecký, M. Drlík, and M. Munk, "The Effect of Visual Query Languages on the Improvement of Information Retrieval Skills," *Procedia-Social and Behavioral Sciences*, vol. 2, no. 2, pp. 717–723, 2010.

[5] M. Giese, D. Calvanese, P. Haase, I. Horrocks, Y. Ioannidis, H. Kllapi, M. Koubarakis, M. Lenzerini, R. Möller, and M. Rodriguez-Muro, "Scalable End-user Access to Big Data," *Big Data Computing*, pp. 205–245, 2013.

[6] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jimenez-Ruiz, M. Giese, and I. Horrocks, "Ontology-Based Visual Query Formulation: An Industry Experience," in *Advances in Visual Computing*. Springer, 2015, pp. 842–854.

[7] R. Valencia-García, F. García-Sánchez, D. Castellanos-Nieves, and J. T. Fernández-Breis, "OWLPath: An OWL Ontology-guided Query Editor," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 41, no. 1, pp. 121–136, 2011.

[8] J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva, "Graphical Querying of Model-Driven Spreadsheets," *Proceedings of the International Conference on Human-Computer Interaction*, 2014.

[9] F. Haag, S. Lohmann, S. Siek, and T. Ertl, "Visual Querying of Linked Data with QueryVOWL," *Joint Proceedings of SumPre*, pp. 2014–2015, 2015.

[10] S. Buckl, F. Matthes, C. Neubert, and C. M. Schweda, "A Lightweight Approach to Enterprise Architecture Modeling and Documentation," in *Information Systems Evolution*. Springer, 2010, pp. 136–149.

[11] F. Ahlemann, E. Stettiner, M. Messerschmidt, and C. Legner, *Strategic Enterprise Architecture Management*. Springer, 2012.

[12] A. W. Schneider, T. Reschenhofer, A. Schütz, and F. Matthes, "Empirical Results for Application Landscape Complexity," *Proceedings of the Hawaii International Conference on System Sciences*, pp. 4079–4088, 2015.

[13] H. Störrle, "VMQL: A Visual Language for Ad-Hoc Model Querying," *Journal of Visual Languages & Computing*, vol. 22, no. 1, pp. 3–29, 2011.

[14] F. Matthes, C. Neubert, and A. Steinhoff, "Hybrid Wikis: Empowering Users to Collaboratively Structure Information," *Proceedings of the International Conference on Software and Data Technologies*, pp. 250–259, 2011.

[15] T. Reschenhofer, I. Monahov, and F. Matthes, "Type-Safety in EA Model Analysis," *Proceedings of the International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pp. 87–94, 2014.

[16] F. Matthes and C. Neubert, "Wiki4EAM - Using Hybrid Wikis for Enterprise Architecture Management," *Proceedings of the International Symposium on Wikis and Open Collaboration*, p. 226, 2011.

[17] S. Rehm, T. Reschenhofer, and K. Shumaiev, "IS Design Principles for Empowering Domain Experts in Innovation: Findings From Three Case Studies," *Proceedings of the International Conference on Information Systems*, 2014.

[18] I. Monahov, T. Reschenhofer, and F. Matthes, "Design and Prototypical Implementation of a Language Empowering Business Users to Define Key Performance Indicators for Enterprise Architecture Management," *Proceedings of the Trends in Enterprise Architecture Research Workshop*, 2013.

[19] Object Management Group, "Object Constraint Language (OCL)," 2014. [Online]. Available: http://www.omg.org/spec/OCL/2.4

[20] H. Störrle, "Improving the Usability of OCL as an Ad-hoc Model Querying Language," *Proceedings of the International Workshop on OCL, Model Constraint and Query Languages*, 2013.

[21] A. Heijlsberg and M. Torgersen, "Standard Query Operators Overview," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb397896.aspx

[22] C. Strano and Q. Rehmani, "The Role of the Enterprise Architect," *Information Systems and E-Business Management*, vol. 5, no. 4, pp. 379–396, 2007.

[23] R. Lagerström, C. Y. Baldwin, A. D. Maccormack, and S. Aier, "Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case," *Harvard Business School Working Paper*, no. 13-103, 2013.

[24] Object Management Group, "Unified Modeling Language (UML)," 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5

[25] F. Hermans, M. Pinzger, and A. van Deursen, "Automatically Extracting Class Diagrams from Spreadsheets," *Proceedings of the European Conference on Object-Oriented Programming*, pp. 52–75, 2010.

[26] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou, "Ontology Visualization Methods — A Survey," *ACM Computing Surveys (CSUR)*, vol. 39, no. 4, p. 10, 2007.

[27] P. Delfmann, D. Breuker, M. Matzner, and J. Becker, "Supporting Information Systems Analysis Through Conceptual Model Query–The Diagramed Model Query Language (DMQL)," *Communications of the Association for Information Systems*, vol. 37, 2015.

[28] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Querying Model-Driven Spreadsheets," *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, 2013.

[29] M. Erwig and G. Engels, "ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications," *Proceedings of the International Conference on Automated Software Engineering*, pp. 124–133, 2005.

[30] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Embedding Model-Driven Spreadsheet Queries in Spreadsheet Systems," *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, 2014.