Using Grounded Theory for Domain Specific Modelling Language Design

Sybren Kinderen

Using Grounded Theory for Domain Specific Modelling Language Design

Lessons Learned from the Smart Grid Domain

Sybren de Kinderen

University of Duisburg-Essen, Essen, Germany
sybren.dekinderen@uni-due.de

Abstract. This paper shows how Grounded Theory (GT), a method for domain understanding predominantly used in the social sciences, can be useful for the design of a Domain Specific Modelling Language (DSML). Using a pilot study from the smart grid domain, we discuss how GT can be used to systematically derive the abstract syntax of a DSML from domain data. From this, we derive lessons learned from the application of GT, the most relevant being that (1) with GT, one systematically derives an abstract syntax of a DSML, reflecting domain commonalities and variation points, (2) in line with its explorative character, with GT one gains a grounded domain understanding, and the domain goals that a DSML should satisfy, (3) GT does imply a notable time investment, which one needs to weigh against its prospective benefits. Finally we present a concluding outlook in terms of implications for DSML design mechanisms.

Key words: Grounded Theory, Domain Specific Modelling, Language Design, Smart Grid

1 Introduction

Domain Specific Modelling Languages (DSMLs) reconstruct the concepts of a specific domain (e.g., electricity, healthcare), thereby increasing their expressiveness of the particulars of that domain [8, 17]. DSMLs steadily gain in proliferation [16, 14] in part due to: fostering communication with domain experts, by staying close to domain-specific terminology [8]; increasing modeling productivity, by not having to reconstruct domain specific knowledge from scratch [17, 9], and; acting as a thinking tool [26, p. 41], since DSMLs abstract from unnecessary details.

Being reconstructions of domain concepts, the design of DSMLs should naturally reflect the commonalities and variations of a domain. As an example, consider the "browsing" domain captured in [17]. It has the common features "get", "post", and "index" (which are features necessary for browsing), but has a variable set of plug-ins, such as the feature "flash" (specific plug-ins being

features that differ across the different domain solutions, reflected in different specifications).

To ensure that DSML design is close to a domain, existing DSML design literature aims at purposeful, "top-down", DSML design, and eliciting user feedback. It ranges from comprehensive approaches such as [8, 17, 25], to language design guidelines [14], to supporting diagram types [26, p.521]. As a precursor to language design [8] [25, p.152] suggest use scenarios. These ensure purposeful language design, and foster feedback from prospective users [25, p.152]. Similarly, for feedback purposes, [8] suggests the use of mockup diagrams. Further, [14] provides language design guidelines such as "Reflect only the necessary domain concepts" [14], which points towards purposeful language design. Also, [26, p.521] [17] propose feature diagrams to capture the commonalities and variation points of domain features.

However, comparatively little attention is paid to systematically reconstruct a DSML inductively, "bottom-up", from existing representations of a domain [7, 27], such as technical specifications, source code, or requirements documents. Consequently, one risks that a rich domain variety is not fully accounted for in a DSML. Also, by forgoing a systematic grounding in domain data, current DSML design approaches make little effort to systematically reconcile the externalized knowledge of different domain experts (e.g., in the form of interview transcripts, or meeting minutes). As such, one does not systematically compare the results of elicitation efforts across different prospective language users, in terms of differences and commonalities in their points of view.

One method that is promising for uncovering domain variation is Grounded Theory (GT) [6]. GT is a method for domain understanding that is used predominantly in the social sciences. Two key features of GT make it also of interest for DSML engineering: (1) GT aims at studying *variations and commonalities* between different qualitative sources, deriving its concepts inductively from a qualitative data set. As a result, GT synthesizes commonalities and variation points of domain data, and can be used to systematically compare different points of view of different stakeholders. (2) In GT concepts are key to theory development [3]. Starting from the qualitative data sources one has, one reconstructs the key domain concepts, their perceived meaning and interrelations. Thus, in developing a domain understanding, it complements the way of thinking used in DSML design.

The goal of this paper is to show how GT can be used for DSML design. Using a case study from the smart grid industry, we show how GT can be used to systematically create (1) an object oriented diagram, representing the abstract syntax of a DSML, and (2) a domain goal diagram. In line with the explorative spirit of GT, this goal model complements the abstract syntax with a grounded understanding of the domain goals behind it.

Note here that we focus on developing the abstract syntax of a DSML, plus the understanding of the domain goals behind it. The concrete syntax (i.e., the visualization) is out of scope.

This paper is structured as follows. In Sect. 2, we introduce the key ideas of GT in the light of DSML design. In Sect. 3, we introduce our case study from the smart grid domain. Subsequently, Sect. 4 shows how GT can be useful for DSML design, using the case study as a running example. It also discusses lessons learned. Sect. 5 presents related work. Finally, Sect. 6 provides a concluding outlook, setting out implications for DSML design.

2 Theoretical Foundations: Grounded Theory

GT is a *qualitative* research method that produces a *domain* conceptualization in a largely *inductive* manner. As a *qualitative* research method, the research questions driving GT are exploratory [6, p. 25]. As such, in line with qualitative research generally, GT is aimed at increasing understanding of a phenomenon about which relatively little is known. Key to a qualitative research project carried out with GT, is that it concerns itself with systematic *domain conceptualization* from a qualitative data set [3, 6]. Here, "domain conceptualization" refers to characterizing a domain in terms of concepts and their properties, as well relationships between concepts [3]. In deriving the conceptualization from a qualitative data set – hence a theory "grounded" in a data set – GT is largely (though not exclusively [24]) *inductive*. In line with its explorative character, its point of departure for theory building is less a well established body of knowledge, more a pool of qualitative data in the form interview transcripts, technical documentation, or otherwise.

The systematic domain conceptualization by means of induction is what makes GT an especially appealing method for supporting DSML design, in that (1) for both DSML design and GT, concepts are in focus. This helps in combining their respective insights. (2) The systematic domain conceptualization by means of induction helps to structurally "ground" a DSML in domain data, as well as, as we shall see, to systematically establish a solid understanding of a relatively unexplored domain.

Finally, of note is that GT is concerned with helping to analyse "... the actual production of meanings and concepts used by social actors in real settings" [24], quoting [10]. As such, it is aimed at understanding commonalities and differences in how different actors *interpret* a certain phenomenon. It is not aimed at defining one true and final positivist account of a domain. However, due to the predominantly technical nature of the pilot study (the conceptualization of IT infrastructure for the smart grid), this particular side of GT will be considered as out of scope for the remainder of this paper.

Next, we explain the case study used for our GT-driven DSML design effort.

3 A Smart Grid IT infrastructure DSML

Study Setup. This case study is part of a larger research project with the objective of providing model-driven support for doing a technical as well as a

cost-benefit analysis of smart grid initiatives. Smart grid initiatives refer to initiatives in the electricity domain that are enabled by developments in ICT [2, p. 14], such as smart meters. Once a smart grid initiative is shown to be technically feasible, the next step is to actually realize it as an economically viable business [19]. For example, one needs to assess who will be involved, we well as the involved costs and benefits, both purely quantitatively (e.g., investments in equipment) and qualitatively (e.g., societal benefits such as CO_2 reduction).

In this paper, we focus on developing a language for expressing a technical IT infrastructure of a smart grid initiative. The technical details of such a smart grid IT infrastructure have an influence on the corresponding economic side [21], hence they need to be considered together with the economic analysis.

For DSML design with GT the advantages of this focus are as follows:
• It is a relatively unexplored domain, in terms of conceptual modelling languages that detail technical IT infrastructure for the smart grid. The closest that we have is the Smart Grid Architecture Model (SGAM [2]). However, due to its high level of abstraction SGAM is mainly suitable as an abstract reference model (e.g., to state that we consider "components" but not "communication", or "data", at the levels of "customer premises" and "distribution"). It is less suited for modelling specific IT infrastructure components and their relations.
• Technical specifications are appropriate for a pilot study: they are diverse enough to capture variation of domain concepts, yet concrete enough to minimize the risk of "getting lost".

For scoping reasons, we focus on conceptualizing one specific part of the IT infrastructure: a substation PC. Electricity substations transform electricity from the high-voltage grid into low-to-medium voltage for the end-users (such as households). Substation automation, then, can help with monitoring this electricity transformation, e.g., in noticing an electricity overload or outage before customers place a call.

Materials. Our qualitative data set consists of technical specifications of substation automation solutions: hardware, software, and the pricing of both.

The specifications consist of two parts: purely technical specifications (e.g., permanent storage having a "RAID1" capability), and advertising text (e.g., "maximize data availability" [22] for "RAID1" capabilities). We analysed both parts, to find both purely technical features (which would be interesting for specifying the technical hardware capabilities), and to gather input for domain understanding. Concerning the latter: one needs of course be careful with advertising texts, but the used wording does hint at what is important for the domain, thus - especially over multiple documents - helping us to gather core goals that the technical specifications should comply with.

We coded the specifications in MAXQDA[1], a software tool that supports qualitative research. Apart from supporting qualitative research activities (such as, as explained in Sect. 4.1: code reuse over multiple documents, having a memoing function, the ability to build coding trees), we find that the main advantage

[1] http://www.maxqda.com/products/maxqda-standard

of using a software tool for qualitative research is that we have a direct trace between codes and parts of the document they belong to. Thus, one can always revisit assumptions made on the basis of existing codes, or complement already generated codes if approached from a different angle.

In total we coded eleven documents: two for pricing, five for hardware, two for software, two all-in-one solutions. We continued coding specifications until no new insights would be gained by coding additional specifications.

In addition, we had an informal feedback session with an academic expert from the smart grid domain to discuss our developed models. This session acted mainly as a sanity check for the features emphasized in the advertising texts.

4 Grounded Theory for DSML Design

We now explain how GT is used for DSML design. Our GT-driven design process consists of two steps. In Step 1, we use GT to develop a domain conceptualization in terms of GT coding trees derived from domain data. In Step 2, we use the GT coding trees to develop diagrams for DSML design.

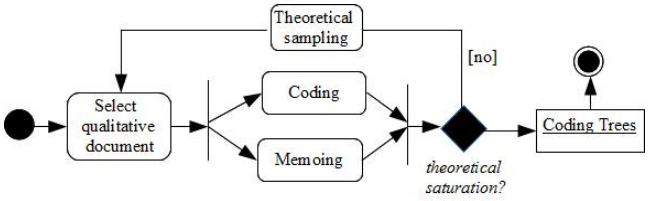

Fig. 1. The key GT activities for qualitative data analysis

4.1 Step 1: Using Grounded Theory to Develop Coding Trees

Fig. 1 shows the key activities of GT to derive a domain conceptualization from qualitative data: for each additional data source, we do coding and memoing for qualitative analysis. Then, using theoretical sampling, the next document is selected. Subsequently it is analyzed using, again, coding and memoing. This process continues until theoretical saturation is reached [3].

In what follows, we discuss the activities of coding, memoing, theoretical sampling, and theoretical saturation, and guidelines for each. These guidelines follow a Straussian interpretation of GT (cf. [6]). We follow Straussian GT since it provides more guidance in carrying out GT activities compared to Glaserian GT, the key alternative interpretation of GT. See [5] for a detailed comparison of Glaserian and Straussian GT. In our discussion, we highlight points that play an important role in language design. Note here that, since GT is a qualitative research method from the social sciences, we speak here of "guidance" rather than a specific algorithm to follow.

Coding. With coding, the researcher assigns meaning to qualitative data [18]. While there is no "wrong way" of coding in GT, it is important to keep in mind that coding is a central analysis activity, requiring one to work actively with the qualitative data at hand. It is not mere labeling of text [24].

In GT coding happens in parallel to data collection [6, p. 57]. This means that one starts coding with the first qualitative document, and progressively modifies the code system while coding additional documents. To avoid drowning in the dataset, in Straussian GT, the qualitative analysis is guided by a research question which should be specific enough to focus of the coding effort [6, p. 24].

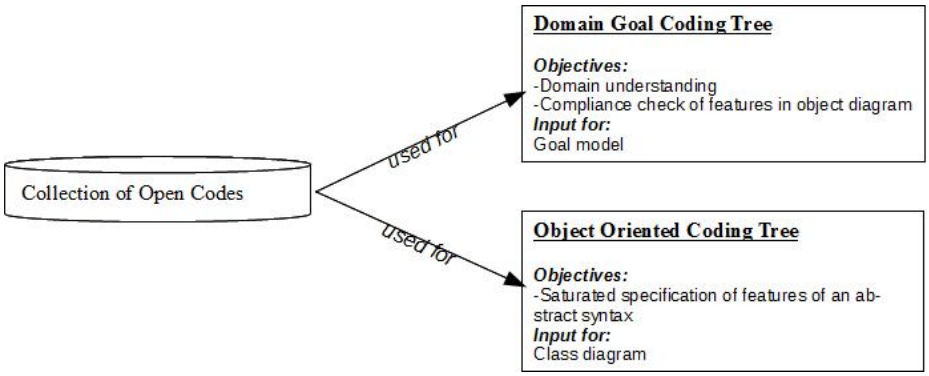

Fig. 2. Grounded Theory for creating a domain goal coding tree and an object oriented coding tree

Straussian GT distinguishes three types of coding: open, axial, and selective/focused (cf. [6]). With *open coding* one breaks qualitative data apart by labeling blocks of raw data with concepts. Note that for open coding there is no fixed unit of analysis: it can happen on a word, sentence or paragraph level [3]. With *axial coding* one relates concepts to each other. Finally, with *selective/focused coding* one defines the concepts that are important to explain the phenomenon at hand [3]. In GT, such concepts are referred to as *categories*. Categories are created either by the open codes directly, or - often the case - by aggregating multiple codes under a more abstract category. To aid in the definition of categories, Straussian GT provides several hints such as asking journalistic questions: "why", "who", "when", "with what consequences" [6, p. 199].

It is important to note that these coding activities are not necessarily orthogonal to each other. For one, [6, p. 118] point out that open and axial coding often take place in parallel. This is because, by delineating codes from each other, naturally ideas emerge on possible relations between them.

The result of a coding effort is a coding tree, which summarizes how detailed (often open) codes are gradually aggregated into more abstract categories. Such a coding effort then also acts as a key input for DSML design. As depicted in Fig. 2, for DSML design we develop two such coding trees: an object oriented coding

tree, and a domain goal coding tree. We do so because for DSML design we desire to create two diagrams: (1) a domain goal diagram, providing an understanding of a domain, and identifying requirements that the domain imposes on a DSML, and (2) an object oriented diagram, in terms of the abstract syntax of the DSML. The creation of a separate domain goal diagram is also in line with DS(M)L design literature such as [26], who propose to use feature diagrams as a precursor for DS(M)L development.

As per Fig. 2 notice that a set of codes is reused amongst the two coding trees. This was done because, although the domain goal diagram and object oriented diagram evolve in different directions, their starting point is essentially the same: a set of domain specific codes, which subsequently result in different coding trees because of different questions asked.

EXAMPLE: Fig. 3 shows an excerpt of a domain goal tree and a object oriented code tree for a substation PC. We developed these trees by first generating open codes of technical substation specifications, remaining faithful to original wording. For example, a technical specification of a "SSD", described as an option for a "HardDrive" (cf. the technical specification in [22]), can be developed into two open codes: "Solid State Drive" and "Secondary Storage" (a slightly more accurate rewording of "Harddrive").

Object Oriented Diagram Coding Tree (Fragment)	Domain Goals Diagram Coding Tree (Fragment)
❖ Primary Storage ➢ ECC Memory ➢ storageCapacity ❖ Secondary storage ➢ SecondaryStorageType ▪ Harddrive ▪ Solid State Drive ➢ RAID1 ➢ Hot swapping storage ➢ storageCapacity	❖ Minimize data loss ➢ RAID1 ➢ ECC Memory ❖ Minimal moving parts ➢ Solid State Drive ❖ Replace components without interrupting operation ➢ Hot swapping storage ➢ …

Fig. 3. Creating two different coding trees while reusing a set of open codes

However, developing categories can sometimes require textual interpretation. While the category development of the object oriented code tree can largely follow the natural hierarchy of technical specifications (e.g. we group the code "Solid State Drive" under the category "Secondary Storage"), for the domain goal code tree one needs predominantly interpretive coding.

For example, for the domain goal tree we found the following qualitative data to indirectly refer to the code "Solid State Drive": "...eliminating all moving parts, including rotating hard drives" [22]. Thus, we developed a category "minimize moving components" and grouped the code "Solid State Drive" under this. Then, in coding a different - later - technical specification, this category and

the link to "Solid State Drive" was consolidated by interpreting the snippet "No rotating parts (except for hard disk drive option)" [23].

Finally, note that by means of such coding we created two coding trees while reusing open codes of the technical specification (this reuse was especially well supported by the software tool that we used for our coding effort). For example, the open code "Solid State Drive" is present in both an object oriented diagram and a domain goal diagram. It is just that, when developing the coding trees, the categorization of respective trees differs: in an object oriented tree, "Solid State Drive" is part of a category "Storage Type", whereas for the domain goal tree it is part of the category "minimize moving parts".

Memoing. While coding, GT recommends to write down analytic notes called memos [3]. This forces one to (1) make explicit speculations that emerge while thinking about the qualitative source one is coding, and (2) to keep track of how ideas evolve over time [6, p. 118]. With memoing, one records ideas that emerge during coding of the initial data set, which inspire the coding of additional data. In turn, this helps one to mature the conceptualizations in the coding trees.

EXAMPLE: Memoing has mainly aided us in structuring our understanding while creating the domain goal coding tree. For example, while coding the first qualitative documents, we created the following memo for hot swapping of storage devices: "Exchange component without interrupting the system at hand. It seems to increase serviceability/system maintenance operations". The latter part of this (admittedly rough) memo speculates at maintenance being a theme, prior to maintenance actually becoming a category in its own right.

Later the category becomes important while coding that a front panel provides "easy access to hardware", thus leading to "easier maintenance" (according to the advertising text). Looking back at the notes, one sees common themes emerge, leading one to (1) define maintenance as a category in its own right, and (2) to aggregate under the category "maintenance" the categories "easy access to hardware" and "exchange component without interrupting operations".

Theoretical sampling. This sampling method means that one decides what additional data to analyze next based on the conceptualization of qualitative data already gathered [6, p. 143]. As such, theoretical sampling is exemplary for GTs idea of staying close to a data set, rather than following an existing theory.

EXAMPLE: In our data selection, the newly developed categories did not lead us away from coding substation PC specifications as originally intended.

Theoretical saturation. One continues coding until theoretical saturation, i.e. until no new insights emerge while coding further qualitative data. This means that variation and commonality between the categories in qualitative data is accounted for, and importantly: that a consistent explanatory story can be told on the basis of the developed conceptualization [6, p. 197].

EXAMPLE: we analysed substation specifications until no new insights were gained from coding further specifications. This meant that elements of both the object-oriented coding tree and domain goal coding tree were stable.

For the domain goal coding tree, additionally a stopping criterion is that it can be used to "tell a consistent explanatory story"(cf. [6, p. 197]) about the hardware of a substation PC, especially in terms of what sets this apart from "regular" PC hardware. In this case, a substation PC was found to have domain goals such as continuity of operations, and minimal maintenance.

4.2 Step 2: Develop DSML with Coding Trees

In this step, we develop our diagrams for DSML design from the coding trees that result from the GT effort.

Mirroring the two coding trees developed in sect. 4.1, we develop two diagrams for language design: (1) an object-oriented model, developed in terms of a UML class diagram [20], depicting an abstract syntax, and (2) a goal model, created in the Goal Requirements Language (GRL, [1])). As stated in Sect. 4.1, the goal model complements the abstract syntax by providing a domain understanding, and explicating the key goals behind the abstract syntax.

For the domain goal diagram, we turn leaf-level codes of the domain goal code tree into resources, and use more abstract categories to develop a goal structure. Furthermore, we ask what actors actually state the domain goals.

For the object oriented diagram we have to decide, for each element of the tree, whether to turn it into a an element of a class diagram (class, attribute, constraint, generalization, etc.). Actually, part of the hard work has already been done during the coding effort. Nevertheless, to support this decision we offer the following commonsense guidelines: (1) categories form classes and enumerations. Here, enumerations refer to datatypes with a fixed set of values [20, p.173]. Once defined, the enumeration allows one to restrict the values one can select for the datatype. E.g., to restrict the values of the datatype "SecondaryStorageType" to "SSD" or "HDD", (2) codes at the leafs of the coding tree form attributes and enumeration elements. Furthermore one of course decides on the attribute type, (3) one category subsuming another forms a generalisation/specialisation relation, (4) redundant leaf codes in categories that are subsumed by other categories form attributes of the subsuming category. Finally, we identify (5) constraints by reviewing what codes would constrain an object-oriented model, rather than specifying it further. The idea is that we develop constraints in part from domain goals captured in the domain goal model. We can do this due to the close relation between the domain goal model, and the object oriented model (recall Fig. 2).

EXAMPLE: Figs 4 and 5 depict, respectively, the domain goal and object oriented models of a substation PC. For reference, compare these to the excerpt coding trees developed in Fig. 3.

For the domain goal model, we see how the leaf level codes of the domain goal coding trees have been used to develop resources, such as "Hotswapping power

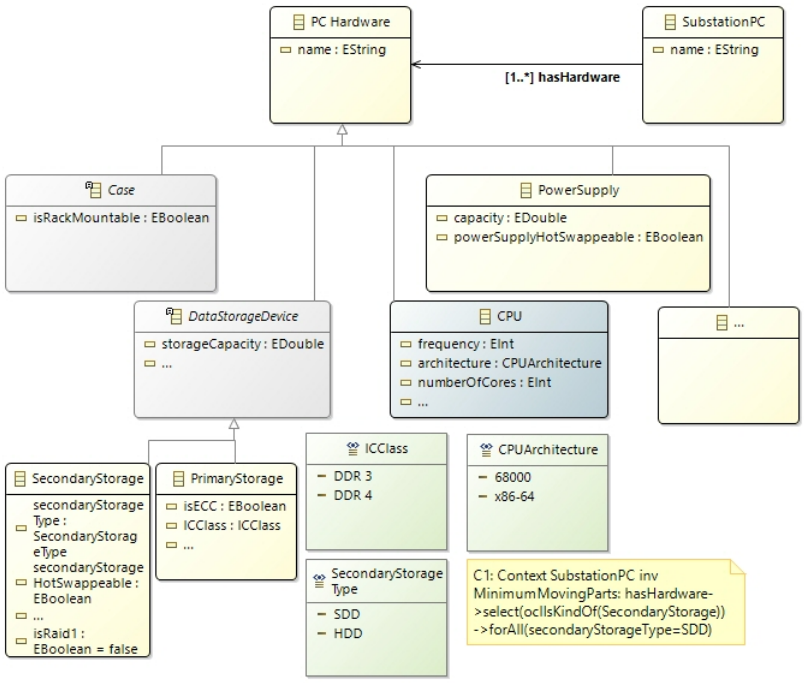

Fig. 4. Excerpt of the substation PC Object Oriented Diagram, cf. UML [20]

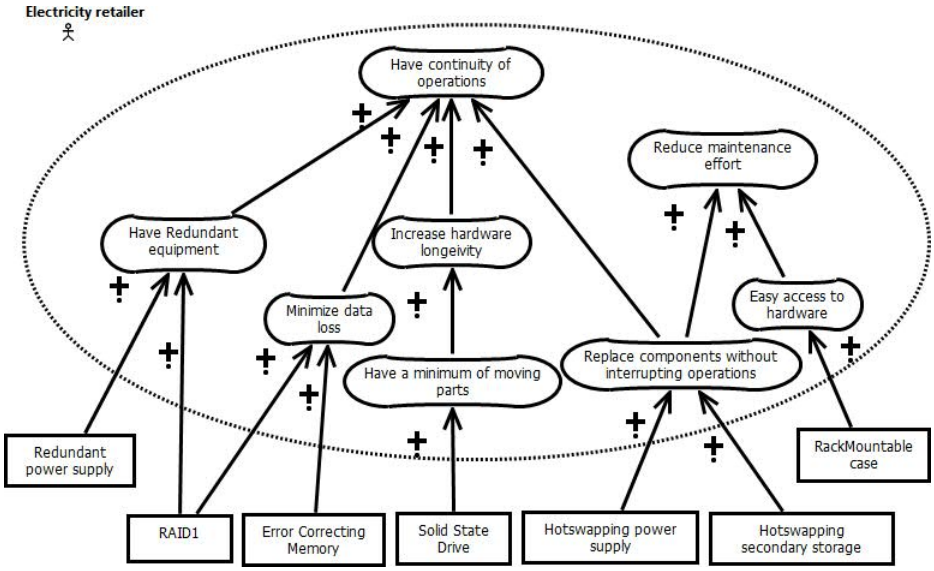

Fig. 5. Excerpt of the substation PC Domain Goal Diagram, cf. GRL [1]

supply", and "Solid State Drive". Furthermore, mirroring the coding tree we see how these resources contribute to achieving domain goals: "Replace components without interrupting operations" for the resource "Hotswapping power supply", and "Have a minimum of moving components" for the resource "Solid State Drive".

For the object oriented model, we develop leaf level codes of the object oriented coding trees into attributes and enumeration elements, being part of categories that form classes/enumerations respectively. For example, we see how harddrive and solid state drive become part of the enumeration "SecondaryStorageType", while the attribute "isRaid1:Boolean" is developed from the code "Raid1". Finally, of note is the definition of constraints. For example, consider the constraint C1. It states that all enumeration elements of "SecondaryStorageType" must be set to SSD, in case "MinimumMovingParts" from the domain goal model is set to true.

Note that the class diagram depicts a hierarchical set of features rather than faithful reproduction of a PC architecture. The reason for this is two-fold: (1) substation PC specifications are presented as taxonomies (so: without specific relations between the different components). Our conceptualization is a faithful representation of this. (2) While we could have coded regular PC architectures to develop relations between the concepts, languages for PC configuration have already been researched at length (see, e.g., [12]). Our abstract syntax can be perceived as a complement, adding extra attributes (such as ECC memory) and constraints (such as redundancy of a power supply) that set a substation PC apart from a regular PC.

4.3 Lessons Learned

Lesson 1: GT is a suitable instrument for synthesizing, in a structured manner, domain specific knowledge into an abstract syntax

Even for the relatively focused domain of a substation PC, we found a large variety between the coded specifications, emphasizing commonalities and differences in PC configuration options (e.g., the option to have ECC memory to minimize data loss). GT has helped us to systematically synthesize this diverse set of technical specifications into an abstract syntax.

Lesson 2: A domain goal model is a useful complement to an object oriented model

We find that creating a goal model in addition to an abstract syntax helps us to (1) better understand a domain (why the abstract syntax has certain features), and to (2) test conformance of the abstract syntax to domain goals.

Also, creating a goal model seems in line with the explorative spirit of GT: it is aimed at gaining a better domain understanding. By merely coding technical specifications, one does not fully develop an idea of what is important to a domain. For example, in coming to grips with substation automation, a goal model explicates *why* an abstract syntax has features such as a redundant power supply, and a solid state drive.

Lesson 3: GT requires a notable time investment

The systematic qualitative analysis of GT (coding, memoing) implies a considerable time investment. Thus, prior to using GT for DSML design, the implied time investment needs to be weighed against its prospective benefits.

To reduce time investment, we additionally foresee combinations with scenario-driven approaches, whereby scenario analysis ad interim the qualitative analysis can help to focus the qualitative analysis effort.

5 Related work

Similar to goal models, for DS(M)L design feature diagrams can describe commonalities and variabilities of a domain [17][26, p. 521]. In feature diagrams, one specifies an abstract feature (for example "browsing") into more detailed ones (for example "get" or "post" features for "browsing") using logical operators such as (X)OR, AND. The resulting "configuration tree" of features can be used to reason about valid sets of features, that can be subsequently translated into valid combinations of model elements [26, p. 523].

Importantly however, the capture of domain features has largely been supported by top-down scenarios. For example, [25, p. 93] suggest user stories to elicit features. A systematic grounding of feature models in qualitative domain data is lacking.

In [7] GT is proposed for the design of conceptual modelling languages. However, this is an (early stage) position paper; while it discusses the idea of using GT as a means to develop an empirically grounded conceptual modelling language, it provides no specifics on how this should be actually done. For example, in terms of choosing between Glaserian and Straussian GT, while this has a significant influence on how one proceeds. Similarly [15] suggests the use of GT for grounding a formal ontology in a qualitative dataset. However, as also admitted by [15] only partial end results of the GT are provided: a set of open codes with, for each, the number of occurrences in the studied dataset.

The closest to our work comes [27], who has applied GT to develop a class diagram of blade servers with the help of Grounded Theory. Particularly [27] uses GT to synthesize technical specifications, such as different types of harddisks, or different types of RAM memory. However, by focusing on aggregating technical specifications, no deeper domain understanding is gained of what is actually a Blade Server. Thus, while [27] succeed in systematically deriving detailed technical specifications with GT, arguably they forego one of the key tenets of doing explorative research: to better *understand* the phenomenon at hand. The same holds true for [11], who has used GT for requirements analysis. While [11] succeeds in developing a class diagram summarizing the most important concepts of the system to be developed, a deeper understanding of the reasoning behind them (in line with the explorative nature of GT) remains implicit.

6 Concluding Outlook

In this paper, we showed how Grounded Theory (GT) can be used for DSML design. Using a case study from the smart grid domain, we discussed how GT is useful for grounding two DSML design diagrams in qualitative domain data: (1) an object oriented diagram, providing the abstract syntax of the DSML as synthesized inductively from qualitative domain data, and (2) a domain goal diagram which, in line with the explorative spirit of GT, provides a grounded understanding of the domain goals behind the abstract syntax.

For further work, we first of all work towards what we term *metamodel provenance*. This means that we further establish an explicit trace between features in an abstract syntax, the domain goals behind it, and the coded fragments of the qualitative sources on which the domain goals are based.

For instance, such metamodel provenance allows us to selectively "switch on and off" elements of the abstract syntax or goals in the domain goal model, and see how this influences the domain goal model respectively object oriented model. Here, an interesting starting point is work on feature models, and how these relate to actual language design considerations. Also we aim to further capitalize on GRL, whose supporting software tool JUCMNav[2] actually supports automated reasoning on goal satisfaction. Given relations between goals, and their respective importance, the impact of satisfying/denying leaf level goals can be propagated towards calculating satisfaction/denial of top-level goals. This can contribute to establishing automated reasoning about satisfaction of goals desired by domain actors, and the impact this has on features of an abstract syntax.

Second, we aim at further developing domain goal models by making explicit the context variables under which a certain domain understanding is relevant (in terms of geography, timeframe or otherwise). At the very least such context is relevant for our ongoing case study in the smart grid, where many concerns that one wants to express depend on context (e.g., regional differences in regulation). In this light constructivist grounded theory is a particularly promising research strand. It makes explicit such context variables by means of a conditional/consequential matrix [6, p. 95], and situation maps [4]. Third, we wish to explore how conceptual modelling can be useful for GT. In this paper we focused GT as a complement to DSML design, however we suspect that the relation can be mutually beneficial. For example, while remaining faithful to GT one can use a domain goal diagram during coding tree development, to help structure codes into categories rather than developing the domain goal model ex-post.

Finally we intend to gain experiences with using GT for creating DSMLs in domains wherein the *social* aspect plays an important role, for example by interviewing domain experts and thus gaining insights into how a particular phenomenon is interpreted. In the Smart Grid domain, this concerns for instance regulatory or valuation aspects. We aim at this because, while the used case study was sufficiently non-trivial to act as a first experiment, in the end it boiled

[2] http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome

down to the analysis of technical specifications (with the exception of a final informal feedback round with a domain expert). As such GT's claimed ability to uncover how domain stakeholders interpret a particular phenomenon remains under explored.

Acknowledgements. The author would like to thank Qin Ma and Monika Kaczmarek-Heß for their valuable feedback.

References

1. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.: Evaluating goal models within the goal-oriented requirement language. International Journal of Intelligent Systems 25(8), 841–877 (2010)
2. Bruinenberg, J., Colton, L., Darmois, E., Dorn, J., Doyle, J., Elloumi, O., Englert, H., Forbes, R., Heiles, J., Hermans, P., et al.: Cen-cenelec-etsi smart grid co-ordination group smart grid reference architecture. Tech. rep. (2012)
3. Charmaz, K.: Grounded theory as an emergent method. In: The Sage handbook of qualitative research, pp. 155–172. The Guilford press (2008)
4. Clarke, A.: Situational analysis: Grounded theory after the postmodern turn. Sage (2005)
5. Cooney, A.: Choosing between glaser and strauss: an example. Nurse researcher 17(4), 18–28 (2010)
6. Corbin, J., Strauss, A.: Basics of qualitative research. London: Sage, third edition edn. (2008)
7. Feller, J., Finnegan, P., , Nilsson, O.: Applying theory-building techniques to the design of modelling languages. In: European Confernce on Information Systems (ECIS). AISNet (2009)
8. Frank, U.: Domain-specific modeling languages: requirements analysis and design guidelines. In: Domain Engineering, pp. 133–157. Springer (2013)
9. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling 13(3), 941–962 (2014)
10. Gephart, R.P.: Qualitative research and the academy of management journal. Academy of Management Journal 47(4), 454–462 (2004)
11. Halaweh, M.: Using grounded theory as a method for system requirements analysis. JISTEM 9(1), 23–38 (2012)
12. Heise, D.: Unternehmensmodell-basiertes IT-Kostenmanagement als Bestandteil eines integrativen IT-Controllings. Logos, Berlin (2013)
13. Ingolfo, S., Jureta, I., Siena, A., Perini, A., Susi, A.: Nomos 3: Legal compliance of roles and requirements. In: International Conference on Conceptual Modeling. pp. 275–288. Springer (2014)
14. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design guidelines for domain specific languages. arXiv preprint arXiv:1409.2378 (2014)
15. Lamp, J., Milton, S.K.: Grounded theory as foundations for methods in applied ontology. In: QualIT 2007: Qualitative research: from the margins to the mainstream abstracts and papers. pp. 1–13. Victoria University of Wellington (2007)
16. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: A survey. IEEE Transactions on Software Engineering 39(6), 869–891 (2013)

17. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. Software Engineering [SEN] E 0309 (2003)
18. Miles, M.B., Huberman, A.M.: Qualitative data analysis: An expanded sourcebook. sage (1994)
19. Niesten, E., Alkemade, F.: How is value created and captured in smart grids? a review of the literature and an analysis of pilot projects. Renewable and Sustainable Energy Reviews 53, 629–638 (2016)
20. Object Management Group: The OMG Unified Modeling Language (OMG UML), version 2.5. Tech. rep. (2015)
21. Razavian, M., Gordijn, J.: Consonance between economic and IT services: finding the balance between conflicting requirements. In: REFSQ pp. 148–163. Springer (2015)
22. Selinc: SEL-3355 Computer. Tech. rep., last accessed on 20-04-2017
23. Siemens: SICAM - Substation Automation and RTUs. Tech. rep., last accessed on 20-04-2017
24. Suddaby, R.: From the editors: What grounded theory is not. Academy of management journal 49(4), 633–642 (2006)
25. Villanueva Del Pozo, M.J.: An agile model-driven method for involving end-users in DSL development. Ph.D. thesis, Universidad Politecnica de Valencia (2016)
26. Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C., Visser, E., Wachsmuth, G.: DSL engineering: Designing, implementing and using domain-specific languages. dslbook. org (2013)
27. Zwanziger, A.: Reconstructing the blade technology domain with grounded theory. In: Advanced Information Systems Engineering Workshops: CAiSE 2011 International Workshops, London, UK, June 20-24, 2011. Proceedings. pp. 187–196. Springer (2011)