# How Big are Models – An Estimation

Markus Scheidgen

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

**Abstract.** In this report, we look at examples for three modeling applications. We do this for two reasons. The first reason is to discuss the actual practical relevance of large models and the respective mode sizes. The second reason is to identify model usage patterns: which of the common modeling tasks (create, traverse, query, partial load) are actually used, in what frequency, and with what parameters. At the end of this report, we provide a tabular summary of our assessment.

In this report, we look at examples for three modeling applications. We do this for two reasons. The first reason is to discuss the actual practical relevance of large models and the respective mode sizes. The second reason is to identify model usage patterns: which of the common modeling tasks (create, traverse, query, partial load) are actually used, in what frequency, and with what parameters. At the end of this report, we provide a tabular summary of our assessment.

The first one is software modeling, which is what most of the MODELS community is concerned. But techniques originally developed for software modeling are also used for other applications. More abstractly, software modeling can be explained as the representation, analysis, and manipulation of structured data.

Our research group, for example, uses EMF to represent and analyse sensor data received from our wireless sensor network test-bed (*Humboldt Wireless Lab* [8]). Models of heterogeneous sensor data will be our second application. This application is what actually motivated us as authors for this work, because existing approaches where either impractical (lazy loading of EMF resources), or far to slow to store the data at the rate it was produced (CDO and manual relational data base persistence). Refer to [4] for more details.

The third application that we want to analyse, are geo-spatial models and more specifically 3D object models of cities. Such models represent the features of a city (boroughs, streets, buildings, floors, rooms, windows, etc.) as a containment hierarchy of objects and their properties. These models are closely related to sensor data analysis.

For all three applications, we estimate the size of typical models. Furthermore, we determine the most important use cases for all three applications. From these use cases we derive the commonly used modeling tasks and their characteristics.

## 1 Software Models

Model Driven Software Development (MDSD) is the application that modelling frameworks like EMF were actually designed for. In MDSD all artifacts including traditional software models as well as software code are understood as models [6], i.e. directed labelled graphs of typed nodes with an inherent containment hierarchy.

### Model Size

Since models of software code (code models) provide the lowest level of abstraction, we assume that models of software code are the largest software models. Therefore, software projects with a large code base probably provide the largest examples for software models. We will first look at the Linux Kernel as an example for a large software project and then extend our estimates on other operating system projects.

Traditionally code size is measured in *lines of code* LOC (physical lines), SLOC (source LOC, like LOC but without empty lines, comments, duplicates), and LLOC (logical LOC, like SLOC but normalized to one statement per line) [7]. These measures exist for many known software projects (and their history) and can be easily captured for open source projects.

To estimate the size of code models in number of objects, we additionally need to know how many model objects constitute an average LOC. We used the C Development Toolkit (CDT) to parse the current version of the Linux Kernel (3.2.1) and count the number of abstract syntax tree (AST) nodes required. We assume that model objects and AST nodes are equivalent for our estimation. We also counted the LOCs of the Kernel's code with *sloccount*. This provides a model objects per LOC ratio of 5.99 that we use for all further estimates.

From the GIT versioning system we learned the average numbers of added, removed, and manipulated LOCs per month based on last year's history (4,300 LOC added, 1,800 LOC removed, and 1,500 LOC modified per day). We extrapolate these numbers to estimate the older history of the Kernel. Based on the actual grow in LOC over the last years and our model objects per LOC ratio, we calculated the average number of modified objects in a modified LOC, and can finally estimates the number of all objects in the Kernel's code including its history. This is a model that only contains the changes.

We also transferred all ratios from the Kernel to other OS software projects and publicly reported LOC counts. The results represent rough estimates and are presented in Fig. 1. As you can see, these large software models can have a size up to a magnitude of $10^9$ objects.

Furthermore, it is not clear whether further growth in software project size is exponential or linear. Some believe that software size follows Moore's Law. Others think that software is bound by increasing complexity and not by the limitations of underlying hardware.

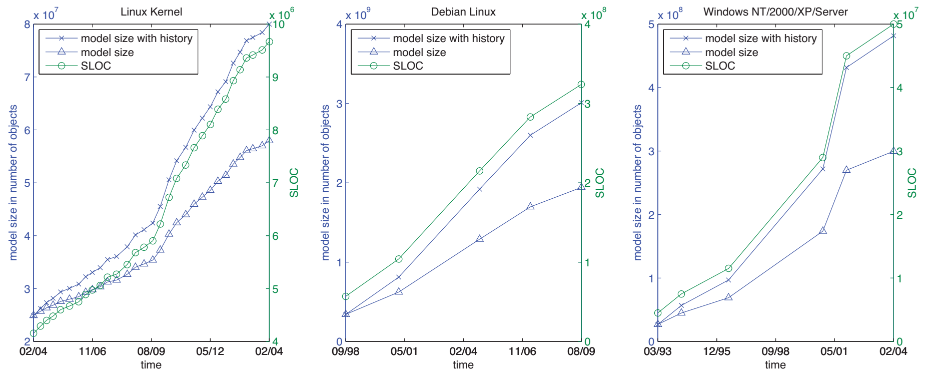

**Fig. 1.** Rough estimates for software code model sizes based on actual SLOC counts for existing software projects.

## Usage Patterns

There are two major use cases in today's software development: editing and transforming or compiling. The first use case is either performed on diagrams (graphical editing) or on compilation units (e.g. Java-files, textual editing). Diagram contents roughly corresponds to package contents. Both packages and compilation units are sub-trees within the containment tree of a software model. Transformations or compilations are usually either done for the whole model or again on a per package or compilation unit basis. Within these aggregates, the (partial) model is traversed. A further use-case is analysis. Analysis is sometimes performed with single queries. But due to performance issues, model analysis is more often performed by traversing the model and by executing multiple queries with techniques similar to model transformations. Software models are only accessed by a few individuals at the same time.

## 2 Heterogeneous Sensor Data

Sensor data usually comprises of time series of measured physical values in the environment of a sensor; but sensor data can also contain patterns of values (e.g. video images). Sensor data is collected in sensor networks, that combine distributes sensors with an communication infrastructure. Sensor data can be heterogeneous: a sensor network can use different types of sensors that measure a multitude of parameters. [1,3].

Our research group build the *Humboldt Wireless Lab* [8], a 120 node wireless sensor network. Nodes are equipped with 3 axis accelerometers, but more essentially also produce data from monitoring all running software components (mostly networking protocols), and other system parameters (e.g. CPU, memory, or radio statistics). We represent and analyze this data as EMF based models ([4]).

### Model Size

HWL's network protocols and system software components provide 372 different types of data sets. Each data set is represented as an XML document. Per second each node in the network produces XML entities that translate into an average of 1120 EMF objects. A common experiment with HWL involves 50 nodes and measures of a period of 24 h. During such an experiment, the network produces a model of $5 \times 10^9$ objects.

In general, sensor networks are only bounded by the limitations of the technical infrastructure that supports them. Ambiguous computing and Smart Cities suggest future sensor data models of arbitrary sizes.

### Usage Patterns

There are two major use-cases: recording sensor data and analyzing sensor data. Recording sensor data means to store it faster than it is produced. If possible in a manner that supports later analysis. Sensor data is rarely manipulated. Analysis means to access and traverse individual data sets (mostly time series). Each data set or recorded set of data sets is a sub-tree in the sensor data model. Recording and analysis is usually performed by only a single (or a few) individuals at the same time.

## 3 Geo-spatial Models

3D city models are a good example for structured geo-spatial information. The CityGML [2] standard, provides a set of XML-schemata (building upon other standards, e.g. GML) that function as a meta-model. CityGML models represent the features of a city (boroughs, streets, buildings, floors, rooms, windows, etc.) as a containment hierarchy of objects. Geo spatial models usually come in different levels of details (LOD); CityGML distinguishes 5 LODs, 0-4) [2].

### 3.1 Model Size

As for many cities, a CityGML model is currently established for Berlin [5]. The current model of Berlin covers all of Berlin, but mostly on a low-medium level of detail (LOD 1-2). To get an approximation of the model's size, we counted the XML entities. The current Berlin model, contains $128 \times 10^6$ objects.

Estimated on existing CityGML models published for Berlin [5] a LOD 1 building consists of 12.5 objects, a LOD 2 building of 40 objects, and a LOD 3 building of 350 objects. A complete LOD 3-4 model of Berlin would therefore consist of $10^9$ objects. Berlin inhabits 3.5 million people. About 50% of the worlds $7 \times 10^9$ people live in cities. This gives a whole LOD3-4 approximation of $10^{12}$ objects for a *world 3D city model.*

### 3.2 Usage Patterns

Compared to model manipulation, model access is far more common and its efficient execution is paramount. If accessed, users usually load a containment hierarchies (sub-tree) corresponding to a given set of coordinates or address (geographic location): partial loads. Queries for distinct feature characteristics within a specific geographic location (i.e. with-in such a partial load) are also common. Geo-spatial models are accessed by many people at the same time.

## Summary

The following table summarizes this section. Two + signs denote that execution times of the respective tasks are vital for the success of the application; a single + denotes that the task is executed often, but performance is not essential; a $-$ denotes that the task if of minor importance.

| **application** | **model size** | **create/mod.** | **traverse** | **query** | **partial load** |
|---|---|---|---|---|---|
| software models | $0 - 10^9$ | + | ++ | + | + |
| sensor data | $10^9$ | ++ | ++ | - | ++ |
| geo-spatial models | $10^9 - 10^{12}$ | - | - | ++ | ++ |

## References

1. Estrin, D., Girod, L., Pottie, G., Srivastava, M.: Instrumenting the world with wireless sensor networks. In: IEEE International Conference on Acoustics, Speech, and Signal Processing. Salt Lake City, UT , USA (2001)
2. Gröger, G., Kolbe, T.H., Czerwinski, A., Nagel, C.: OpenGIS City Geography Markup Language (CityGML) Encoding Standard, Version 1.0.0. Tech. Rep. Doc. No. 08-007r1, OGC, Wayland (MA), USA (2008)
3. Lynch, J.P.: A Summary Review of Wireless Sensors and Sensor Networks for Structurr al Health Monitoring. The Shock and Vibration Digest 38(2), 91–128 (2006)
4. Scheidgen, M., Zubow, A., Sombrutzki, R.: ClickWatch – An Experimentation Framework for Communication Network Test-beds. In: IEEE Wireless Communications and Networking Conference. France (2012)
5. Stadler, A.: Making interoperability persistent: A 3D geo database based on CityGML. In: Lee, J., Zlatanova, S. (eds.) Proceedings of the 3rd International Workshop on 3D Geo-Information, pp. 175–192. Springer Verlag, Seoul, Korea (2008)
6. Thomas, D.: Programming With Models – Modeling with Code. Journal of Object Technology 5(8) (2006)
7. Wheeler, D.A.: Counting Source Lines of Code (SLOC). http://www.dwheeler.com/sloc (2002)
8. Zubow, A., Sombrutzki, R.: A Low-cost MIMO Mesh Testbed based on 802.11n. In: IEEE Wireless Communications and Networking Conference. France (2012)