Hypersonic: Model Analysis and Checking in the Cloud

Acretoaie, Vlad; Störrle, Harald

Published in:
Proceedings of the 2nd Workshop on Scalability in Model Driven Engineering (BigMDE 2014)

Publication date:
2014

Document Version
Peer reviewed version

Hypersonic: Model Analysis and Checking in the Cloud

Vlad Acretoaie and Harald Störrle
Department of Applied Mathematics and Computer Science
Technical University of Denmark
rvac@dtu.dk, hsto@dtu.dk

ABSTRACT

Context: Modeling tools are traditionally delivered as monolithic desktop applications, optionally extended by plug-ins or special purpose central servers. This delivery model suffers from several drawbacks, ranging from poor scalability to difficult maintenance and the proliferation of "shelfware".
Objective: In this paper we investigate the conceptual and technical feasibility of a new software architecture for modeling tools, where certain advanced features are factored out of the client and moved towards the Cloud. With this approach we plan to address the above mentioned drawbacks of existing modeling tools.
Method: We base our approach on RESTful Web services. Using features implemented in the existing Model Analysis and Checking (MACH) tool, we create a RESTful Web service API offering model analysis facilities. We refer to it as the Hypersonic API. We provide a proof of concept implementation for the Hypersonic API using model clone detection as our example case. We also implement a sample Web application as a client for these Web services.
Results: Our initial experiments with Hypersonic demonstrate the viability of our approach. By applying standards such as REST and JSON in combination with Prolog as an implementation language, we are able to transform MACH from a command line tool into the first Web-based model clone detection service with remarkably little effort.

Keywords

Hypersonic, MACH, Models in the Cloud, clone detection, Web services, Prolog

1. INTRODUCTION

Until very recently, the only practically viable architecture for modeling tools was the traditional rich client architecture for desktop applications, sometimes complemented by specialized central servers, e. g., to provide model versioning and group collaboration capabilities. Such modeling tools are large, monolithical applications, even though some do offer scripting facilities, application programming interfaces (APIs), or a plug-in mechanism to allow for a certain degree of extensibility. A typical example is NoMagic's MagicDraw [21], which can be extended through an API, and complemented by the Teamwork Server plug-in [20] for centralized version control. With this type of modeling tools, the main revenue source for tool vendors is the sale of perpetual licenses for their product, possibly supplemented by ongoing technical support fees. Both the rich client architecture and the associated business model suffer from a series of disadvantages, some of a generic nature and some more specific to modeling tools.

In many other areas of computing, however, other, more flexible architectures are commonplace today. In particular, recent technological developments have brought about the widespread adoption of Cloud-based software architectures. Typically, such architectures involve the deployment of computationally intensive tasks to a centralized and fully transparent shared pool of configurable computing resources (i. e., "the Cloud") [18]. In this context, Web applications are nowadays a widely used method of delivering software functionality of many different kinds, including lightweight editors for parts of UML (e. g., GenMyModel [1], yUML [25]). Though already attracting users, most such Web-based offerings are currently not able to match traditional desktop tools in terms of features. Nevertheless, modeling in the Cloud does have the potential to address some important problems, such as achieving scalability in relation to increasingly large models and model repositories [16].

To realize this potential, we propose a solution where the features required in a fully-fledged modeling environment are hosted remotely and accessed in a transparent way. Playing the role of basic building blocks for a modeling workflow, these features should be accessible independently of each other and across a variety of devices. Fig. 1 visualizes the contrast between the widely used rich client architecture and the solution we propose. The crucial part of this proposal is the identification of features of a modeling environment that should be executed locally, and of those that should be executed on remote servers.

Arguably one of the most suitable application areas for Cloud-based approaches in modeling is model analysis. We select the requirements of this area as a background for constructing Hypersonic, a test vehicle for demonstrating our proposed approach for the delivery of modeling services - in this case, model analysis services. To implement Hypersonic, we use the features offered by our existing model analysis tool, MACH [27]. In its current form, MACH is a desktop application with a textual user interface. As most desktop applications, it requires installation prior to its usage, as well as explicit user actions/approval for installing updates. With Hypersonic, the features implemented in MACH become RESTful Web services. They can be accessed remotely from a wide range of clients, without requiring installation or explicit user actions for installing updates. To demonstrate the usefulness of our Web service API we also create a sample client in the form of a responsive Web application.

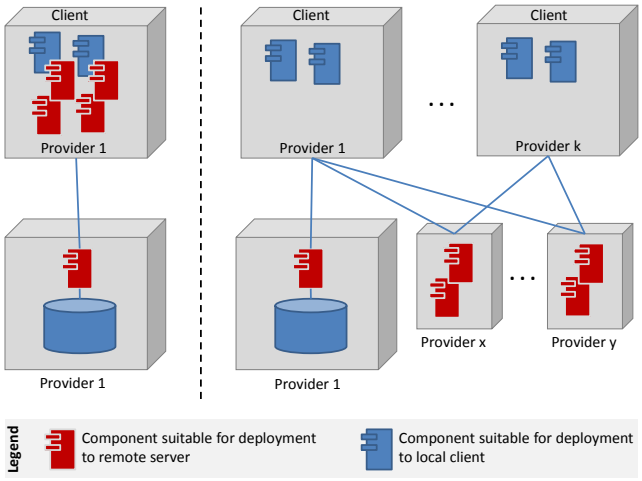

Figure 1: Possible software architectures of modeling tools: most common solution today (left); modeling in the Cloud with Hypersonic (right)

In this paper we discuss the application scenarios and business cases for Cloud-based modeling services, derive requirements and constraints for the associated tools (Chapter 2), propose a distributed architecture to satisfy these requirements (Chapter 3), and report on a proof-of-concept implementation (Chapter 4). We also provide an overview of related work (Chapter 5) and summarize the conclusions of our study (Chapter 6).

2. DEFINING HYPERSONIC

2.1 Analysing Requirements by Stakeholder

Moving modeling tools to the Cloud is not primarily motivated by technological reasons, but by new application scenarios and business cases. In this section we analyze these application scenarios and business cases to highlight the advantages of our proposed architecture. We start by describing the status-quo, considering the stakeholders "tool provider", "modeler", "IT administrator", and "MBSD community", where MBSD stands for Model Based Software Development. Observe that all of these stakeholders exist in similar ways both in academic and commercial settings. We argue that the current state of the MBSD tool landscape is unsatisfactory for the stakeholders in several ways.

To a modeler, tools come as fixed packages: it is usually not possible (or not practical) to use one aspect, the editor, say, of one tool, and another aspect of a second tool. For instance, if the modeler appreciates the editing facilities of tool A, but that tool does not provide (adequate) code generation facilities, code generation may be difficult. It may not be economically viable due to the cost of purchasing two tools, or it may actually be impossible to use the editor of tool A combined with the code generator of another tool, unless both tools strictly adhere to model interchange standards. Also, from a modeler's perspective, resource intensive tasks may take an unreasonably long time to complete when executed locally.

To an IT administrator, repeatedly deploying tools to a large number of computers implies additional effort. Even if this effort is not incurred by the actual deployment (that might be expected to be taken care of by the modelers themselves), it becomes inevitable when distributing updates and fixes, help-desk services, and possibly ensuring that the tool's license usage is compliant with the agreement entered in with the tool vendor.

To the tool provider, delivering a modeling solution as a self-contained product with all the business logic on the client side makes it difficult to support the wide array of emerging computing devices. Migrating a large modeling tool to a tablet, smartphone, or Web interface would likely require extensive re-implementation, not to mention that the hardware requirements of many modeling tools are still prohibitive for mobile devices. Furthermore, separating the highly critical (and sometimes highly innovative) and non-critical functionalities of the application is cumbersome, yet still achievable through plug-ins. The distribution of counterfeit copies of the tool is also difficult to mitigate.

To the MBSD community, all of these factors are limiting, in much the same way as superficial differences between pre-UML modeling languages created niche markets for many different tools that were more expensive and less powerful than the UML tools that emerged after the unification of mainstream modeling languages in the late 1990's.

In a nutshell, the existing situation is suboptimal. Thus, we propose a different architecture: compute-intensive features and features with a high degree of innovation should be deployed to and executed in the Cloud rather than on the client machine. The client and server in this architecture are coupled loosely, by Web services, so that providing a new feature amounts to providing a new Web service. Such features can include advanced model analysis tools, code and report generation, and model transformation. Conversely, features of smaller distinctive value and small change rates, as well as features that require a higher degree of interactivity, can continue to be implemented as part of the client application. This will likely be the case for pure editing features, which many commercial vendors already offer as free community editions of their tools.

We have hinted at a set of criteria for determining which features of a modeling tool should be executed locally, and which would benefit from being migrated to the Cloud. Table 1 summarizes these criteria and presents examples.

2.2 Benefits of Modeling in the Cloud

Several advantages can be achieved by breaking up today's monolithic modeling tools into one stable, remote part that provides little added value distinguishing products (e. g., the editor), and a second, centralized part that comprises more advanced features with a higher change rate and higher distinctive value.

First, there are the advantages associated with thin-client systems in general: maintaining one central server instead of many remote clients reduces the work effort for IT administrators and ensures that modelers always have access to the latest version of the modeling tool. It also becomes easier to monitor license usage, which benefits administrators and tool providers alike.

Second, there are advantages rooted in the specific properties of Cloud-based solutions. This includes the availability of considerable computational resources to each individual user at reduced overall cost, as well as high scalability of the available resources with an increasing number of users.

Third, a scenario in which some modeling facilities are provided as services is conductive to model interchange stan-

	Local Features	Cloud-based Features
Property	high interactivity & incidence high degree of stability exchangeable features	high resource consumption high degree of innovation unique features
Examples	editing simple syntax checking automatic layout context specific help difference visualization querying and navigation	reporting global consistency checking advanced model analysis check in/out, locking difference computation code generation

Table 1: Criteria for assigning features to a local client or remote Web service, together with examples

dards conformance. In such a scenario, service providers and client tool providers must both conform to model interchange standards such as the XML Metadata Interchange (XMI, [22]) in order to meet the requirements of modelers.

Fourth, there are advantages brought about by changes in the business model and enabling market mechanisms. Today, the modeling tool market is dominated by companies providing feature-complete bundles. A new competitor can only enter the market by providing a feature-complete solution, which all but excludes small companies and academic tool providers. Innovative analysis methods, specialized code generators, and similar tools can only be provided as plug-ins to one of the existing platforms, sometimes depending on the approval of the platform owner. In contrast, with a service-based architecture, tool providers can enter the market at lower cost, since they only have to provide their contribution *per se*, not a feature complete-tool. They can also address a larger part of the market, as they provide a generic Web service rather than a platform specific plug-in that applies to only one modeling tool. Modelers, on the other hand, can mix and match services as they like within the limits of standardized exchange formats.

It is likely that the described change in dynamics of the modeling tools market will inspire an increase in competitiveness between existing tool providers, while at the same time providing an incentive for new providers to enter the market. Both these developments will likely lead to a higher degree of innovation. This could translate into new concepts from academia dissipating to industrial modeling at a faster pace. Users will thus have both a financial and a technical incentive to experiment with new features.

From a financial perspective, tool providers will be able to bill features separately as subscription Web services. This could reduce unit prices for customers, who only buy the features they need, and may subscribe to services as they need them. Spending on expensive "shelfware" can be reduced or avoided altogether. For the supplier, this opens the perspective of a new business model in which a steady stream of revenue is generated through subscription services, while features of high distinctive value are much better protected against counterfeiting.

It must be mentioned, though, that adopting a service-based approach to modeling entails some trade-offs. Most are caused by the distributed nature of the approach. For instance, the process of uploading large models to a Cloud-based service may constitute a performance bottleneck. Security and privacy are also new aspects that come into play, considering that a centralized warehouse will store models owned by different organizations. These organisations must be able to trust that not only will their models not be accessible to other users, but they will also be protected against unauthorised mining by the modeling service provider. And, perhaps most importantly, the usefulness of the solution is dependent on a working Internet connection. Nevertheless, these drawbacks are common to the majority of Software as a Service (SaaS) solutions, and have not undermined this architecture style's acceptance.

2.3 A Test Case for Hypersonic

In Section 2.1 we have discussed which types of features lend themselves to deployment as Web services. We now select one such feature, clone detection, as a test case for exploring the proposed architecture (i. e., the Hypersonic API). Our selection is motivated by the following considerations.

- The feature is well researched, published, and implemented (see [26] and [27], respectively), and has been used by a large number of students in several courses in which it has demonstrated its usefulness. So, the feature is readily available and arguably valuable.
- Clone detection demands significant resources, as it is based on semantic and structural model matching. For large models, each clone detection run can be time consuming, so the latency implied by uploading a model to the Cloud and receiving the result may be offset by savings in run-time achieved by using a powerful machine in the Cloud.
- Detecting clones is an activity carried out as part of the model quality assurance process. It is normally not executed concurrently to other modeling activities. Therefore, in some scenarios, a clone detection feature is not required or even useful. For example, models reverse-engineered from code do not require clone detection if the code is known to be clone-free.
- It is a unique feature: no UML modeling tool currently offers a clone detection feature. This includes research prototypes other than our own MACH tool. It is therefore safe to assert that this feature provides a high degree of innovation. Had a tool provider invested resources into implementing a clone detection facility in its product, it would likely be interested in protecting its investment.

3. ARCHITECTURE

A first step towards the realization of the ideas presented in Section 2 is the definition of a common Web service interface accepted by all stakeholders. The details of such an interface must be the result of a wide reaching discussion, which is beyond the scope of this paper. Instead, we take an exploratory approach and design a RESTful Web service API for the purpose of Cloud-based model analysis. We refer to this API as the Hypersonic API. By doing so, we study the requirements and potential setbacks of processing models via RESTful Web services.

In keeping with the REST architectural style [10], the Hypersonic API exposes resources for clients to interact with via HTTP requests. The two exposed resources are *models* and *model*. The *models* resource plays the role of an access point to Hypersonic's model warehouse, whereas the *model* resource represents a single model in the warehouse. These resources are manipulated via HTTP requests, where the HTTP method determines the operation to be performed. In addition, the Hypersonic URL scheme specifies explicit operations on the *model* resource as part of the request URL. The list of supported operations is presented in Table 2.

This architectural approach allows physically decoupling clients from the execution of the various analysis algorithms exposed by the Hypersonic API. This aspect is part of the motivation behind the creation of Hypersonic, since many of these algorithms are resource demanding on models of non-trivial size. By using such a Web service API, a large variety of clients can have access to model analysis facilities regardless of their hardware capabilities. Fig. 2 highlights this aspect, showing that different clients can access existing analysis algorithms provided by the MACH tool via the Hypersonic API wrapper. The only prerequisites for API clients are HTTP support, the ability to process documents returned by the API, and, optionally, a model viewer. Note that all of these prerequisites are entirely reasonable for modern mobile devices.

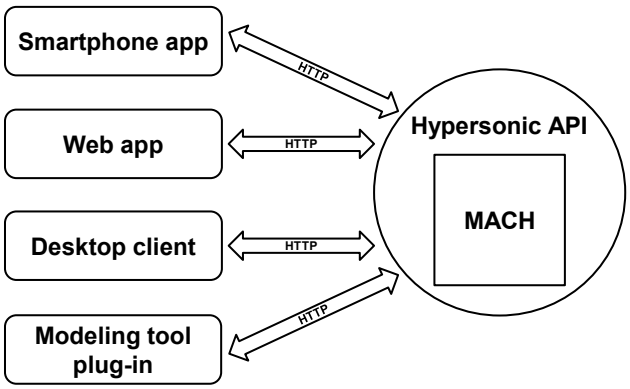

Figure 2: High-level architecture of Hypersonic

Additional non-functional considerations must be taken into account to ensure the practicality of the Web service API. Since using the API implies uploading entire models to a remote server, security becomes an important factor. With this in mind, the OAuth [11] authentication protocol is a widely used solution that can provide some important guarantees to Hypersonic users. The most important such guarantee is that a user cannot gain access to the models uploaded by other users. When combined with a role-based authentication policy, a sound authentication mechanism such as OAuth is an effective way to manage model access rights. At a technical level, implementing OAuth will require all Hypersonic API clients to obtain an access token prior to using the API. This process can be carried out through a separate channel, such as a dedicated API management Web application.

From a file format standpoint, Hypersonic currently supports models stored in the MDXML format, the XMI-based native format of the MagicDraw modeling tool. That is to say, some API requests (e. g., POST requests to the *models* resource) are expected to have an MDXML document as payload. Most API response messages carry a JSON [12] document as payload, representing either the result of an analysis operation or a confirmation or error message.

The internal components involved in answering a call to the Hypersonic API are presented in Fig. 3. The *RESTful API* component handles HTTP communication with remote API clients and delegates all actual model processing to the *MACH* component. It also forwards all models sent by clients to the *XMI2PL* component, which performs a format translation from the MDXML format to the internal Prolog-based file format described in [26]. Once translated, models are stored in a dedicated model warehouse for future analysis upon the client's request. The *MACH* component exposes several supported model analysis and checking algorithms [27]. These algorithms can be applied on models stored in the warehouse. The *MACH* component functions as a self contained black-box, hiding all algorithm implementations from other components and returning the produced results encoded as Prolog lists. The *RESTful API* component handles the translation of these lists into JSON analysis reports ready for consumption by the API client. All processing components are executed inside a single instance of the SWI-Prolog runtime [29], thus allowing seamless inter-component communication.

The SWI-Prolog runtime should be deployed to either a public or private Cloud platform. Since all models are stored separately in a model warehouse, several instances of the SWI-Prolog runtime can be deployed, assuming that the model warehouse provides appropriate concurrent access policies. Persistent model storage can be provided by a separate Cloud storage service. Since models are stored as XML and Prolog files, the storage service should support a document-oriented database management system.

4. EVALUATION

To demonstrate the feasibility of the architecture described in Section 3, a subset of the proposed Web service API has been implemented and is publicly accessible[1]. Due to the reasons elaborated on in Section 2.3, we have focused on a Web service providing model clone detection as a minimum useful scenario. Though important for a final release, we have considered features such as user accounts and authentication beyond the scope of our proof of concept. Both the prototype API and the model warehouse are currently hosted on a dedicated server. They do not benefit from the scalability of a true Cloud deployment, although for the current proof of concept this is hardly a limiting factor.

[1]The Hypersonic API is available at the following base URL: `http://hypersonic.compute.dtu.dk`

Resource	Method	Req. payload	Resp. payload	Description
/models	GET	—	JSON	List all uploaded models.
/models	POST	MDXML	JSON	Upload a model.
/model/<id>	GET	—	MDXML	Download a model.
/model/<id>	PUT	MDXML	JSON	Replace a model.
/model/<id>	DELETE	—	JSON	Delete a model.
/model/<id>/clones	GET	—	JSON	Detect clones in a model.
/model/<id1>/diff/<id2>	GET	JSON	JSON	List differences between two models. Options are specified in the request payload.
/model/<id>/dump	GET	—	JSON	List model elements included in a model.
/model/<id>/dump/<me_id>	GET	JSON	JSON	List the details of a model element. Options are specified in the request payload.
/model/<id>/find/<string>	GET	JSON	JSON	Find a string in a model. Options are specified in the request payload.
/model/<id>/frequency	GET	—	JSON	Compute the meta class frequency distribution in a model.
/model/<id1>/similarity/<id2>	GET	JSON	JSON	Compute the similarity between two models. Options are specified in the request payload.
/model/<id>/size	GET	JSON	JSON	Compute the size of a model. Options are specified in the request payload.

Table 2: List of operations supported by the Hypersonic API

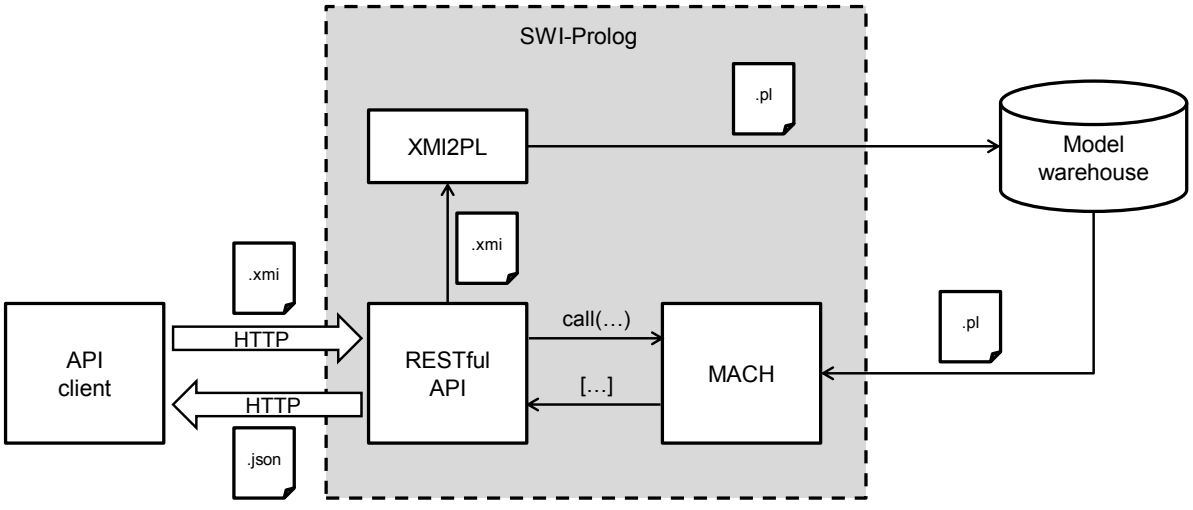

Figure 3: Components which participate in responding to a Hypersonic API request

Fig. 4 represents a message exchange between a client and the Hypersonic API. The purpose of this exchange is to perform clone detection on a model. First, the model is added to the Hypersonic model warehouse by a POST request to the *models* resource. Upon this request's successful handling, a new *model* resource representing the model is available to the client. The resource has a unique identifier returned in the JSON response document of the initial POST request. The client then parses this document and extracts the identifier. Thus, the client is subsequently able to use the identifier to construct the appropriate URL for a GET request to the *clones* operation of the identified *model* resource. The GET request returns a list of clone candidates, also in the form of a JSON document (see Listing 1).

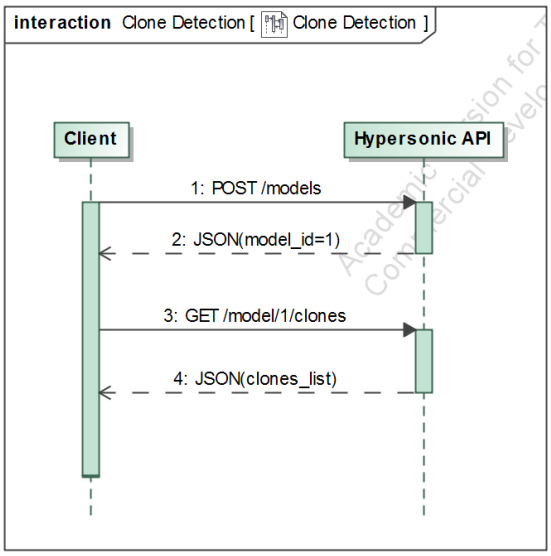

Figure 4: HTTP message exchange for model clone detection

The response document includes a model identifier, the number of detected clones, and a list of discovered clone candidates. Each clone candidate is described by two model elements, where one is a possible clone of the other, a numeric similarity metric computed for the two elements following the approach presented in [26], and a clone "kind" identifying the candidate as either a naturally occurring clone or a seeded clone. Candidates also Clone candidates are returned in the descending order of their similarity scores, i. e., the most likely clone is the first one in the list.

Listing 1: JSON clone detection report

```
{
  "model": "1",
  "candidates": 2,
  "clones": [
    {
      "type_1": "package",
      "id_1": 29,
      "name_1": "Reserve Medium",
      "type_2": "package",
      "id_2": 938,
      "name_2": "Reserve Medium",
      "similarity": 185.7143,
      "kind": "natural clone",
    }
    {
      "type_1": "package",
      "id_1": 189,
      "name_1": "Lend Medium",
      "type_2": "package",
      "id_2": 1194,
      "name_2": "Lend Medium",
      "similarity": 128.7287,
      "kind": "natural clone",
    }
  ]
}
```

By conforming to the architecture presented in Fig. 3, the effort required to implement the clone detection proof of concept has been minimal. The RESTful API functioning as a wrapper around MACH's existing clone detection implementation consists of around 100 lines of Prolog code, largely thanks to the comprehensive support offered by SWI-Prolog for the HTTP protocol. Work on implementing the remaining API calls described in Table 2 is ongoing, as is work on the API management application that must be in place in order to to enable user authentication in API calls.

As a preliminary validation of the API's fitness for purpose, we have created a simple, mobile device friendly Web application as an API client[2]. The application supports selecting a local model file, uploading it to the Hypersonic model warehouse, and requesting a clone report which it subsequently displays in tabular form. The application is written in JavaScript and is executed entirely in the browser (i. e., it does not rely on a server-side script for calling API operations). Though it is so far basic in terms of functionality, this sample client exemplifies our vision of Web service driven modeling tools: using Web 2.0 technologies (REST, JavaScript, JSON) to enable advanced model analysis outside the constraints of the desktop and of traditional modeling environments.

5. RELATED WORK

Model analysis is an activity typically performed in local, non-distributed environments. As an example, the model clone detection operation considered here as a proof of concept has scarcely been addressed in itself, but is closely related to the intensely studied model matching and differencing operations. To name just a few proposals in this area, SiDiff [14] presents a differencing algorithm targeting UML Class Diagrams, while the approach presented in [19] targets sequence charts, and [17] is a clone detection proposal aimed at UML Sequence Diagrams. EMF DiffMerge [8] and EMF Compare 3.0 [3] represent more generic approaches targeting the Eclipse Modeling Framework (EMF, [9]).

With the increase in size of industrially relevant models and the increase in complexity of the operations performed on these models, the need for distributed, Cloud-enabled modeling solutions has become apparent [5, 16]. So far, the main driver behind Cloud-based modeling research has been

[2]The client application is available at `http://www.compute.dtu.dk/~rvac/hypersonic`. It is currently under development, and will be updated to support all operations of the Hypersonic API as they are deployed.

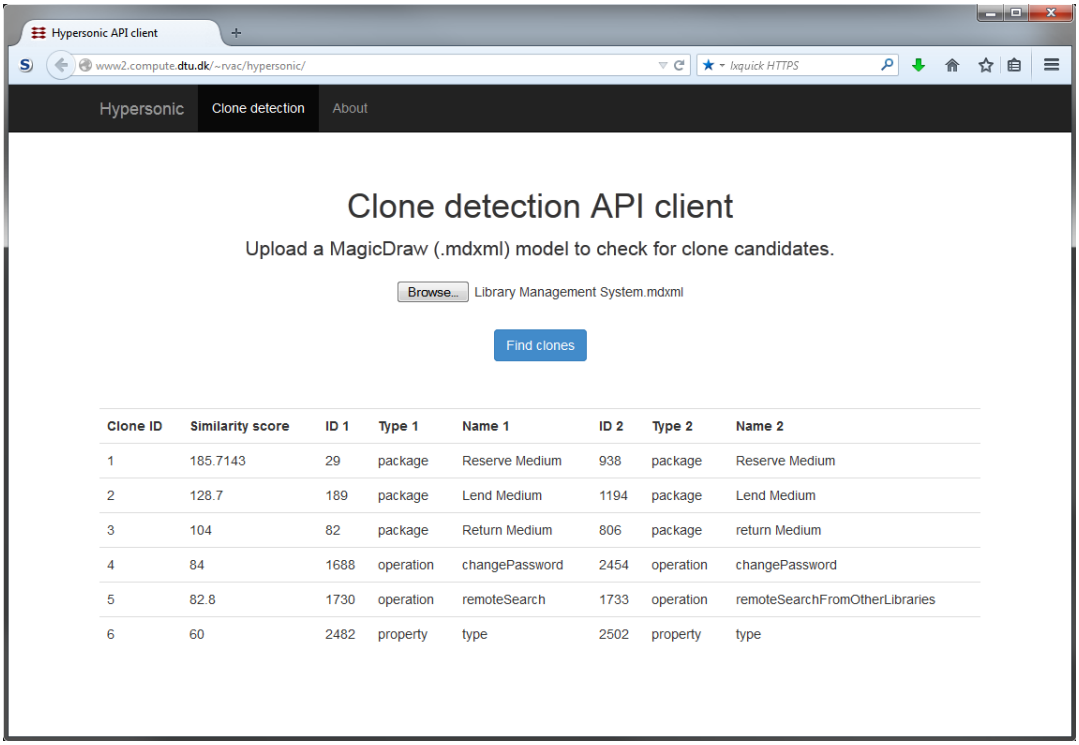

Figure 5: Screenshot of the Hypersonic API client Web application

the promise of important performance and scalability gains for all modeling activities. Perhaps the most fundamental of these activities, model storage, has attracted several proposals, including ModelBus [4], EMFStore [15], and Morsa [23]. These are all remote model warehousing solutions offering Web service access to the stored models.

More advanced activities such as model querying and transformation have also been addressed. IncQuery-D [13] is a tool which takes the established IncQuery tool and adapts it to a scenario where it can be deployed and accessed in the Cloud. The Morsa model repository also benefits from a dedicated query language, MorsaQL [24]. A roadmap for research on Cloud-based model transformations has been presented in [7].

However, performance gains due to Cloud deployment are only a part of the overall vision of Hypersonic. Rather than focusing on the benefits to the application itself (i. e., model analysis), Hypersonic emphasises the benefits brought by a Cloud-based approach to the interface and availability of this application. The idea of performing model analysis via a RESTful Web service API has yet to receive significant attention in the literature. The closest related proposal is the EMF-REST project [6], aimed at automatically generating RESTful Web service interfaces for EMF models, much like existing EMF tools generate Java APIs for such models. Like Hypersonic, EMF-REST uses JSON documents to transport information about remotely stored models. Nevertheless, while it does provide basic model manipulation operations, EMF-REST is not designed as a model analysis tool. Similarly to Hypersonic, EMF-REST is a tool under ongoing development, one of its current limitations being the lack of full support for HTTP methods other than GET.

6. CONCLUSIONS

6.1 Summary

In this paper we have discussed the application conditions, benefits, and general business case for Cloud-based modeling tools. In particular, we have presented a scenario in which modeling capabilities are delivered as Web services to a wide array of clients, ranging from desktop applications to Web and mobile applications. We have contrasted this scenario with the current status-quo of rich client desktop modeling tools, reaching the conclusion that, in many respects, the Cloud-based approach is superior.

To explore our proposal, we have introduced Hypersonic, a RESTful Web service API aimed at offering high-throughput processing for Cloud-based model analysis. We have implemented this architecture and made it available online. Currently, the only service it offers is the detection of model clones, a feature that was previously only available in the MACH command line tool. Today, MACH is a stand-alone desktop tool providing only a textual user interface. Through the Hypersonic API, the features of MACH can be made available over the Internet to any API client. As a proof of concept for the utility of the API, we have developed a Web application acting as a client to the Hypersonic API and providing Web-based model analysis capabilities.

6.2 Future Work

The concepts presented in this paper offer us ample opportunities for future work. As a first step, we will continue the development of the Hypersonic API with the aim of reaching functional parity with the MACH model analysis tool. Once this has been achieved, the API will be deployed to

a Cloud platform. In parallel, we will update the sample Web-based API client to both validate and showcase the model analysis features of Hypersonic. Second, in order to become a practical tool, the API client must offer several critical features such as user authentication and model security mechanisms. Third, we will carry out a systematic performance evaluation of MACH in order to substantiate the claim that Cloud-based model analysis can bring significant performance benefits. Finally, we intend to develop a second client for the Hypersonic API in the shape of a plug-in for MagicDraw. This will permit seamless integration of our Web services approach with a commercial modeling tool and complement our existing model querying MagicDraw plug-in, MQ-2 [2]. As a parallel development, we envision a Web service API similar to Hypersonic for RED, our requirements engineering tool [28].

7. REFERENCES

[1] GenMyModel. `http://www.genmymodel.com`, retrieved 16.05.2014.

[2] V. Acretoaie and H. Störrle. MQ-2: A Tool for Prolog-based Model Querying. In *Proc. co-located Events 8th Eur. Conf. on Modelling Foundations and Applications (ECMFA'12)*, pages 328–331.

[3] M. Barbero. EMF Compare 3.0. `http://www.eclipse.org/emf/compare`.

[4] X. Blanc, M.-P. Gervais, and P. Sriplakich. Model Bus: Towards the Interoperability of Modelling Tools. In *Proc. European MDA Workshops: Foundations and Applications (MDAFA'03/'04)*, volume 3599 of *LNCS*, pages 17–32. Springer Berlin Heidelberg, 2005.

[5] H. Bruneliere, J. Cabot, F. Jouault, et al. Combining Model-Driven Engineering and Cloud Computing. In *Proc. 4th Ws. on Modeling, Design, and Analysis for the Service Cloud (MDA4ServiceCloud'10)*, 2010.

[6] J. Cabot. EMF-REST. `http://emf-rest.com`, retrieved 16.05.2014.

[7] C. Clasen, M. D. Del Fabro, and M. Tisi. Transforming Very Large Models in the Cloud: a Research Roadmap. In *Proc. First Intl. Ws. Model-Driven Engineering on and for the Cloud (CloudMDE'12)*, pages 3–12, 2012.

[8] O. Constant. EMF Diff/Merge. `http://wiki.eclipse.org/EMF_DiffMerge`.

[9] Eclipse Foundation, Inc. Eclipse Modeling Framework (EMF). `http://eclipse.org/modeling/emf`.

[10] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[11] Internet Engineering Task Force (IETF). IETF RFC 6749: The OAuth 2.0 Authorization Framework. `http://tools.ietf.org/html/rfc6749`, 2012.

[12] Internet Engineering Task Force (IETF). IETF RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. `http://tools.ietf.org/html/rfc7159`, 2014.

[13] B. Izsó, G. Szárnyas, I. Ráth, and D. Varró. IncQuery-D: Incremental Graph Search in the Cloud. In *Proc. Ws. Scalability in Model Driven Engineering (BigMDE'13)*, pages 4:1–4:4, New York, NY, USA, 2013. ACM.

[14] U. Kelter, J. Wehren, and J. Niere. A Generic Difference Algorithm for UML Models. In K. Pohl, editor, *Proc. Natl. Germ. Conf. Software-Engineering (SE'05)*, number P-64 in Lecture Notes in Informatics, pages 105–116. Gesellschaft für Informatik e.V. 2005.

[15] M. Koegel and J. Helming. EMFStore: a Model Repository for EMF models. In *Proc. 32nd ACM/IEEE Intl. Conf. on Software Engineering (ICSE'10)*, volume 2, pages 307–308. ACM, 2010.

[16] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, and J. Cabot. A Research Roadmap Towards Achieving Scalability in Model Driven Engineering. In *Proc. Ws. Scalability in Model Driven Engineering (BigMDE'13)*, pages 2:1–2:10, New York, NY, USA, 2013. ACM.

[17] H. Liu, Z. Ma, L. Zhang, and W. Shao. Detecting duplications in sequence diagrams based on suffix trees. In *13th Asia Pacific Software Engineering Conf. (APSEC)*, pages 269–276. IEEE CS, 2006.

[18] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology, 2011.

[19] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proc. 29th Intl. Conf. Software Engineering (ICSE)*, pages 54–64. IEEE Computer Society, IEEE Computer Society, 2007.

[20] NoMagic INC. MagicDraw Teamwork Server. `http://www.nomagic.com/products/teamwork-server`, retrieved 16.05.2014.

[21] NoMagic INC. MagicDraw UML 17.0.3. `http://www.nomagic.com/products/magicdraw`, retrieved 16.05.2014.

[22] Object Management Group (OMG). OMG MOF 2 XMI Mapping Specification, Version 2.4.1. `http://www.omg.org/spec/XMI/2.4.1`, 2013.

[23] J. E. Pagán, J. S. Cuadrado, and J. G. Molina. Morsa: A Scalable Approach for Persisting and Accessing Large Models. In *Proc. 14th Intl. Conf. Model Driven Engineering Languages and Systems (MODELS'11)*, volume 6981 of *LNCS*, pages 77–92. Springer Berlin Heidelberg, 2011.

[24] J. E. Pagán and J. G. Molina. Querying Large Models Efficiently. *Inf. Softw. Tech.*, pages 586–622, 2014.

[25] Pocketworks. yUML. `http://yuml.me`, retrieved 16.05.2014.

[26] H. Störrle. Towards Clone Detection in UML Domain Models. *J. Softw. Syst. Model.*, 12(2), 2013.

[27] H. Störrle. UML Model Analysis and Checking with MACH. In *4th Intl. Ws. Academic Software Development Tools and Techniques (WASDETT'13)*, 2013.

[28] H. Störrle and M. Kucharek. The Requirements Editor RED. In *ECOOP, ECSA and ECMFA 2013: Joint Proceedings of Tools, Demos and Posters*, pages 32–34, 2013. DTU Technical Report 2014-01.

[29] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.