IncQuery Server for Teamwork Cloud

Scalable Query Evaluation over Collaborative Model Repositories

Ábel Hegedüs[1], Gábor Bergmann[1], Csaba Debreceni[1], Ákos Horváth[1], Péter Lunk[1], Ákos Menyhért[1], István Papp[1], Dániel Varró[1], Tomas Vileiniskis[2], István Ráth[1]
[1]IncQuery Labs Ltd, Hungary
firstname.lastname@incquerylabs.com
[2]No Magic Inc, Texas, USA
firstname.lastname@nomagic.com

ABSTRACT

Large-scale cyber-physical systems are co-engineered, especially in safety-critical industries, by various specialists within an organization and, increasingly, across organizations. The collaborative aspect of the process is facilitated by hosting engineering artifacts in model repositories. In order to validate the adherence to design rules, perform change impact analysis across projects, generate reports etc., engineers specify model queries and evaluate them using query engines, traditionally available in client modeling tools.

In this paper we introduce IncQuery Server for Teamwork Cloud (IQS4TWC), a standalone middleware service that connects to Teamwork Cloud model repositories, and builds on Viatra Query to provide fast querying over their content. The new server-side solution provides advanced features including single-model ad-hoc queries as well as repository-wide change impact analysis (correlating projects across branches and revisions); access to version snapshots as well as queries on the latest state; and a range of performance fine-tuning options (such as elasticity and in-memory indexes) to achieve high scalability.

Screencast demo: https://goo.gl/rScgco

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**;

KEYWORDS

model-driven engineering, co-engineering, collaborative modeling, model query

ACM Reference Format:
Ábel Hegedüs, Gábor Bergmann, Csaba Debreceni, Ákos Horváth, Péter Lunk, Ákos Menyhért, István Papp, Dániel Varró, Tomas Vileiniskis, István Ráth. 2018. IncQuery Server for Teamwork Cloud: Scalable Query Evaluation over Collaborative Model Repositories. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3270112.3270125

1 INTRODUCTION

A growing number of industries embrace model-based systems engineering (MBSE) methodologies [31], or at least integrate pieces of these workflows into their conventional development process.

Model queries [30] form the underpinning of various technologies that are essential in state-of-the-art modeling techniques: validation of design rules and well-formedness constraints; generating reports, presenting views and performing ad-hoc analyses on the system under design, such as finding dependencies between various components or assessing the impact of modifications; controlling the application of model-to-model transformation and code generation; and simulation / execution according to domain-specific behaviour semantics. Therefore efficient and expressive support for model queries is of great importance.

Complex cyber-physical systems, especially in safety-critical industries, are *co-engineered* by heterogeneous teams and various domain specialists. Engineers may collaborate over model files hosted in file-based version control systems (VCS), just like software developers do with source code. Alternatively, work may be organized around advanced model-aware collaboration servers; several industry-ready *model repositories* have emerged, including both open source (e.g. CDO [24], EMFStore [26]) and commercial (e.g. Teamwork Cloud [13], Modelio Constellation [19], GenMyModel [1]) solutions. Such repositories may allow for sharing, collaboratively editing, reviewing models, as well as reusing model fragments and component designs across an organization.

Problem statement Software developers benefit from server-side analysis capabilities (continuous integration, automated testing, static analyis), which can exploit more information (e.g. extended test suites) than what is available on the client computer of any single developer. However, such project-wide or organization-wide analyses are rarely supported in MBSE tools. Next, we motivate by various user stories why this is a problem.

In co-engineering processes, team members may gain insight into the shared models using various ad-hoc or tool-integrated analysis techniques. Design rule validation results or requirement-driven status reports driven by model queries may inform *specialist engineers* on whether the model is ready for them to work with. Such reports also help *project managers* track the progress of the team towards a final, consistent model. Moreover, dependency queries may help *system engineers* in discovering the organization-wide usage of given component models or reusable fragments, so that a *change impact analysis* can be carried out before upgrading / modifying said component. Finally, *model reviewers* may investigate

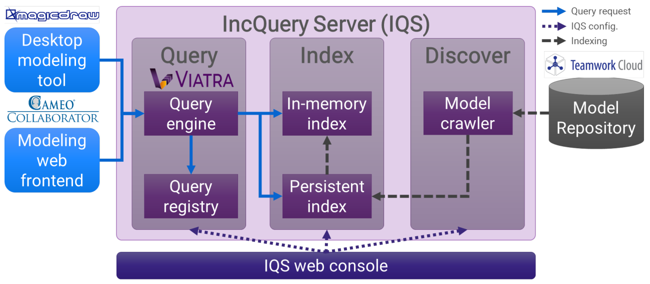


Figure 1: IncQuery Server Architecture

the internal structure and connections of a model using model views defined by custom, ad-hoc queries.

There are several model query technologies built on top of client-side modeling environments: e.g. IncQuery for MagicDraw [15] provides end-user model validation features, by adapting the query engine Viatra Query [30] for MagicDraw [12] via the connector V4MD [21]. Unfortunately, queries over the contents of model repositories are not universally available. If specialists and managers can only access model query results by downloading and processing all the models in question using a client-side modeling tool, this puts an unnecessary burden on the whole process. Similarly, no engineers are willing to or able to check out all models from the repository to perform change impact analysis.

Target Audience This paper, as well as the software presented therein, is primarily aimed at tool developers building collaborative extensions over the modeling tool MagicDraw. We are aware of ongoing efforts for the development of such tools within *No Magic Inc* (Cameo Collaborator), as well as the OpenMBEE [18] community led by *NASA JPL*. However, end-users will also indirectly benefit from enhanced query capabilities.

Contributions In this paper we introduce IncQuery Server for Teamwork Cloud (IQS4TWC), a cloud-deployable server-side query middleware service that is integrated between collaborative modeling frontends and model management backends as a query result provider. The proposed solution indexes models hosted in a Teamwork Cloud model repository in order to extend it by expressive and efficient model querying capabilities. Based on Viatra 2.0 technology, IQS4TWC is capable of producing query results that could only be obtained by a traditional modeling client after downloading each model and evaluating the query on it separately. Moreover, it correlates disparate models with each other for purposes of cross-repository reporting, such as change impact analysis. This is achieved using query-driven soft links [11], i.e. identifiers are used as "foreign keys" to relate elements across models.

2 INCQUERY SERVER ARCHITECTURE

IQS4TWC achieves all of the above by adopting an architecture (Fig. 1) comprising of the following core components:

Server-side model query engine The above functionality is provided by *evaluating model queries* on a specialized *query server*. The queries are interpreted on one or more models, as determined by the *query scope*. The scope may vary temporally: it may be defined for a given *version snapshot*, or *tracking* the latest state (so that re-evaluating a query would deliver updated results after a new revision is created), or even the entire version history. Spatially, the scope may include (a single branch of) a single modeling project, or the breadth of an entire repository. Under the hood, IQS4TWC uses the core engine of Viatra Query to efficiently evaluate such graph queries using one of several backends.

Query registry User-defined queries in the Viatra Query Language (VQL) syntax can be loaded into the server-side query registry. Queries selected from this registry can then be initialized on a given scope. Queries can either be uploaded directly into the registry, or alternatively, extracted from well-formedness and design rule constraints defined within models themselves.

In-memory and persistent model index Models are not directly loaded into IQS4TWC; they are rather represented by *index structures* supporting efficient lookups. The query engine is adapted to consume these index structures instead of EMF [25] models. Once constructed, indexes (independently for each individual model) can either be kept *in memory* for quick access and fast query evaluation, or persisted to disk using various storage backends (such as Cassandra [23]). The latter is especially practical for keeping a historical archive of important snapshots (e.g. milestone versions) against which queries are executed occasionally, without taking up space in memory. For saving space, in-memory indexes can be configured to load only those model types and features that are relevant for the initialized queries (*type filtering*).

Model crawler The model crawling and index updating pipeline extracts content from the model repository (Teamwork Cloud server or cluster). The results are filtered (e.g. according to relevant features), formatted and then inserted into the index structures mentioned above. In case of Teamwork Cloud, we have found that the best performance is attained by invoking a combination of several API endpoints (REST and Java RPC).

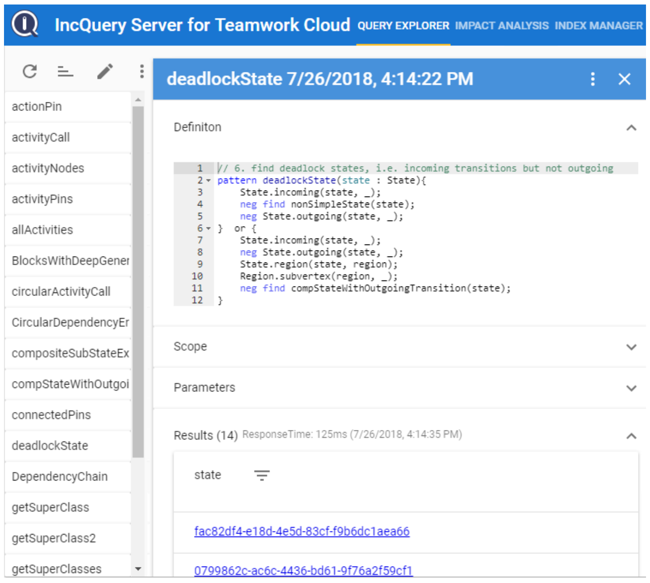
Service interface Since the querying capability is expected to be mostly used within advanced collaborative modeling application front-ends, IQS4TWC exposes a remote API through which the following main features are accessible: (a) query execution; (b) query registry management; (c) index management.

Web Console While end user modelers are expected to access IQS4TWC functionality through integrations in front-end modeling tools including Cameo Collaborator, we also provide a simple web-based UI that exposes the most important functionality for developers and administrators. Although the focus is on management features, it is also possible to directly select queries from the query registry (or upload new custom queries) for execution. Fig. 2 shows a validation query in the syntax of Viatra Query that identifies deadlocks; along with the obtained query results in tabular form (model elements are hyperlinked to their representation on the Teamwork Cloud repository).

3 ADVANCED FEATURES

The following additional capabilities are currently provided:

Cross-model dependency analysis Modeling front-ends perform change-impact analysis on models to learn how other elements are affected by a given change. IQS4TWC is capable of extending this functionality by discovering (direct and transitive) cross-model dependencies, which is necessary when model fragments and



```
// 6. find deadlock states, i.e. incoming transitions but not outgoing
pattern deadlockState(state : State){
    State.incoming(state, _);
    neg find nonSimpleState(state);
    neg State.outgoing(state, _);
} or {
    State.incoming(state, _);
    neg State.outgoing(state, _);
    State.region(state, region);
    Region.subvertex(region, _);
    neg find compStateWithOutgoingTransition(state);
}
```

Results (14) ResponseTime: 125ms (7/26/2018, 4:14:35 PM)

state

fac82df4-e18d-4e5d-83cf-f9b6dc1aea66

0799862c-ac6c-4436-bd61-9f76a2f59cf1

Figure 2: Web Console Screenshot

components are reused across an organization. This is an example of the repository-wide, cross-model, cross-version query capability, as dependencies from any revision and branch of any hosted model may be relevant for the results. Note that it is currently realized using a combination of in-memory index structures and persistent lookup tables pre-processed at model crawling time.

Elastic distributed processing Repository crawling, model indexing and query evaluation processes are implemented as asynchnronous microservices based on Eclipse Vert.x [27], thus their performance can be adjusted by varying the number of parallel instances ("verticles") and physical machines. In particular, we have found that it is highly beneficial to fine-tune the crawling process to the specific performance characteristics of the given model repository, which may itself be a cluster of several Teamwork Cloud nodes, served by an additional cluster as storage backend. IQS4TWC component *verticles* are containerized (via Docker [6]) and cloud-ready, especially fitting for model repository clusters that are themselves hosted on public cloud infrastructure.

Stream-based model crawling Repository crawling is treated as a data stream processing task. Thanks to the per-element model access APIs of Teamwork Cloud, this stream can be executed efficiently, i.e. without manifesting the entire crawled model in memory during the crawling process. This is most relevant for extremely large models that may not easily fit into the memory of any processing node. (In those cases, of course, in-memory indexes are only possible with very selective type filtering.)

Differential crawling Industrial experience with large-scale models shows that crawling an entire model from a repository may take considerable time. Fortunately, there is no need for this if most of the content is already indexed, and only a difference has to be retrieved. Since Teamwork Cloud model repositories provide information on the difference between subsequent revisions of a model, IncQuery Server will only crawl the changed model elements to update its index to the newest revision.

"Universal" queries Using Viatra Query on the server, queries become "Univeral" or "Isomorphic" (in the sense these words are frequently used for Javascript [14]), i.e. the same query code (Viatra Query) can be evaluated in two different environments: embedded within a traditional client-side modeling tool (IncQuery for MagicDraw) with a model loaded in memory, or in IQS4TWC where model elements are not actually manifested. This facilitates the development of query-based model processing functionality that can run either within a desktop client or a server-based front-end modeling environment, with query results supplied fully or partially by IQS4TWC. In particular, as an alternative source of queries, IQS4TWC can load and apply well-formedness and design rule constraints defined in models, exactly as they are handled by IncQuery for MagicDraw in the desktop modeling client.

Presentation and tracing of result elements While the results of server-based analyses may be model elements, it is not straightforward how to present them to the user. The web console of IQS4TWC is customized so that UML elements are hyperlinked back to their representation on the model repository, so that the developer can immediately inspect them in context.

4 INITIAL EVALUATION AND DISCUSSION

4.1 Initial Evaluation

For a rudimentary performance assessment, we have adapted an existing MagicDraw-based model query benchmark, which is available open source [16]. The benchmark task is to evaluate a given set of model queries provided by *NASA JPL*. The benchmark input is organic (rather than synthetic) in the sense that it derives from OpenMBEE's *Thirty-meter-telescope* (TMT) model [18], a publicly available, *large* SysML model of an astronomical device. The benchmark is run on increasing workloads (from approx. 300K to roughly 1.26M elements), obtained by creating up to 4 additional identical copies of a certain part of TMT (approx. 240K elements), the *work packages*. This scaling might be taken as a proxy for having up to 5 *independent* projects (each of which is large and complex) in an organizational model repository. The structurally complex queries [16] have relatively sparse results (as common for constraint validation); even on the largest model instance, there are only 1303 elements in the union of the result sets of all individual queries (as computed by the `allBenchmarkedQueries` query of the suite [16]).

To adapt this benchmark to the server-side scenario, we have prepared a Teamwork Cloud model repository hosting the benchmark input model, and executed a workflow consisting of (a) instructing IQS4TWC to crawl the repository and build an index of these models, and then (b) retrieving the results of the model queries (more specifically, the result of `allBenchmarkedQueries`) from IQS4TWC. These steps can be performed on increasingly larger input models and with a range of configuration options (cluster size, index persistence, etc.) to measure multiple metrics (execution time, memory and disk usage). However, for brevity, here we are reporting only on the scalability of execution times, in case of two configurations: one where persistent indexes are simply stored in files local to IQS4TWC servers, and one where they are hosted in a separate Apache Cassandra [23] database cluster (these configurations are

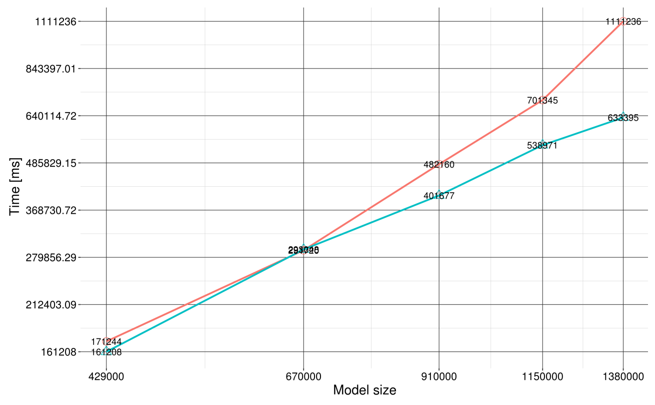


Figure 3: Model crawl time vs. # of model elements Cyan: file-based, Red: Cassandra-based persistence

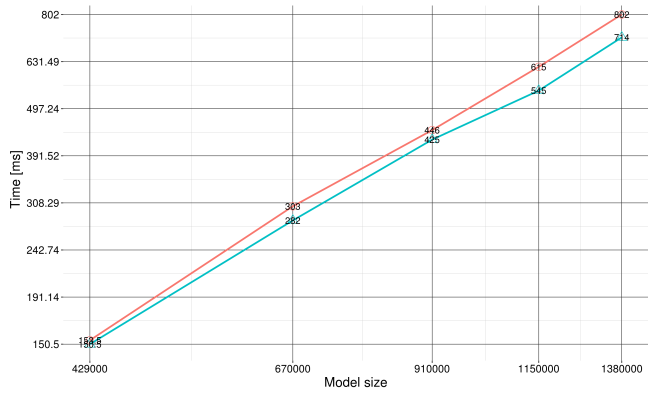


Figure 4: Query evaluation time vs. # of model elements Cyan: file-based, Red: Cassandra-based persistence

represented on the charts with cyan resp. red lines). In both cases, an in-memory index was also built.

Measurements were carried out in the Amazon EC2 public cloud, where three `m4.xlarge` machine instances formed the Teamwork Cloud cluster, an additional three `m4.large` instances hosted the Cassandra cluster serving as the storage backend of Teamwork Cloud, and three `m4.2xlarge` nodes hosted IQS4TWC. In the runs with Cassandra-based persistent indexes, three more `m4.xlarge` instances provided a second Cassandra cluster for this purpose.

The total time required to crawl the repository and build the persistent and the in-memory model index at IQS4TWC is depicted on Fig. 3; the average crawling speed turned out to be approx. 2180 elements / sec in the case of file-based persistence, and significantly slower if Cassandra was used. Afterwards, the total set of model queries are requested on the entirety of the freshly indexed model; the execution times depicted by Fig. 4 include the sending of the request to IQS4TWC, the from-scratch evaluation of the queries (using the search-based query backend of Viatra Query), and the retrieval of the query results.

4.2 Discussion

Crawling time Crawling time depends primarily on the number of elements and negligibly on the depth of containment hierarchy as Teamwork Cloud provides API to access the full content of elements using their identifiers. Identifiers are attainable from the list of contained child elements. Measurements (not shown) have confirmed that additional variables (e.g. distribution of cross-references) have no impact on crawling time.

Indexing The *persistent index* is split along revisions and in case of Cassandra, additional splitting is used along types, references and attributes. Replication, in case of Cassandra, is fully transparent and it depends on cluster configuration. As of now, we felt no need for "Big Data" *horizontal sharding*, as single revisions are expected to fit into the memory of a client modeling tool. The *in-memory index* uses revision-based split as well, but there is no need for replication in this case as the in-memory index is only temporary. Queries can use already indexed revisions only: a model is excluded as long as its indexing / loading is ongoing.

Performance factors Dependency between model properties / metrics and evaluation cost of queries is discussed in [22] hence the investigation of their correlation is out of scope for this paper.

5 RELATED WORK

IQS4TWC has a different focus than either model repositories (e.g. Teamwork Cloud, as well as CDO [24], EMFStore [26]), or web-based model editors and collaboration servers (e.g. Cameo Collaborator, as well as GenMyModel [1], MetaEdit+ [29], Obeo Designer Team [20]). We consider efficient and expressive model querying as the primary goal of IQS4TWC, relegating the persistence of models themselves to the underlying repository backend, and the user-facing model editing and management to a (desktop or web-based) modeling front-end.

Compared to the model indexer Eclipse Hawk [7], which indexes models stored in VCSs and the Modelio Constellation model repository, IQS4TWC provides these key distinguishing features: (a) queries over older versions of models (Hawk currently only indexes the most recent version); (b) the availability of filtered, compact in-memory indexes for fast query evaluation (as opposed to the disk-only graph database backends of Hawk); (c) elastic streaming repository crawling (Hawk has to parse the entire model into the memory of a single machine in the process of indexing it). However, the VCS crawling and indexing capability of Hawk may prove useful in the future as an additional data source to IQS4TWC; the underlying query engine Viatra Query can already execute queries over Hawk indexes using the adapter HawkScope [28].

NeoEMF [3] allows EMF models to be transparently persisted in various NoSQL storage backends. While it only provides access trough standard EMF model manipulation API, Mogwaï [4] adds the capability of executing OCL queries (more efficiently), by translating them to database queries that the storage backend can undestand and execute with optimizations. Recent developments in EMF Views [2] allow *query mediation* of OCL queries between a heterogeneous collection of backend models (local files or NeoEMF databases): this means an (optimized) translation of the user-specified query into smaller queries directly handled by each storage backend. However, both Mogwaï and EMF Views are client plug-ins; server-side model repository querying is not provided. See [8] for an in-detail comparison of the design and performance of Mogwaï, Hawk and CDO.

Syndeia Cloud [17] is a middleware that connects to remote model repositories; it also provides storage for explicit inter-model traceability relationships between the models stored in the former. For elementary queries (e.g. following a single traceability link), it performs query mediation across all these data sources, i.e. forwards each requests to the appropriate storage where the information is to be found. Complex graph queries, however, are only supported by exporting the entire *total system model* into a dedicated graph database, which can then be directly queried by the user.

6 FUTURE WORK

Further features currently under development include:

Access control Collaborative model repositories have to be configured to restrict read access in corporate environments where subcontractors have limited clearance of intellectual property, or export regulations prevent certain artifacts from being divulged to external company offices. In these cases, the query server must also reflect such access restrictions to guarantee that it does not constitute a security leak. IQS4TWC supports the coarse-grained (project-level) access control scheme of Teamwork Cloud; efforts to transparently integrate Teamwork Cloud authorization are ongoing. Alternative approaches with fine-grained (e.g. object-level) access control [5] are also on the horizon.

Live queries and incrementality Taking advantage of the incremental query capabilities in Viatra Query, it will soon be possible to initialize *live (standing) queries* against the latest revisions of models. The results of live queries are then incrementally maintained upon the appearance of newer revisions, based on the existing delta crawling (see above) feature.

Revision histories and temporal queries Some use cases might require queries over the entire version history, not just notable milestones; e.g. in order to find all past revisions of a project that contain certain kinds of bugs. Naively building indexes separately for all such versions would multiply storage requirements. As models typically experience small changes across revisions, most of these index entries would be duplicates across a large version range. A more advanced temporal index representation (via e.g. the ChronoDB/ChronoGraph/ChronoSphere [9] stack or Greycat [10]) would eliminate this redundancy.

ACKNOWLEDGMENTS

We would like to thank Enrique Krajmalnik, Andrius Armonas and Vygantas Gedgaugas from No Magic for supporting our collaborative work and helping with testing and evaluations. We are grateful to the OpenMBEE community [18], and in particular to Robert Karban and Ivan Gomes from the *NASA JPL*, for providing a lot of valuable ideas and feedback regarding IQS4TWC features and benchmarking [16]. Finally, we would like to thank Viatra and IQS4TWC contributors: Zoltán Ujhelyi, Balázs Bea, Zoltán Béres, Bence Beöthy, Krisztián Mayer, Tamás Németh and Péter Varga.

REFERENCES

[1] Axellience. [n. d.]. GenMyModel. http://www.genmymodel.com.

[2] Hugo Bruneliere, Florent Marchand de Kerchove, Gwendal Daniel, and Jordi Cabot. 2018. Towards Scalable Model Views on Heterogeneous Model Resources. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18)*.

[3] Gwendal Daniel, Gerson Sunyé, Amine Benelallam, Massimo Tisi, Yoann Vernageau, Abel Gomez, and Jordi CABOT. 2017. NeoEMF: A Multi-database Model Persistence Framework for Very Large Models. *Science of Computer Programming* (Aug. 2017). https://doi.org/10.1016/j.scico.2017.08.002

[4] Gwendal Daniel, Gerson Sunyé, and Jordi Cabot. 2018. Scalable Queries and Model Transformations with the Mogwaï Tool. In *Theory and Practice of Model Transformation*. Springer International Publishing, Cham, 175–183.

[5] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. 2017. Enforcing fine-grained access control for secure collaborative modelling using bidirectional transformations. *Software & Systems Modeling* (21 Nov 2017). https://doi.org/10.1007/s10270-017-0631-8

[6] Docker Inc. [n. d.]. Docker. https://www.docker.com.

[7] Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S Kolovos, Marcos Aurelio Almeida da Silva, Antonin Abherve, and Alessandra Bagnato. 2016. Integration of a Graph-based Model Indexer in Commercial Modelling Tools. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)*. ACM, New York, NY, USA, 340–350. https://doi.org/10.1145/2976767.2976809

[8] Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S. Kolovos, Ran Wei, and Richard F. Paige. 2017. Stress-testing remote model querying APIs for relational and graph-based stores. *Software & Systems Modeling* (30 Jun 2017). https://doi.org/10.1007/s10270-017-0606-9

[9] Martin Haeusler, Thomas Trojer, Johannes Kessler, Matthias Farwick, Emmanuel Nowakowski, and Ruth Breu. 2018. Combining Versioning and Metamodel Evolution in the ChronoSphere Model Repository. In *SOFSEM 2018: Theory and Practice of Computer Science*. Springer International Publishing, Cham, 153–167.

[10] Thomas Hartmann, Francois FOUQUET, Matthieu Jimenez, Romain Rouvoy, and Yves Le Traon. 2017. Analyzing Complex Data in Motion at Scale with Temporal Graphs. In *The 29th International Conference on Software Engineering & Knowledge Engineering (SEKE'17)*. KSI Research, Pittsburgh, United States, 6. https://hal.inria.fr/hal-01511636

[11] Ábel Hegedüs, Ákos Horváth, István Ráth, Rodrigo Rizzi Starr, and Dániel Varró. 2016. Query-driven soft traceability links for models. *Software & Systems Modeling* 15, 3 (01 Jul 2016), 733–756. https://doi.org/10.1007/s10270-014-0436-y

[12] No Magic Inc. [n. d.]. MagicDraw. https://www.nomagic.com/products/magicdraw

[13] No Magic Inc. [n. d.]. Teamwork Cloud. https://www.nomagic.com/products/teamwork-cloud

[14] Michael Jackson. [n. d.]. Universal Javascript. Retrieved July 19, 2018 from https://cdb.reacttraining.com/universal-javascript-4761051b7ae9

[15] IncQuery Labs. [n. d.]. IncQuery for MagicDraw. https://incquerylabs.com/incquery

[16] IncQuery Labs. 2017. The MagicDraw VIATRA Query performance benchmark. https://github.com/IncQueryLabs/magicdraw-viatra-benchmark.

[17] Intercax LLC. [n. d.]. Syndeia. http://intercax.com/products/syndeia

[18] MBEE [n. d.]. http://www.openmbee.org/.

[19] Modeliosoft. [n. d.]. Modelio Constellation. https://www.modeliosoft.com/en/products/modelio-constellation.html

[20] Obeo. [n. d.]. Obeo Designer Team. https://www.obeodesigner.com/en/collaborative-features.

[21] The VIATRA Project. [n. d.]. V4MD. https://github.com/viatra/v4md

[22] Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. 2017. The Train Benchmark: cross-technology performance evaluation of continuous model queries. *Software & Systems Modeling* (17 Jan 2017). https://doi.org/10.1007/s10270-016-0571-8

[23] The Apache Foundation. [n. d.]. Cassandra. http://cassandra.apache.org

[24] The Eclipse Foundation. [n. d.]. CDO. http://www.eclipse.org/cdo.

[25] The Eclipse Foundation. [n. d.]. Eclipse Modeling Framework. http://www.eclipse.org/emf/.

[26] The Eclipse Foundation. [n. d.]. EMFStore. http://www.eclipse.org/emfstore.

[27] The Eclipse Foundation. [n. d.]. Vert.x. https://vertx.io

[28] The MONDO Project. 2015. Deliverable 5.4: Heterogeneous Model Management Framework. https://tinyurl.com/mondo-d54

[29] Juha-Pekka Tolvanen. 2007. MetaEdit+: Domain-Specific Modeling and Product Generation Environment. In *Software Product Lines, 11th Int. Conf. SPLC 2007, Kyoto, Japan*. 145–146.

[30] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. 2016. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Software & Systems Modeling* 15, 3 (01 Jul 2016), 609–629. https://doi.org/10.1007/s10270-016-0530-4

[31] Jon Whittle, John Hutchinson, and Mark Rouncefield. 2014. The State of Practice in Model-Driven Engineering. *IEEE Software* 31, 3 (2014), 79–95.