Evaluating a Graph Query Language for Human-Robot Interaction Data in Smart Environments

Norman Köster[1], Sebastian Wrede[1 2], and Philipp Cimiano[1]

[1] Cluster of Excellence Center in Cognitive Interactive Technology (CITEC)
[2] Research Institute for Cognition and Robotics (CoR-Lab),
Bielefeld University, Bielefeld Germany
{nkoester,swrede,cimiano}@techfak.uni-bielefeld.de,

Abstract. Solutions for efficient querying of long-term human-robot interaction data require in-depth knowledge of the involved domains and represents a very difficult and error prone task due to the inherent (system) complexity. Developers require detailed knowledge with respect to the different underlying data schemata, semantic mappings, and most importantly the query language used by the storage system (e.g. SPARQL, SQL, or general purpose language interfaces/APIs). While for instance database developers are familiar with technical aspects of query languages, application developers typically lack the specific knowledge to efficiently work with complex database management systems. Addressing this gap, in this paper we describe a model-driven software development based approach to create a long term storage system to be employed in the domain of embodied interaction in smart environments (EISE). The targeted EISE scenario features a smart environment (i.e. smart home) in which multiple agents (a mobile autonomous robot and two virtual agents) interact with humans to support them in daily activities. To support this we created a language using *Jetbrains MPS* to model the high level EISE domain w.r.t. the occurring interactions as a graph composed of nodes and their according relationships. Further, we reused and improved capabilities of a previously created language to represent the graph query language *Cypher*. Lastly, in a third language we compose the other languages, extend them where necessary, perform the required model-to-model transformations and finally perform the desired artifact generation. As a result, we present the *EISE Query-Designer*, a fully integrated workbench to facilitate data storage and retrieval by supporting and guiding developers in the query design process and allowing direct query execution without the need to have prior in-depth knowledge of the domain at hand. To validate our approach we are currently conducting an usability experiment to quantify the advantage of using the proposed approach workbench in comparison to a baseline environment. In this paper we report in detail on the study design, execution, first knowledge gained from our experiments, and lastly the lessons learned from the development process up to this point.

1 Introduction

Smart home technology is gaining more and more popularity and becomes increasingly widespread. The most prominent implementations target support for private households and are available in various complexities from a full system, such as a *KNX*[3] system or a *Apple Home Kit*[4], to rather simple personal assistants, such as *Alexa*[5] or the *Amazon Dash Button*. Beyond the deployment of smart home technology in private homes, one can observe an increased adoption in elderly care settings. Work in this area often further additionally incorporates personal robots to support humans in their daily living and provide an embodied interaction for them[1][2]. Our laboratory setup, the Cognitive Service Robotics Apartment (CSRA), provides such an embodied interactive smart environment (c.f. Figure 1)[3]. It is a fully equipped apartment that is extended with various sensors (e.g. depth sensors, cameras, capacitive floor, light/temperature sensors) and actuators (e.g. screens, colorable lights, audio). Besides two virtual agents which allow users to interact verbally, there is a bi-manual mobile robot named *Floka* operating autonomously within the apartment that allows embodied interaction.

Fig. 1: An example picture from within the CSRA from the living room showing an interaction with Floka in the apartment.

The Cognitive Service Robotics Apartment is used to develop new smart home technology systems as well as to study man-machine interaction in the context of smart environments. One central aspect of the CSRA project is concerned with interaction relevant data/knowledge storage, retrieval and transfer between agents and/or robots. Additionally to the robot eco-system, this environment consists of multiple devices that provide data (motion sensors, cameras, microphones, etc.), software components that generalise (person tracking, situation recognition, etc.), and other components that all employ different (and

[3] https://www.knx.org
[4] https://www.apple.com/ios/home/
[5] https://developer.amazon.com/alexa

often multiple) protocols (KNX, RSB, REST, etc.). Given the afore mentioned amount of sensors and actuators employed, there is hence a large amount of data of various modalities available which needs to be stored in a fashion that it can be readily queried by application developers. Further, with the goal of long-term applications the requirements for storage, retrieval and useful incorporation in applications become a non trivial tasks - even for the application developers themselves. The developers creating applications in this domain need to have detailed knowledge with respect to the different underlying data schemata, semantic mappings, etc. In addition, extensive knowledge about the query language used by the storage system (e.g. SPARQL, SQL, or general purpose language interfaces/APIs) is a requirement and depends on the chosen storage solution. While database developers are familiar with the required details, application developers typically lack the specific knowledge to efficiently work with complex database management systems.

Applying model-driven software development (MDSD) techniques can provide a variety of advantages, for instance the ability for code generation, analysis/checking, or platform independence [4]. A core advantage for the application in the EISE domain is to give application developers feedback at query design time rather than after design and execution (e.g. current query outcome, expected query run duration, possible query improvements, etc.). Further, we think that the extensive modeling accompanying MDSD will be a helpful tool to handle the accidental complexity of the EISE domain and facilitate the task of data exploration and querying for application developers and end and end usersusers. Hence we applied a model-driven approach to create a data system to be employed in the domain of embodied interaction in smart environments with the primary goal to support and ease data retrieval. As a basis for the approach we previously presented an ontology for modelling human machine interaction in smart environments [5]. Using this declarative specification of the EISE domain as a starting point we created multiple external domain specific languages (DSLs). Generally there are clear benefits of DSL applications, such as increased productivity, quality, validation and verification, and lastly productive tooling [4]. In our application domain the latter can especially provide developers with helpful functionality such as static analysis, code completion, visualisations, or debugging at design time.

In particular, our model-driven approach provides implemented artifacts (e.g. a specific IDE and access APIs) to support the query design and execution for application developers. One central tool in this context is the *EISE Query-Designer*, a full IDE that allows developers to design and execute queries against the database setup within the fully integrated CSRA environment. This tool is the result of a composition of multiple individual models designed independently following a model-driven software development approach. While there are many direct advantages of such a tool for developers (e.g. reduction of complexity, static analysis, etc.), we still think there is a need for proper evaluation of the usefulness and usability for developers. In this paper we hence report on our approach for evaluating and quantifying the advantage of this specific IDE.

The remainder of this paper is structure as follows. In Section 2 we briefly describe the created and reused languages and solutions as well as their composition that allows to produce a standalone IDE. In Section 3 we present a detailed description of the study design and implementation we plan to use to evaluate our work and present preliminary results from a small pilot study in Section 4. Section 5 then discusses the results and lessons learned during the development process. Lastly, after giving a short overview about related work in Section 6, we end with a brief conclusion in Section 7.

2 Language Modeling

In the following we briefly describe the modelling approach and present the resulting implementations. We chose *Jetbrains MPS* - one of the most feature rich integrated DSL development platforms - over other tools (e.g. Eclipse Xtext[6]) to implement our work for the following reasons[6][7]. From our point of view the general design of languages is supported well within *MPS* as it provides an integrated support for the development of structure, syntax, type systems, IDE, and model-to-model transformations. The differentiation of languages, solutions (more specifically: runtime solutions and build solutions), and their interoperability allow for complex but yet flexible and easily extensible constructs. Further, *MPS* supports the generation standalone IDEs and specific workbenches tailored to individual specific requirements. Another very important feature is the possibility to easily have multiple projections of our language(s) allowing us to design different views for various roles in the development cycle. For example, a database developer will have to make changes to the lowest levels of the modeling using read-write queries while an application developer only needs to execute simple read-only queries without write access to the database. This difference can be addressed by providing multiple projections within the same artifact or alternatively by generating different role specific artifacts based on the individual needs.

We make explicit use of language extension and language embedding (meant as a special case of language reuse) to model the domain in our framework (c.f. Figure 2)[4]. Architecturally, we hence separate the framework into a total of three languages and seven solutions (refer to Table 1 for a full listing). We chose this level of abstraction as we intend to expand and add in further functionality to allow for example annotation and grounding for data types, time (intervals), and database back-ends. In this paper we only briefly describe three core languages of the framework: (1) the *SecondLevelInstance* (SLI) language, (2) the *Cypher* language, and (3) the *CypherSLI* language. A full overview is presented in Table 1, which lists all additional currently present languages and solutions alongside their function in our architecture.

[6] http://eclipse.org/xtext
[7] https://github.com/corlab/mps-second-level-instance
[8] https://github.com/corlab/Neo4jCypher

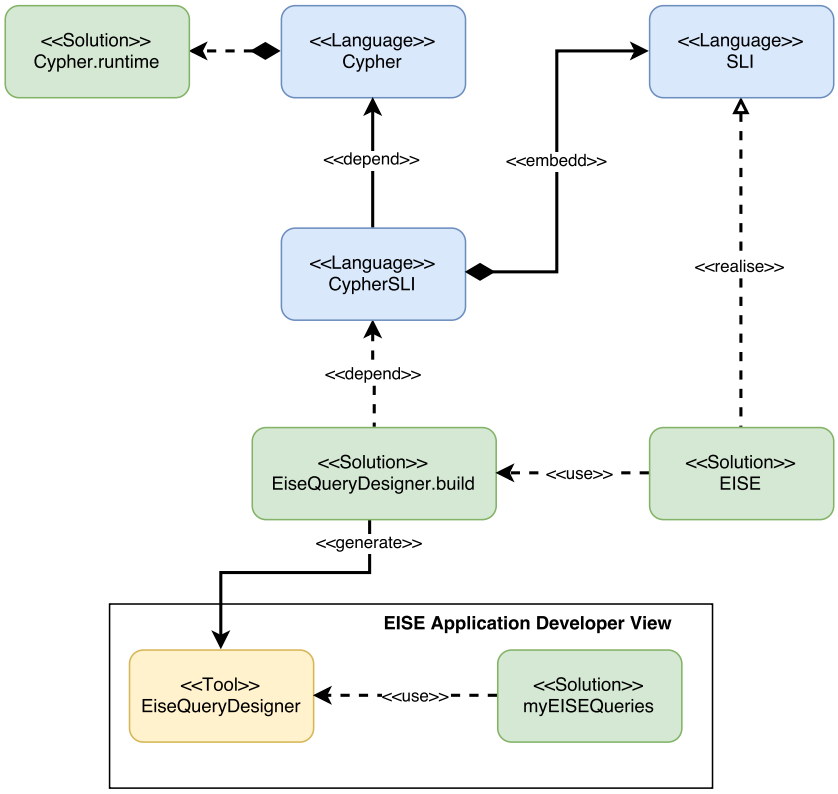


Fig. 2: Minimal architectural overview of the individual *MPS* languages, solutions and their connections required to generate the artifacts for our user study. Central languages are SecondLevelInstance (SLI), Cypher, and CypherSLI.

SecondLevelInstance (SLI)

The Second level instance (SLI) language supports the representation of an application domain as a graph by providing nodes and edges alongside their properties and according data types. The main reason to implement this meta language is the fact that we perceive the application domain description as a dynamic and changing process. When using a *MPS* language to model this sub-domain, domain experts would have to undergo changes in this rather strict and complex environment. Defining this meta language allows us to model the application domain as a solution which can easily undergo variation while being easy to understand and require less detailed knowledge about language design using MPS. We used this language to create the *EISE* solution which models the embodied interaction in smart environments domain.

Table 1: List of languages and solutions alongside their functionality.

	Name	Function
Languages	SecondLevelInstance (SLI)[7]	Language designed to model an application domain by representing their concepts via nodes and edges that compose a graph. The nodes and edges can have properties with base data types.
	Cypher[8]	Language to model the Cypher graph query language. Also adds required concepts and functionality to execute queries against a database from within the language. Further also allows embedding of Cypher constructs within Java.
	CypherSLI	Composing language that extends the SLI and Cypher languages and allows to embed SLI solution instances within Cypher queries
Solutions	EISE	Actual application domain description of the EISE domain that uses the SLI language to model the used entities
	Cypher.runtime	Runtime solution that models the required Neo4j libraries to allow Cypher query execution
	Cypher.build	Build solution to generate a *MPS* plug-in, language-pack and standalone IDE of the Cypher language
	CypherSLI.build	Build solution to generate a *MPS* plug-in, language-pack and standalone IDE of the CypherSLI language
	QueryDesigner.runtime	Runtime solution that models all further needed Java libraries that are not covered by the Cypher.runtime
	QueryDesigner.build	Build solution to generate a general Query-Designer *MPS* plug-in, language-pack and standalone IDE
	EISEQueryDesigner.build	Build solution to generate the EISE Query-Designer, a standalone IDE that fuses the Cypher CypherSLI language together with the EISE domain description solution in order to generate an EISE domain specific workbench.

Cypher

We decided to represent interaction relevant data as a graph and chose Neo4j[9] as our database back end. One important factor is Neo4j's query language Cypher, which is currently trying to gain further adoption with the *openCypher* initiative[10][8]. We think this language provides a good interface for non domain experts to abstract and formulate their queries. The Cypher

[9] http://neo4j.com
[10] http://www.opencypher.org

language hence provides the Cypher graph query language as a *MPS* language. It adds all required concepts to provide the functionality to compose and execute queries against a database. Further, this language also allows to embed Cypher constructs within Java programs. An initial approach was already available as an open source language[11] so we adopted the language and extended it where necessary.

CypherSLI
Describing the application domain as a SLI solution requires us to create a composing language that allows to combine any SLI solution with the Cypher language. The CypherSLI language embeds SLI concepts within the Cypher language and therefore provides this functionality. The language itself is very simple and is mainly concerned with providing the correct scoping for individual concepts of the extended languages. As a result, an according build solution can combine this language with any application domain description defined as a SLI solution. For our study we hence created the EISE Query-Designer that uses the CypherSLI language and embeds the EISE solution concepts (as depicted in Figure 2).

3 Workbench Evaluation

Generally speaking, evaluation of DSLs and IDEs represents a challenge as assessment of their advantage requires to analyse various properties. Especially due to their complexity the evaluation of a full integrated workbench is not as straightforward and may requires long term observation of target users. Case studies which draw lessons learned are an option for evaluation - especially when the benefit is obvious and the user base is large [9]. A good alternative is to follow an iterative testing approach and focus on clear defined metrics (such as lines of code or perceived usability).

To properly evaluate our work we therefore decided to conduct a full user study with potential application developers in an early phase of the development to be able to feed back the results into the development. We let the target audience use the EISE Query-Designer to solve several tasks and compare their performance against a group which uses a baseline default environment (i.e. the default Neo4j web interface). Our primary supporting hypothesis for this study is that programmers formulate queries of various complexities easier and quicker when using the extended Cypher Query language embedded in a specific IDE compared to the normal condition. Secondary, we expect programmers who use the provided tool to exhibit an improved learning curve when solving similar tasks during the study (even though they have to familiarise themselves with the tool first).

[11] https://github.com/rduga/Neo4jCypher

3.1 Study Design

We employ a between-group design with the following two distinct conditions (c.f. Figure 3):

(A) **Normal condition**: Participants have no IDE support. Instead they use the default web interface provided by Neo4j which will allow them to write plain text Cyper queries to solve the four sets. The comprehension tasks present queries to the participants using the default syntax highlighting in the web interface (c.f. Figure 4 (a))

(B) **Extended condition**: Participants will use the EISE Query-Designer and benefit from the IDE support. The IDE provides a custom projection of the Cypher language and incorporates the EISE domain knowledge. Comprehensions tasks are presented directly within the IDE and therefore use the custom projection (c.f. Figure 4 (b)).

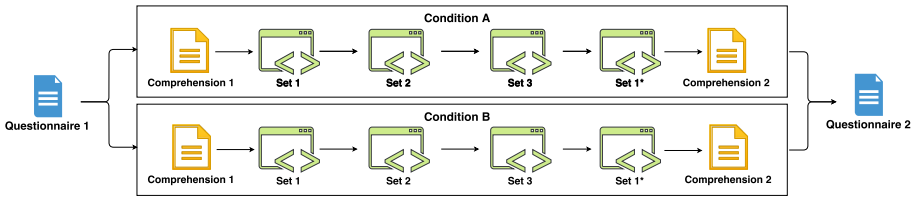
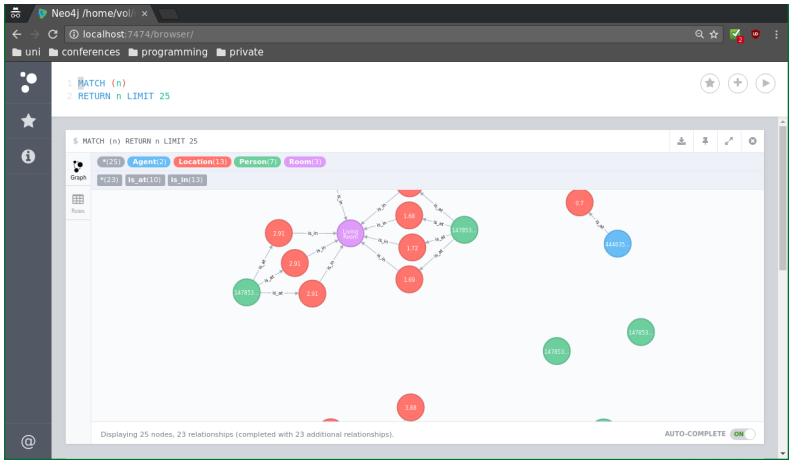


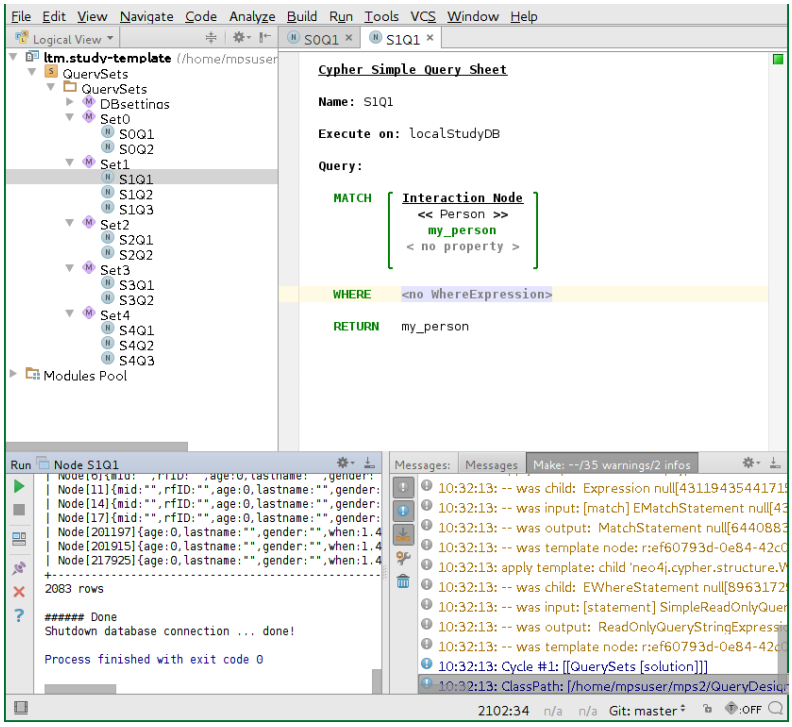
Fig. 3: We chose a between-group study design setup with two conditions: (A) Users use the default Neo4j interface, and (B) users use the EISE Query-Designer. Both conditions have to create and execute the same queries (Set 1 - Set 1*). Before and afterwards users have to describe the meaning of queries presented to them in the comprehension task (Comprehension 1 and 2).

3.2 Tasks

As an introduction and to support the participants during the study we provide three information sheets. First, the "Cypher Cheat Sheet" provides basic knowledge about the Cypher query language and its constructs relevant to compose the queries. This is the same across conditions. Second, a "Tool Sheet" is provided describing the basic usage of the tools used in the according condition (web interface or IDE) together with basic examples, shortcuts, language reference and explanations. Third and last, the "EISE Domain Sheet" contains a graph describing the overall schema of the prepared dataset and is identical across the conditions. It represents a simple visualisation of the EISE solution named in Section 2. Given this material, participants have to solve a total of 6 sets of tasks that are divided into two types: comprehension and work tasks. The following briefly explains each of the task type.



(a) Screenshot of the Neo4j web interface which allows to write and execute the queries. Results can be inspected as a graph or table.



(b) Screenshot of the EISE Query-Designer which allows to write and execute the queries. Results can be inspected as a table at the bottom.

Fig. 4: Screenshot of the tooling used by each condition.

Comprehension Tasks

Comprehension tasks present the participants with example queries in their environment of their condition to investigate their level of understanding of existing queries. The goal is investigate the ability to read and interpret existing queries without prior domain knowledge of the underlying graph structure. A set is each presented each before and after the work tasks (c.f. Figure 2, Comprehension 1 and 2) to investigate our second hypothesis.

Work Tasks

Work tasks are presented to the participants in natural language text and describe a question to be answered by a query (c.f. Figure 5 for an example). The participants have to write a Cypher query based on this provided question. Each question is annotated with a time to give an estimate on the expected required effort. Based on pre-study tests we also provide textual hits as an additional support to elaborate on the query and avoid misunderstandings. Further the expected result is also listed so that participants can easily spot correct and incorrect queries. In total there are four sets of work tasks: Set 1 (3 questions), Set 2 (2 questions), Set 3 (2 questions), and Set 1* (3 questions). The difficulty rises from Set 1 to 3 and each set introduces new concepts of the Cypher language the participants have to use to write successful queries. The last work task (Set 1*) is a modified Set 1 question group with the goal to quantify the expected user learning effects. There is also a Set 0 (omitted in the graphical representation) which is used to present the task-questionnaire procedure to the participants and to foreclose eventual execution errors. For each condition we use the same pre-populated

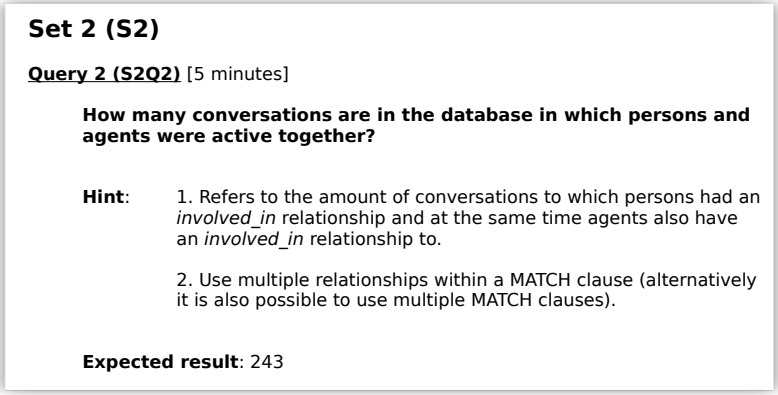


Fig. 5: Exemplary work task as presented to the study participants.

EISE dataset which we gathered within our laboratory setup and refined for

this study. This allows us to formulate a gold standard for each question against which results of the participants can be compared.

3.3 Participant Preconditions

With study participants having to actually use the query language and the according tools for each condition, they have to fulfill certain (rather demanding) criteria. We require participants to have a certain basic knowledge concerning databases and database access (e.g. SQL, SPARQL, etc.). However, they are not required to have strong programming skills as both conditions do not require to write any surrounding source code and allow direct query execution. Their understanding and knowledge about the tools (Neo4j and MPS), query language (Cypher) and overall domain (the EISE domain) for the experiment should however be on a similar level and not differ significantly to allow us to draw conclusions for a representative group. We ensure this by adding according items to our questionnaires asking the participants about their knowledge about the involved elements.

3.4 Questionnaire

To quantify the usability we let users fill in several questionnaires on a separate computer[10]. Between each set of tasks participants have to answer the 6 item *Task Load Index* (TLX) to measure their cognitive load during each set[11]. Besides measuring the TLX metric this also allows us to gather durations for each set of tasks independent from the condition. Once finished, the last questionnaire will ask the user to fill in the *System Usability Scale* (SUS) as well as the *User Experience Questionnaire* (UEQ) in order to assess the tool usability[12][10]. Further properties are recorded on the executing computer allowing us to investigate metrics such as time per task set, key strokes, correct and incorrect queries against the database and as a quality measure a screen recording. The targeted sample size is 15 to 20 participants per condition.

3.5 Expectations

With this given study setup we have certain expectations towards each groups performance. As participants in the extended condition (B) will use the IDE to compose queries we expect them to perform better compared to the normal condition (A). We expect a higher accuracy and lower error rates due to syntax and error checking that is provided by the IDE. A similar assumption we hold is that the overall duration for each task and the keystrokes required will be higher for the normal condition as they receive fewer feedback and support when writing the queries. With the separation in multiple sets we expect participants of condition (B) to learn how to create queries faster - even though they will have to familiarise themselves with a more complex tool. We expect this familiarisation step to be a constant that will impact the initial sets but otherwise will not be present in later more difficult tasks.

Table 2: Average perceived usability of pilot study.

	Metric	Condition (A)	Condition (B)
SUS	Score	75	63.33
UEQ	Attractiveness	1.111	0.944
	Perspicuity	1.417	1.083
	Efficiency	0.917	0.833
	Dependability	1.500	0
	Stimulation	0.833	0.917
	Novelty	-0.083	0.667

4 Pilot Study Results

In a pilot study we applied the study design to a total of six participants (three per condition). Preliminary results of the user SUS and UEQ questionnaires are listed in Table 2. It shows that the SUS score for condition (B) ranks within an average level while condition (A) is slightly above average [13]. Further, we observed that for all UEQ questionnaire based usability metrics the web interface (condition (A)) scores slightly higher with stimulation and novelty being the exceptions. The pilot study results regarding the actual time the participants required per task set is depicted in Figure 6. It shows that the baseline condition

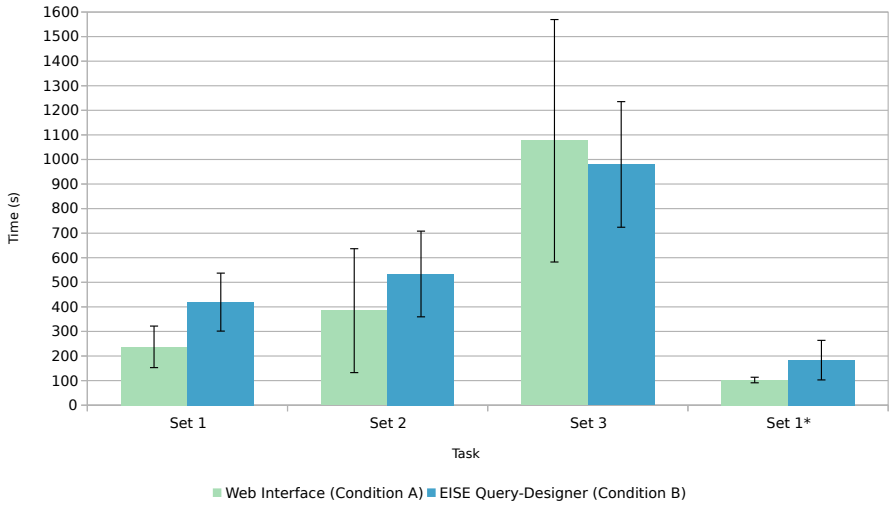


Fig. 6: Average time it took participants to finish each set of tasks.

(A) allowed participants to finish easier tasks (Set 1, Set 2, and Set 1*) faster when compared to condition (B). However, the set with more difficult tasks (Set 3) demanded more time investment form participants in condition (A). An analogous result is observable for the cognitive load participants felt during each set (c.f. Figure 7).

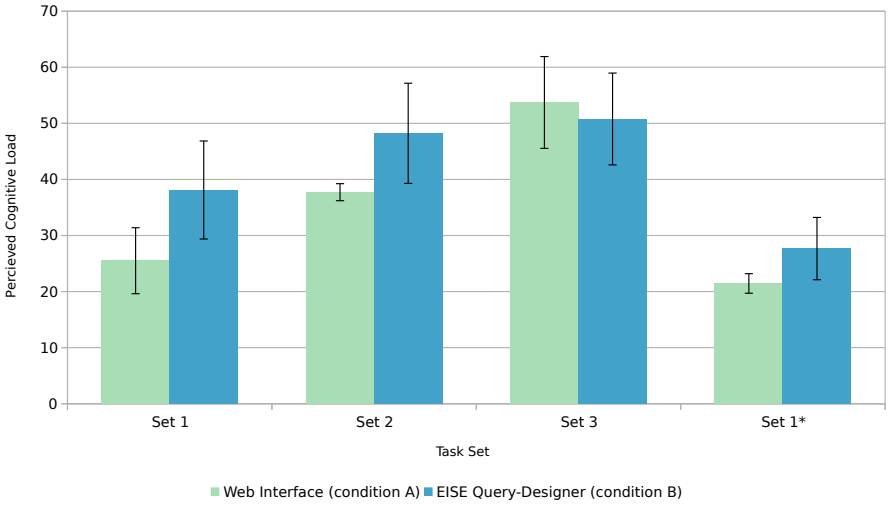


Fig. 7: Average cognitive load participants perceived per task set.

5 Discussion & Lessons Learned

Though the preliminary study has only few participants, its results shows recognisable trends. In the first two sets the participants in the baseline condition (condition A) were faster and had less cognitive load. We think this is due to the steep learning curve projectional editors (i.e. the EISE Query-Designer) have. Participants did not use any comparable projectional editing software before and had to familiarise themselves with the concept of direct abstract syntax tree manipulation. Compared to this the baseline participants could directly begin their work with common source editing. Once participants of condition (B) overcome this initial difficulty we can see that more complex tasks (i.e. Set 3) begin to show the advantages They require less time while at the same time perceive less cognitive load when compared to the base line condition. Further, for all conditions a learning effect seems to be present: In both conditions the participants finish Set 1 (a permuted Set 1) faster than the initial Set 1 itself. The significance of this effect will have to be proven once an adequate sample size is collected.

Qualitatively we realised that presenting participants with an unknown domain, tools and DSLs requires well written introduction material. This leads to

on average 20 minutes that are necessary to fully read the provided material. As a result we had to reduce the amount of tasks per set and removed the comprehension tasks completely to stay within planned 1 hour maximum duration.

The development of the tool and all corresponding individual languages and solutions left us with several lessons learned. Language design is a difficult task - especially with a small team size. This leaves us with a recommendation for good prioritisation of sub tasks in the development life cycle. Feedback from target IDE users is very valuable and their acceptance is influenced by multiple factors of the tool. The provided editor/projection is an important element in this context as it is the first entry point for users. It will impact user performance and acceptance when simple functionalities (e.g. auto completion of simple data types) do not work as expected. These rather marginal properties can overshadow the valuable actual modeling of the domain the tool provides.

Another issue we encountered is concerned with the reuse of existing languages. As mentioned by Voelter, the *DSL Hell* (creating half-baked DSLs instead of reuse) should be avoided and reuse is one a key features of in DSL development [4]. This in mind we reused an existing *MPS* Cypher language and could reduce our workload significantly. However, a reused language can have its own problems which can include possible unfinished design decisions, requirements/dependencies on legacy languages or software, abandonment, or others. To avoid mitigating workarounds in the surrounding new languages the only option is to improve the reused language itself and feedback the improvements. Lastly, along goes the need for language versioning, which is a key discipline to be employed from the start in the language engineering process as it will otherwise hinder the development. With a recent update *Jetbrains* even addresses this issue and provides build-in support in MPS.

6 Related Work

While usability is a well researched and standardised field in software engineering the benefits of specific IDEs or workbenches are often difficult to be evaluated. Improvement claims can be supported either formally, automatically, heuristically, or empirically. Bari et al. therefore proposed an evaluation process for the usability of DSLs that is applied during the development life cycle via various metrics, including questionnaires targeting the subjective measures such as cognitive load or perceived usability [14]. Further, others propose an integrated iterative testing approach that focuses on clear defined metrics [15][16][17]. The core idea is to let evaluation span the entire DSL life cycle by assessing motivation, qualitative interviews, a validation of DSL design, and quantifying benefits. According to Wegeler et al. a mix of quantitative and qualitative criteria is required as simple metrics cannot cover all advantages and risks. However each measure is important and should impact the DSL development process.

Evaluation on created tools in practice with is often carried out with domain experts. For example, Karna et al. used and evaluated their finished developed solution in product development [18]. They let six users familiar with their target

domain develop an application using the created tool and compared the outcomes w.r.t. the three factors of developer productivity, product quality and the general usability of the tooling.

An alternative approach is in the form of providing in depth case study analysis. This is a valid evaluation approach especially when the presented tool already has a big user base that makes extensive use of the provided functionalities. Voelter et al. recently provided an excellent example case study providing great insight into the *mbeddr* platform [9]. This extensive review evaluates the language engineering process using Jetbrains *MPS* as a language workbench and provides valuable lessons learned. From their point of view language modularity while handling the domain complexity is feasible and their conclusion is that useful large scale system design using *MPS* is possible.

Lastly, on a meta layer benchmarks for language workbenches themselves are being researched. In this context the annual Language Workbench Challenge was instantiated in 2011 to provide an opportunity to quantitatively and qualitatively compare approaches [7]. They compared 10 workbenches an provide a great overview presenting the state of the art options for developers.

7 Conclusion and Outlook

In this paper we present our current work to support query design and execution for application developers in the domain of embodied interaction in smart environments. Following a model-driven approach we created multiple *MPS* languages and solutions with the goal to support developers at programming applications that are depending on long term interaction data. We make explicit use of a multitude of features provided by MPS, most notably the generation of a domain specific IDE to be used by application developers. With the difficulty of evaluation of generated IDEs we presented the study design we decided to pursue for an early assessment. First study runs yield promising results, however continued further work and research is required. For example, not all described architectural decisions could be implemented as planned due to time and resource constraints. As a result some functionalities have been implemented in different languages contrary to the original plan.

Further, all languages require polish and fine tuning, especially the entry points for users (i.e. the language editors/projections). Additionally the planned extension points (such as data type mapping) have to be integrated into the architecture. Lastly, we plan to further include the complete approach in the CSRA project and hence create stronger dependencies. Along with this plan goes the idea of a long term study and evaluation of the generated artifacts.

Acknowledgements

This research/work was supported by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG).

References

1. Adair, B., Miller, K., Ozanne, E., Hansen, R., Pearce, A.J., Santamaria, N., Viegas, L., Long, M., Said, C.M.: Smart-home technologies to assist older people to live well at home. Journal of aging science (2013)
2. Cavallo, F., Aquilano, M., Bonaccorsi, M., Limosani, R., Manzi, A., Carrozza, M.C., Dario, P.: Improving domiciliary robotic services by integrating the astro robot in an ami infrastructure. In: Gearing Up and Accelerating Cross-fertilization between Academic and Industrial Robotics Research in Europe:. Springer (2014) 267–282
3. Holthaus, P., Leichsenring, C., Bernotat, J., Richter, V., Pohling, M., Carlmeyer, B., Köster, N., Meyer zu Borgsen, S., Zorn, R., Schiffhauer, B., et al.: How to adress smart homes with a social robot? a multi-modal corpus of user interactions with an intelligent environment. In: Proceedings of the 10th Language Resources and Evaluation Conference. (2016)
4. 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)
5. Köster, N., Wrede, S., Cimiano, P.: An Ontology of Human Machine Interaction Data in Smart Environments. In: SAI Intelligent Systems Conference 2016, IEEE (2016)
6. Voelter, M., Pech, V.: Language modularity with the mps language workbench. In: Software Engineering (ICSE), 2012 34th International Conference on, IEEE (2012) 1449–1450
7. Sebastian Erdweg, Tijs van der Storm, Markus Völter, Laurence Tratt, Meinte Boersma, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, J.v.d.W.: Evaluating and Comparing Language Workbenches. Computer Languages, Systems and Structures **44**(A:2447) (2015) 1–38
8. Neo Technology Inc.: openCypher (2015)
9. Voelter, M., Kolb, B., Szabó, T., Ratiu, D., Deursen, A.V.: Lessons learned from developing mbeddr : a case study in language engineering with MPS. Software & Systems Modeling (2017)
10. Laugwitz, B., Held, T., Schrepp, M.: Construction and evaluation of a user experience questionnaire. In: Symposium of the Austrian HCI and Usability Engineering Group, Springer (2008) 63–76
11. Hart, S.G., Staveland, L.E.: Development of nasa-tlx (task load index): Results of empirical and theoretical research. Advances in psychology **52** (1988) 139–183
12. Brooke, J., et al.: Sus-a quick and dirty usability scale. Usability evaluation in industry **189**(194) (1996) 4–7
13. Sauro, J.: A practical guide to the system usability scale: Background, benchmarks & best practices. Measuring Usability LLC (2011)
14. Barišic, A., Amaral, V., Goulao, M., Barroca, B.: Recent advances in multi-paradigm modeling (mpm 2011). Electronic Communications of the EASST **50** (2011)
15. Wegeler, T., Gutzeit, F., Destailleur, A., Dock, B.: Evaluating the Benefits of Using Domain-Specific Modeling Languages - an Experience Report Categories and Subject Descriptors. In: Proceedings of the 2013 ACM Workshop on Domain-Specific Modeling. (2013)

16. Bariic, A., Amaral, V., Goulao, M.: Usability Evaluation of Domain-Specific Languages. 2012 Eighth International Conference on the Quality of Information and Communications Technology (2012) 342–347
17. Barišić, A.: Iterative evaluation of domain-specific languages. CEUR Workshop Proceedings **1115** (2013) 100–105
18. Kärnä, J., Tolvanen, J.P., Kelly, S.: Evaluating the use of domain-specific modeling in practice. In: Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling. (2009)