Composition Challenges for Sensor Data Visualization

Ivan Logre Sébastien Mosser Michel Riveill

Univ. Nice Sophia Antipolis
CNRS, I3S, UMR 7271
06900 Sophia Antipolis, France
lastname@i3s.unice.fr

Abstract

Connected objects and monitoring systems continuously produce data about their environment. Dashboards are then designed to aggregate and present these data to end-users. Technologies used to design and implement visualization dashboards are babbling from a software engineering point of view. This paper highlights how this domain could benefit from leveraging separation of concerns and software composition paradigms to support dashboard design.

Categories and Subject Descriptors D.2.8 [*Software Engineering*]: Data visualization, Software composition

Keywords Visualization, sensor, data, composition

1. Introduction

The *Internet of Things* relies on physical objects interconnected between each others, creating a mesh of devices producing information. In this context, sensors are surrounding our environment (*e.g.*, cars, buildings, smartphones) and continuously collect data about our living environment. In order to add value to these raw data sets, visualization dashboards are designed to support end-user decision making process. Unfortunately, the tools available to design and implement such dashboards are holistic and do not take into account the inherent modularity of this domain. This paper does not aim to describe a solution, but instead focuses on the challenges triggered by the design of visualization dashboards, and align them with modular paradigms such as separation of concerns and software composition.

2. Visualization Dashboards

Process overview. To design and implement a visualization dashboard, one selects the visualizations applied to refined datasets to achieve some identified goal.

This process involves three roles:

1. a *Requirement Engineer* (RE);
2. a *Data Manager* (DM);
3. a *Dashboard Designer* (DD).

Each role is responsible for several tasks: *(i)* the RE defines what will be the purpose of the dashboard and audit the resulting dashboard according to the initial motivation, *(ii)* the DM selects data to be visualized and treat it to provide the needed well formatted datasets, *(iii)* the DD choose visualization for each group of data and arrange them spatially into a dashboard. These roles needs to collaborate, *e.g.*, the choice of visualization the DD has to make depends on the purpose exposed by the RE. They rely on distinct domains of expertise and formation, each one bringing its own challenges and none is optional in order to produce a satisfying final product. Thus, each role is usually impersonated by a dedicated stakeholder.

Designing dashboard. One could use existing solutions to implement each of the mentioned tasks, for example using SQL would be a suitable choice to query and refine sensor data. The *Interaction Flow Modeling Language* [6] models the wanted organization of the dashboard, and temporal logic or *Concurrent Tasks Tree* (CTT) [5] define the aimed product and the sequence of actions to be performed with it. Those tools were not designed for the visualization field and suffers from their generic approaches by being difficult to use by visualization designers. For example CTT does not offer concepts to characterize visualization needs. Moreover, there is a lack of interoperability between those tools due to the distinct field they come from. This results in a difficulty for the stakeholders to dialogue and converge toward a solution when a compromise is needed.

Implementing dashboard. To implement a given dashboard, one can use visualization widget libraries, either professional solutions such as HighChart[1] and AmChart[2] or community-based libraries such as D3.JS[3]. Then, one will add HTML5/CSS code to structure the result. However, those widgets do not allow their integration with a lot of data format, since the development effort is put on the interaction aspect instead of the interoperability. In addition, the huge amount of available widgets (*e.g.*, D3.js offers 235 widgets on January 2015) increase the difficulty to select a suitable visualization. There is a lack of effort in the categorization of those new visualization capabilities[4]. These last two points strengthen the difficulty to cooperate with other domains, considering the gap between the conceptual role of the RE and the implementation role of the DD, and because of the incompatible constraints imposed by the chosen libraries on data format then reduce reusability.

[1] `http://www.highcharts.com/`

[2] `http://www.amcharts.com/`

[3] `http://d3js.org/`

3. Challenges

The previous section described how people support the design and implementation of dashboards using a classical development process. In this section, we describe the challenge of isolation that undermine this process, detailing it through two axes: evolution capacities and dashboard integrity. We discuss how separation of concerns and software composition could be helpful to support this process.

Isolation challenge. Each stakeholder should be able to work in his domain of expertise, isolated from the other domains. This main challenge implies that each one work without the irrelevant noise of others' contributions, focusing on concepts of her own domain, or, in case of a shared concept, only on the facet of this concept relevant for the task of this role. For example, as a dashboard designer, one can define a link between the visualization being constructed and some data, but should not be requested to be competent as a data manager while working on it. The relevant information from a data polishing point of view are handled by a specialist of this domain, but not only this specialist would need to reference a specific dataset.

Evolution capabilities. Data visualization is a growing, fast evolving field: 6,440,000 Google results for *"data visualization"* on 01/15, D3.js offered 133 widgets on 04/14 and now 235 on 01/15. *Requirement engineering* (RE) is a very active research field offering new way to capture needs: more than 730 papers published since 2014 and referenced by Google Scholar contains *"requirement engineering"*, and 640 *"goal model"*. In addition, data scalability is still an open scientific lock. For these reasons, their respective tool or *Domain Specific Languages* (DSL) used have to evolve in time. Exploiting a composition-based approach to represent the widgets will support the evolution capabilities of the whole design framework. In addition, each of these domains bring a unique expertise useful to the data visualization design field. Keeping them separated allows each stakeholder to perform dedicated tasks with state of the art capabilities of the associated domain. Nonetheless, this separation of role specific solutions require to handle the interaction between those partial results, bridging the gap between the domains and manage the high versatility of those research fields.

Integrity. Separation of concerns allows one to contribute to the dashboard design process in her own domain. It is then possible to check if this work is consistent inside this domain, even if the other domains are in an unstable or incoherent state or if it is not possible to check the global consistency at the moment. For example, a data manager is allowed to edit a resource measurement unit to optimize the data transfer and validate her contribution from the data point of view, even if it may have broke the choice of visualizations used in the dashboard design domain. In order to support a proper separation of concerns, each domain solution has then to be usable independently from any considerations of interaction with the other collaborating domains. This point raises a challenge on the interaction of these partial solutions, introducing the notion of local and global consistency to handle.

Actionable insights: DSL composition. One interesting way to tackle those challenges would be to design a DSL for each of the three domains mentioned and then to compose those partial results in a overall data visualization solution. The state of the art reveals several ways to manage this composition [2]:
(i) Merge, *i.e.*, the operation to produce one bigger meta model from several meta models by identification of a pivot and merging from it, and then refine the associated concrete syntaxes to produce a global one [7].
(ii) Aggregation, *i.e.*, make enough assumptions about the meta-models to be able to link them through the transformation of several meta-models by adding, deleting or editing specific model elements, essentially to align two concepts from different domains or to reference an external concept in order to delegate part of the responsibilities [1].
(iii) Viewpoint unification, *i.e.*, let each domain expert works with her proper DSL, while composing the abstract syntaxes and semantics of each to produce an integrated meta model to reason about [8].
(iv) Embedding, *i.e.*, the specialization of one or several concepts of an host meta model through the definition of a guest one, tailored to extend the host to a new aspect of the whole system complexity while preserving its semantics [3].

4. Conclusions & Perspectives

In this paper, we described the domain of visualization dashboard design. This domain crosscuts several research fields, from human-computer interactions to big data. We highlighted three challenges in this domain where a software engineering approach based on modularity concepts could support it. However, all the challenges triggered by this domain are not yet solved from a separation of concerns point of view.

In our upcoming works, we plan to focus on defining a formal way to support exchanges between the different roles involved in the domain by a formal identification of the relations between each stakeholder domain and through integration of domain specific solutions while emphasizing integrity and isolation properties on this composition.

References

[1] D. Blouin, Y. Eustache, and J.-P. Diguet. Extensible global model management with meta-model subsets and model synchronization. In *GEMOC 2014*, pages 43–52, 2014. URL `http://ceur-ws.org/Vol-XXX/#paper-07`.

[2] M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In *OOPSLA–6th Workshop on Domain Specific Modeling*, pages 123–139, 2006.

[3] D. M. Groenewegen, Z. Hemel, L. C. Kats, and E. Visser. Webdsl: a domain-specific language for dynamic web applications. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 779–780. ACM, 2008.

[4] I. Logre, S. Mosser, P. Collet, and M. Riveill. Sensor Data Visualisation: a Composition-based Approach to Support Domain Variability. In *European Conference on Modelling Foundations and Applications (ECMFA'14)*, pages 1–16, York, United Kingdom, July 2014. Springer LNCS. URL `http://www.i3s.unice.fr/~mosser/media/research/ecmfa14.pdf`.

[5] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *INTERACT*, pages 362–369, 1997.

[6] G. Rossi. Web modeling languages strike back. *IEEE Internet Computing*, 17(4):4–6, 2013. ISSN 1089-7801. .

[7] M. Schottle and J. Kienzle. On the challenges of composing multi-view models. In *Proceedings of the First Workshop On the Globalization of Modeling Languages, GeMOC 2013*, pages 1–6, 2013.

[8] A. Vallecillo. On the combination of domain specific modeling languages. In *Modelling Foundations and Applications*, pages 305–320. Springer, 2010.