Towards a Systematic Method for Developing Meta Attack Language Instances

Simon Hacks
Sotirios Katsikeas
Engla Rencelj Ling
Wenjun Xiong
{shacks|sotkat|englal|wenjx}@kth.se
Division of Network and Systems Engineering
KTH Royal Institute of Technology
Stockholm, Sweden

Jérôme Pfeiffer
Andreas Wortmann
{jerome.pfeiffer|andreas.wortmann}@isw.uni-stuttgart.de
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW)
University of Stuttgart
Stuttgart, Germany

Abstract

The successful deployment and use of domain-specific languages (DSLs) demand that language engineers consider the future organizational context of the languages. Design science research, and in particular action design research (ADR), provide conceptual frameworks for this. Yet, their application to the engineering of DSLs has not been investigated. In this paper, we investigate applying ADR to the development of threat modeling languages based on the Meta Attack Language (MAL), a metamodeling language for the specification of domain-specific attack languages. To this effect, we conducted a survey with experienced MAL developers regarding their activities in the development of MAL DSLs. We extract guidelines and align these, together with established DSL design guidelines, to the conceptual model of ADR. The research presented, aims to be the first step to investigate whether ADR can be used as a basis for the systematic engineering of DSLs.

CCS Concepts: • **Software and its engineering → Domain specific languages**; *Software development techniques.*

Keywords: Software Language Engineering, Action Design Research

ACM Reference Format:
Simon Hacks, Sotirios Katsikeas, Engla Rencelj Ling, Wenjun Xiong, Jérôme Pfeiffer, and Andreas Wortmann. 2021. Towards a Systematic Method for Developing Meta Attack Language Instances. In *tbd*. ACM, New York, NY, USA, 19 pages. https://doi.org/tbd

SLE'21, 17-19 October 2021, Chicago, Illinois

ACM ISBN tbd…$tbd
https://doi.org/tbd

1 Introduction

Cybersecurity continues to be a key concern and a fundamental aspect of information technology (IT) and operational technology (OT) systems [54], from water or energy distribution systems to online banking services. Cyber-attacks on these systems can have severe consequences for individuals and organizations, such as the Ukraine major power outage [9], the WannaCry ransomware attack [41], and the Florida water supply attack [23].

At the same time, it is difficult to assess the security level of IT systems. Despite being challenging, it is necessary to identify all relevant system assets, their weaknesses, and possible mitigations. To proactively deal with security concerns, threat modeling [55, 68] and attack simulations [25] can be used to assess the cybersecurity of systems and make it more difficult for attackers to accomplish their malicious intent. Threat modeling includes holistic identification of the main assets within a system and threats to these assets. These threat models then serve as inputs for attack simulations based on system models to simulate cyber-attacks, identify weaknesses, and provide quantitative security measurements for the system [12, 22]. With the concepts required to model threats being highly domain-specific (i.e., modeling automotive threats leverages concepts different from Industry 4.0 threats), the employed threat modeling languages should be highly domain-specific as well.

Previously, the Meta Attack Language (MAL) [25] was proposed, which serves as a basis to develop domain-specific languages (DSLs) for specific attack contexts and generate attack graphs from the models. MAL is a meta-language featuring generic concepts that need to be instantiated properly for modeling systems, threats, and attacks in different domains. These resulting instances of MAL are domain-specific threat modeling languages, i.e., DSLs. To date, several MAL DSLs have been proposed, such as vehicleLang [32] for modeling cyber attacks on vehicle IT infrastructures, coreLang [31] for modeling attacks on common IT infrastructures, powerLang [17] for modeling attacks on power-related IT and OT infrastructures, SCL-Lang [49] for modeling attacks

on power system substations, and enterpriseLang [69] for modeling attacks on enterprise systems.

However, the development process of these MAL DSLs varies a lot. For example, some of them involved the desired end-users in the development process [19], while some have no stakeholders involved; some validate the instances by creating test cases [32], while some by modeling real-world attacks [17, 69]. To unify the development of MAL DSLs, we propose a software language engineering (SLE) approach to develop MAL instances (i.e., MAL DSLs) using action design research (ADR). This approach aims to systematically collect guidelines and structure them to get a comprehensive overview for the development of MAL instances. To this end, we conducted a survey with experienced MAL developers to understand their approach to developing MAL instances. From these surveys, we extract guidelines and align these, together with established DSL design guidelines, with the conceptual model of ADR. Our approach, thus, is the novel application of ADR to develop DSLs through the lens of MAL instances. As such, it is a first step towards systematically engineering DSLs that we aim to generalize in the future.

The contributions of this paper are threefold:

1. We provide the first systematic ADR-based approach towards the systematic development of MAL DSLs.
2. We provide a comprehensive overview of how DSL guidelines can be used for developing MAL DSLs.
3. We demonstrate our approach on three existing and documented MAL DSLs.

In the following, Section 2, we present the background of this work. Section 3 describes the method of developing our approach. Section 4 describes the approach in detail. Section 5 demonstrates our findings related to the already developed MAL DSLs. Section 6 discusses observations. Section 7 reviews the state-of-the-art. Finally, Section 8 concludes.

2 Background

Following, we give a short introduction to the different foundations that we use within this work. Firstly, we present MAL, a framework for which we design a systematic approach for creating DSLs. Secondly, we give an overview of the basic language engineering concepts, which are of relevance for our work. Finally, we introduce the method of design science research (DSR), which is an often-used approach to create artifacts in information systems research.

2.1 Meta Attack Language

First of all, a MAL DSLs contains the main elements that are encountered on the domain under study, those are called `assets`. The assets contain `attack steps`, which represent the actual attacks/threats that can happen to them.

An `attack step` can be connected with one or more following `attack steps` so that an attack path is created. Those attack paths are then used to create attack graphs

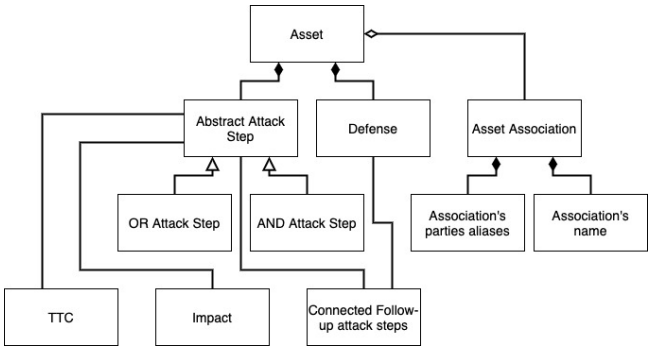

Figure 1. The metamodel of MAL

which are facilitated when the attack simulation is run. An `attack step` can be either of the type OR or of the type AND, respectively indicating that performing any individual parental `attack step` is required (OR) or performing all parental `attack steps` is required (AND) for the current step to be performed. Attack steps of type OR are defined by the symbol | while AND attack steps are defined by the symbol & before their names. Additionally, each `attack step` can be related to specific types of risks.

Furthermore, `defenses` are entities that do not allow connected `attack steps` to be performed if they have the value TRUE which represents them to be enabled. Finally, `probability distributions` can be assigned to `attack steps` in order to represent the effort needed to complete the related `attack step` or the probability of the `attack step` to be possible. This is defined after the name of the `attack step` as it happens on the attack step on line 20 of the exemplary MAL code.

`Assets` have relations between them that are used for the creation of a model, those relations are called `associations` in MAL and are defined by the <- - and - -> symbols. When associations are specified a name for the association as well as cardinalities for both assets should be defined. Inheritance between `assets` is also possible and each child `asset` inherits all the `attack steps` of the parent `asset`. Additionally, the `assets` can be organized into `categories` for purely organization reasons. A diagram of the MAL metamodel can be seen in figure 1.

Listing 1 presents an example of a MAL DSLs. In this example, four modeled assets can be seen together with the connections of attack steps from one asset to another. In the `Host` asset on line 6, the *connect* attack step is an OR attack step while *access* is an AND attack step. Then, the -> symbol denotes the connected next attack step.

For example, if an attacker performs *phish* on the `User`, it is possible to reach *obtain* on the associated `Password` and as a result finally perform *authenticate* on the associated `Host`. In the line 29 to 39 of the example, the `associations` between the assets are defined.

Listing 1. Exemplary MAL Code

```
category System {
  asset Network {
    | access
      -> hosts.connect
  }
  asset Host {
    | connect
      -> access
    | authenticate
      -> access
    | guessPwd
      -> guessedPwd
    | guessedPwd [Exp(0.02)]
      -> authenticate
    & access
  }
  asset User {
    | attemptPhishing
      -> phish
    | phish [Exp(0.1)]
      -> passwords.obtain
  }
  asset Password extends Data {
    | obtain
      -> host.authenticate
  }
}

associations {
  Network [networks] *
    <-- NetworkAccess -->
    * [hosts] Host
  Host [host] 1
    <-- Credentials -->
    * [passwords] Password
  User [user] 1
    <-- Credentials -->
    * [passwords] Password
}
```

For a detailed overview of MAL, we refer readers to the original paper [25].

2.2 Design Science Research

In information systems research, DSR is a popular approach to develop and evaluate designed artifacts. Usually, DSR is attributed to Hevner et al. [20], who distilled seven guidelines as characteristics for this research approach. Mainly, they stress the need for a viable artifact, which addresses a problem relevant to the business. Then, the artifact is evaluated based on the properties of the problem, while ensuring a contribution to research. The entire design and evaluation process needs to follow a rigorous method and equals a search process. Finally, the resulting artifact should be presented to other researchers as well as business representatives.

As these guidelines seemed to be too abstract to effectively steer research, several concrete processes realizing DSR have been proposed. One of the most popular is related to Peffers et al. [46], who propose an iterative six-stepped process: First, the researcher defines the problem. Second, the objectives of a suitable artifact are determined. Third, the actual artifact is designed. Fourth, it is demonstrated that the designed artifact solves the given problem. Fifth, an evaluation is conducted, in which the researcher shows that the new artifact is performing better than existing solutions. Finally, the outcomes are communicated.

Peffers et al.'s process has been criticized for not involving stakeholders sufficiently. Therefore, Sein et al. [52] developed that process further by joining action research (AR) and DSR to ADR. ADR is characterized by a much closer exchange between researchers and stakeholders and mimics the development from the waterfall process to agile methods in software engineering.

3 Method

To design our approach to guide the development of MAL instances, we follow a four-step process: First, we opt for a guiding approach already established in research. Concretely, our approach shall rely on the principle of DSR [20] since we are creating a concrete artifact (a MAL DSL) that will be used in an information systems environment. Over time, various interpretations of DSR have been developed and Venable et al. [61] sketch decision support that helps to choose the best fitting approach. Therefore, they differentiate between objectivist, positivist methodologies and subjectivist, interpretive methodologies. They argue if one expects the designed artifact to be the best solution for a generalized target group that behaves the same, then one should opt for one of the objectivist, positivist methodologies. In our case, we expect that each created language will serve a certain purpose and might not be generalizable to every need [16]. Thus, we opt for a subjectivist, interpretive methodology. Those methodologies are distinguished by the domains they address. Venable et al. [61] state that "If you have a single client that wants to engage in a research undertaking with you. Then choose ADR". As most of the MAL DSLs are developed for or together with a single organization, we decide to adapt ADR [52] for our approach by considering the knowledge of MAL DSL developers and existing research presented next.

Second, we investigate how other research addresses the different stages of ADR. For the stages "Problem Formulation", "Reflection and Learning", and "Formalization of Learning", we consider articles citing the original ADR description [52] explicitly addressing these stages. While we assume that the aforementioned stages are somehow similar for all kinds

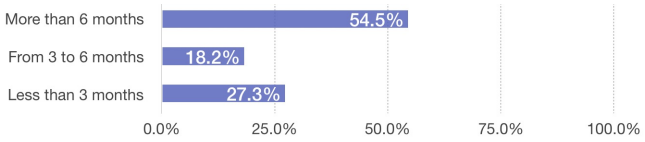

Figure 2. Time needed to develop the MAL language

of designed artifacts, the second stage "Building, Intervention, and Evaluation" is strongly dependent on the artifact(s) to be developed. Consequently, we consider for this stage research on building threat models [3, 11, 36, 51, 58, 65, 70] and creating DSLs []. Next, we follow the conceptual-to-empirical approach of Nickerson et al. [43] based on the identified literature and create categories and labels that could be used to classify the different tasks performed during the creation of MAL instances. This is independently performed by two of the authors, who subsequently discuss their results to come to one set of categories and labels. Doing so, we reduce the subjective influence on the definition of our categories and labels. We used deductive coding [39], which means that the labels are defined before the coding process starts, and therefore the survey responses do not influence the labels that are chosen.

Third, we perform a survey among experienced developers of MAL DSLs to gather information on how they developed the languages. We received eleven fully answered questionnaires. Considering 19 languages that we are aware of [16], this corresponds to a return rate of 63 %, which is satisfying for an online survey [10]. Moreover, we check if the survey is answered for all published MAL instances, which indeed is the case. According to the respondents of the survey, for developing a MAL instance, 72.7% of the respondents have been worked in a team, and 27.3% of them are worked individually. In terms of the time needed to develop a MAL instance, it takes 54.5% of the respondents more than 6 months to develop the language, and 27.3% of them take less than 3 months to develop the language (see Figure 2).

The questionnaire consists of 7 sections with 30 main questions and in total 41 questions since we ask for further details depending on how the person answered the previous question. The first section comprises questions about the MAL developer's background, such as the educational and professional background. The second section asks questions regarding which MAL instance the respondent is referring to when answering the survey. The third section inquires about the purpose of the MAL instance, the fourth section asks about the language engineering approach, and the fifth section about the language artifact itself. Finally, the last two sections comprise validation and maintenance questions.

Since the survey consists of open-ended questions, meaning that the respondents can answer the question in their own words, we need a method of quantitatively analyzing a large amount of free text. The choice was made to use coding, which is a method of assigning labels to text to identify recurring themes [48]. The labels and categories were found in the process described in stage 2. This method is suitable as we aim to find and categorize common or contradicting methods of developing MAL instances.

We split the team of the authors into two groups that work independently to codify the findings of the survey and align it with the stages of ADR. Given the categories and labels from the second step (i.e., reflection on existing research), the answers are classified. Following the empirical-to-conceptual approach of Nickerson et al. [43], we complement the categories and labels if we identify an aspect not covered by the literature. Afterward, the groups discuss their results to come to a unified understanding of the actions in the single stages presented in Section 4. Finally, we reflect our newly designed approach on the documented development processes of three MAL DSLs in Section 5.

4 The Approach

Next, we present the approach itself (cf. Figure 3), which we derived following our method presented before.

4.1 Stage 1 - Problem Formulation

The first stage of ADR is about the formulation of the problem to be solved. A problem can typically be identified either by practitioners or by researchers of the domain. In this first step of the ADR process, the initial scope of the problem, the roles and scope of each stakeholder, and the initial research questions are defined.

The problems formulated in this stage can be assigned to one of the following two problem principles: i) practice-inspired research, and ii) theory-ingrained artifact. In the first principle, the problem that is perceived by practitioners should be induced to a class of problem(s), a process that creates an opportunity for new knowledge to be generated. The problem identified will then be used to exemplify that knowledge. In the second principle, it is suggested that all the artifacts that are created to solve a problem are based on some previous theory or theories. The uses of such prior theories can be: 1) to structure the problem, 2) to identify solution possibilities, and 3) to guide the design of the artifact.

The labels that were detected from the reviewed literature and correspond to the formulation of the problem can be categorized into three discrete categories, as presented below.

G-1.1: Problem definition The first category, which elaborates on encountering and realizing the problem, is comprised of the following dimensions. Those dimensions correspond to the labels we have detected from the literature. First, performing a systematic literature review (SLR) is one way of detecting and formulating a problem [1, 15, 26, 38, 44].

[0]MAL Survey https://forms.gle/Wuv5sJgqZSctgP4LA (Accessed 1 June 2021)

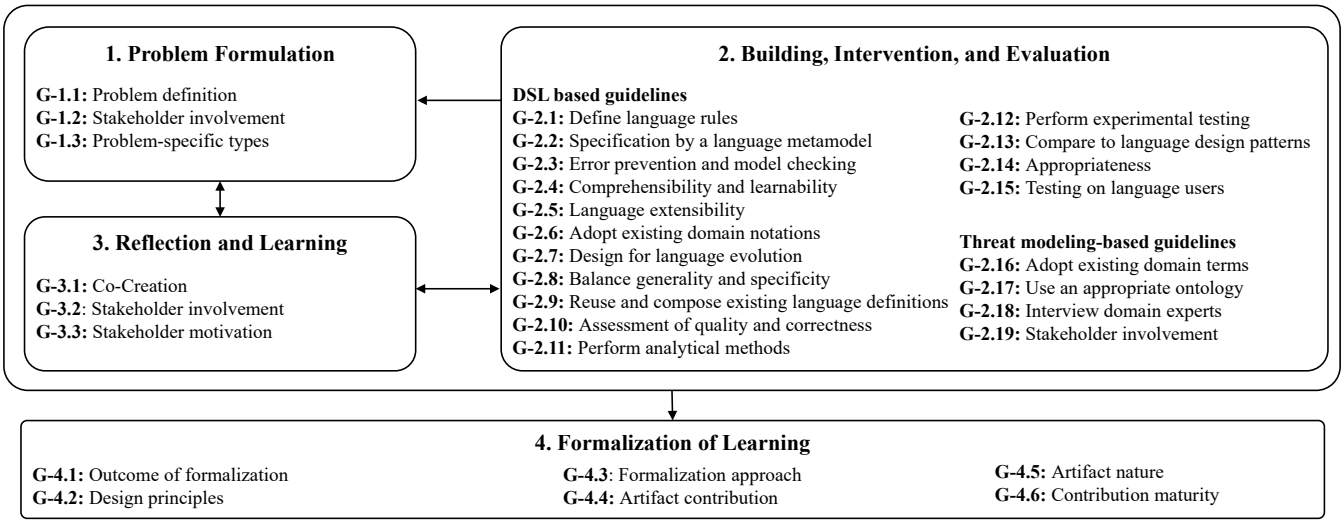

Figure 3. ADR Method: Stages and Guidelines (adapted from Sein et al. [52])

Other possible ways are through systematic empirical investigations, focus groups meetings, or expert interviews [38, 44]. A problem can also be encountered during the typical operations of an organization [38]. Finally, cause-effect diagram modeling can also lead to the detection of a problem [1].

G-1.2: Stakeholder involvement The next category of labels is related to the type of the stakeholders and their involvement in the problem formulation process. The possible involved stakeholder types are: 1) researchers, 2) end-users, and 3) practitioners [15, 38]. Then, the involvement of the stakeholders can typically be achieved via participation in the following detected dimensions/labels: 1) expert interviews [38, 44], 2) focus groups [38, 44], 3) surveys [44], and 4) status seminars [44].

G-1.3: Problem-specific types In the last category, some labels for the categorization of the problem itself and the research gap were identified in the literature. First, the problem itself can either be categorized as an abstract problem or an instantiated problem [1]. Then moving on to the property of the problem, we can differentiate two types of research: theoretical gap or design gap [1]. In the case of a theoretical gap, theoretical knowledge is missing that is needed as justification for the design of the artifact. On the other hand, the design gap refers to knowledge that is missing or not yet validated and has to do with the created artifact rather than the theoretical foundations.

4.1.1 Application on MAL DSLs.

The survey that was performed with MAL DSLs developers showed that in most cases the definition of the problem that the developed MAL DSL tries to solve was identified through interviews with experts and by performing an SLR. In the question about who the different stakeholders are, as shown in Figure 4, the most common answer was that they come from the industry, and are therefore practitioners of the field, while the second most popular reply was that they are researchers in the field. In some other cases, sponsors are also considered as stakeholders, but then they have minimal influence in the design process. The involvement of stakeholders in the design process was most commonly achieved through focus group meetings and by performing surveys (see Figure 5). Finally, the participants stated that the problem was originating from a gap in the design and not a gap in the knowledge, while the percentage of the instantiated types of problem is larger than the abstract types, as shown in Figure 6.

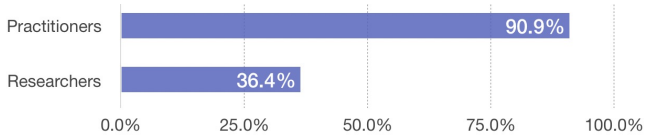

Figure 4. Stakeholder involvement (who)

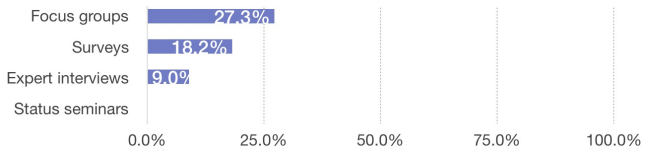

Figure 5. Stakeholder involvement (how)

4.2 Stage 2 - Building, Intervention, and Evaluation

The second stage of ADR builds upon the problem formulation tasks of stage 1. On these premises, stage 2 employs an iterative design cycle with three steps to achieve the realized design of the artifact. The three steps are 1) building

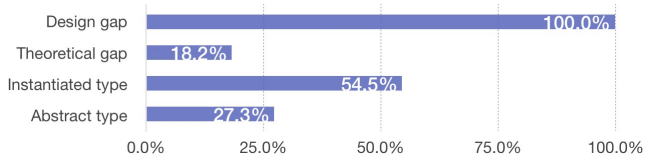

Figure 6. Challenge addressed

of the IT artifact, 2) intervention in the organization, and 3) evaluation (BIE).

The initial design of the artifact is based on the problem statement of stage 1. After the first BIE cycle, the artifact is refined and build upon the feedback from the previous BIE cycle. By deploying the artifact to the organization early, practitioners get to experience and test the design. They can influence it by giving feedback on how well the design performs compared to their expectations and assumptions.

During one BIE cycle, the problem and the artifact are continuously evaluated by deploying the artifact in the organization early and improved by building and refining the IT artifact based on the feedback. This feedback is evaluated and in case the organization adopts or rejects the artifact, a new BIE cycle starts.

The labels identified from the literature paper that are related to the second stage can be classified into two parts: 1) DSL guidelines and 2) threat modeling guidelines.

4.2.1 DSL Guidelines. To achieve a better quality of the language design and improve acceptance among language users, DSL guidelines guide language designers in the process of DSL development. Next, a selection of DSL guidelines is presented that apply to the development of MAL languages as well as DSL development in general.

G-2.1: Define language rules [34] Models often have to adhere to rules derived from the domain they are applied to, the language itself, or usage conventions. These rules may be strict by informing about missing elements or incorrect usage. They even can support conventions and default values. Language rules or well-formedness rules enable early error detection and prevent invalid or unwanted models.

G-2.2: The modeling language is specified by a language metamodel [24] Specifying the modeling language in a language metamodel includes defining the abstract syntax, concrete syntax, well-formedness rules, and semantics in a metamodeling language. This language metamodel facilitates easy understanding of the scope and elements of the language and provides a standardized way for further changes and adaptions. Consequently, missing metamodels impedes the development, use, extension, and reuse of DSLs.

G-2.3: DSL's support for error prevention and model checking [27] Error prevention and model checking are important for producing reliable programs. Because often the inspection of all relevant parts of a model for errors and completeness are either missing or incomplete, DSLs need improvement in this area. Consequently, when designing a modeling language, providing model checking and error prevention measurements plays an important role in helping modelers to build reliable solutions to their problems.

G-2.4: Comprehensibility and learnability [27] For DSLs to be comprehensible, language elements have to be understandable, e.g., by reading their description or doing a tutorial. This facilitates learning the DSL and to design programs with it. When DSLs are overly complex and do not have any documentation, they are hard to understand and thus, drive potential users away

G-2.5: Provide for language extensibility [53] Software languages are software too [6] and, hence, often subject to evolution beyond the conceptions leading to its first release(s). This especially holds where languages are relatively generic and will be specialized by future users, such as the UML with MechatronicUML [4] or UML/P [50]. Without language suitable extension mechanisms [21], users will abandon such a language and create their variants from scratch.

G-2.6: Adopt existing domain notations [13, 30, 34, 40, 63, 67] For the language to be suitable, its concrete syntax should reflect concepts known by the modeler. These concepts, generally, originate from related domains, that flatten the learning curve by providing an intuitive understanding of notations.

G-2.7: Design for language evolution [33, 37, 62, 63] As languages will evolve, they should be designed accordingly. This includes the modularization of concepts. Evolution in language design enables the modification of dedicated aspects of a language, instead of causing extensive changes.

G-2.8: Balance generality and specificity [30, 63, 67] Modeling languages should abstract from implementation details. However, at the same time, they have to offer expressive modeling techniques to be appropriate for their use case. Because these requirements often contradict each other, finding an appropriate balance between both is crucial.

G-2.9: Reuse and compose existing language definitions [30, 33, 40, 63] Modeling languages often are composed of and reuse existing concepts of other languages. To reduce the implementation and maintenance effort, it makes sense to reuse existing language definitions on the technical level as well [5]. Composing multiple fragments is a powerful technique to increase sustainability in developing new languages.

G-2.10: The language is assessable regarding its quality and correctness [24] To ensure a certain quality and correctness of the language, the DSL should be evaluated and refined constantly throughout the development process.

G-2.11: Perform analytical methods [60] To ensure that the modeling language fulfills certain functional requirements it is important to perform analytical methods, e.g., static analysis, architecture analysis, optimization, and dynamic analysis.

G-2.12: Perform experimental testing, and descriptive

methods [60] To measure and to ensure that the language in development fits functional and non-functional requirements, experimental and descriptive methods should be applied.

G-2.13: Compare to language design patterns [2] Research already published a multitude of language design guidelines, patterns, and best practices. During the development process of DSLs, they should be constantly compared to these guidelines to further improve the DSL in the next development iteration.

G-2.14: Users can recognize whether the DSL is appropriate for their needs [27] For the DSL to be successful it needs to fulfill the requirements stated by its users. To achieve this, the users should be enabled to recognize whether or not the DSL is appropriate for their needs.

G-2.15: Testing: Test the language design on language users [66] For the DSL to meet the language user's needs, it is important to constantly involve the intended users in the development process by letting them test the language design.

4.2.2 Application of DSL Guidelines to MAL DSLs.

This section summarizes our findings after we performed the survey. The survey that was performed with MAL DSL developers showed that some of the guidelines presented above were already included in the MAL development process. The survey indicates that most of the language developers defined language rules for their languages.

The presented DSL guidelines are taken into account at different stages of stage 2. When designing or building their DSL, they reused existing documentation or reused already existing language definitions to define the rules for their language. 75% of the survey participants explicitly stated that they designed their DSL for language evolution. For instance, they used abstract assets to be easily extendable in future versions. Regarding balancing specialty and generality, some participants answered that they wanted their DSL to be specialized enough to cover the security aspects of their application, but also wanted it to be general enough for it to be applied to other domains.

In the intervention, often, adopting the existing domain notation was a requirement for the design of the developed DSL, e.g., the DSL should enable to model the security aspects of AWS as close as possible. This was ensured by involving stakeholders of the language and users that can recognize whether the DSL is appropriate for their application.

For the evaluation, the language developers performed a static analysis of their languages and models by constructing attack paths as well as unit tests. By using their language to model-real world attack scenarios for cloud environments, e.g., AWS, and model known security issues, they performed experimental testing. Only a few of the participants evaluated their DSL by comparing it to existing language design patterns.

4.2.3 Threat Modeling Guidelines.

In addition to the DSL guidelines addressed above, the labels identified from the reviewed literature that correspond to threat modeling guidelines are presented as follows.

G-2.16: Adopt existing domain terms In the building stage, using easy-to-understand icons (symbols) associated with the elements of a threat modeling language supports the communication among participants with different backgrounds. Also, the threat modeling language should be easily understandable for practitioners, and be sufficiently precise to allow in-depth risk analysis [65].
In addition, a threat library can be used to guide the identifying and analysis of threats. A threat library can reduce the expertise required for threat modeling [11, 58].

G-2.17: Use an appropriate ontology An appropriate ontology is needed for modeling threats to the system by providing a formal and comprehensive knowledge base [51].

G-2.18: Interview domain experts Interviewing domain experts can be involved during threat modeling (by identifying threats and countermeasures), as well as validating the threat modeling approaches and results [51].

G-2.19: Stakeholder involvement through brainstorming or the Delphi method The most important assets of the system are identified in a brainstorming session with security experts [56]. This expert knowledge can be used as an information source for threat modeling. The Delphi method can be used to identify a possible weak spot because of its dependency on who is participating from the stakeholders [36].

4.2.4 Application of threat modeling guidelines to MAL DSLs.

According to all the answers to the questionnaire, all the four categories above are addressed. Regarding the requirement of building a DSL, 81.8% of the respondents require to reuse a threat library/existing artifacts/standards [11, 58], and 27.3% of them require to use an appropriate ontology [51]. To customize the DSL, 72.7% of the respondents require to use easy-to-understand icons (symbols) [65]. In terms of validating the DSL, 90.9% of the DSLs are validated through test cases, while only 9% of the DSLs are validated through industry/security experts or Turing tests.

However, some dimensions are missing from the answers, e.g., validating the modeling language by the Delphi method [8]. Specifically, threat modeling work can lack semantics making it difficult for both humans and systems to understand the architecture deception exactly and commonly, and ontology-based approaches can be applied to solve this issue [29]. Also, the Delphi method has not been addressed by the answers or at least fully used when evaluating the built DSLs, which is a forecasting process framework based on the results of multiple rounds of questionnaires sent to a panel of experts. While security experts, domain experts are found to be involved in the intervention and evaluation steps of

several DSLs, e.g., sclLang [49], powerLang [17], azureLang [19], and coreLang [31].

4.3 Stage 3 - Reflection and Learning

While in the two previous stages the focus on a problem and its solution for a single instance, in stage 3 the solution is conceptualized to address a broader class of problems [52]. Stage 3 parallels the previous stages and fosters a conscious reflection on the problem, the applied theories, and the emerging artifact. Moreover, the researchers should alter the research process based on evaluation results if necessary. To reflect on the developed artifact, the developers of MAL DLSs mentioned different approaches that were also highlighted in the literature.

G-3.1: Co-Creation For learning activities, the co-creation of artifacts [18] between MAL developers and stakeholders have reported. However, Haj-Bolouri et al. [18] additionally emphasize that for successful learning a tight coupling between researchers and stakeholders is necessary as well as a continuous exchange between these two groups.

The latter is also observable in the questionnaires, as some of the participants acknowledge a continuous evaluation of the artifact in close exchange with the stakeholders. Complementary, Haj-Bolouri et al. [18] name also prototypes, the direct implementation in an organization, and continuous documentation of the artifact as means for reflection.

G-3.2: Stakeholder involvement According to our participants, the exchange between the developers and the stakeholders takes place in workshop formats, in which the artifacts are presented and discussed. Additionally to these workshops, scientific literature [18] mentions training sessions to foster teach the stakeholders about the artifact. Such training sessions can be a useful supplement to the existing approaches to communicate MAL DSLs.

G-3.3: Stakeholder motivation There are two drivers for the stakeholders to participate in the aforementioned workshops. On the one hand, the stakeholders are interested in assessing the security of their systems. On the other hand, the interest is on automating the existing assessment. Hence, we can see a maturation of the stakeholders' interest in MAL related to their actual application of security measures.

4.4 Stage 4 - Formalization of Learning

Finally, in stage 4, the objective is to formalize the findings by providing a general solution to a class of problems [52]. Therefore, the researcher is supposed to reflect on the accomplishments realized in the artifact and characterize the organizational impact.

G-4.1: Outcome of formalization Research related to the formalization differentiates between two different artifact types in the realm of ADR [18, 38]: either the solution is focused on an information system or on changing the organization. As expected, we found in all answers of the questionnaire that the research was related to information systems and not to the organizations themselves.

Unfortunately, we could not generate deeper insights for the formalization with the questionnaires. Most likely, this is caused by the fact that MAL DSLs are usually designed to solve a certain problem and the efforts to generalize these languages to a broader corpus of problems are omitted. Exceptions can be found in the approach of Hacks and Katsikeas [16], who propose an ecosystem of MAL DSLs to foster reuse among the languages, and coreLang [31] providing abstract assets that fit for different domains. To enable future formalization, the development of MAL DSLs can benefit from the existing DSR.

G-4.2: Design principles One approach to formalize the outcomes of the research is to distil design principles. These design principles can relate to the artifact itself [7, 15, 59, 64] and its properties [7, 14, 15], the purpose and context of the artifact [7, 15, 59, 64], the design process of the artifact [7, 64], and the evaluation process of the artifact [7, 15].

G-4.3: Formalization approach The formalization can also be guided by different approaches such as problem structuring [15, 59], utility theory [59], hypothesis building [59], grounded theory [15, 38], heuristic theorizing [15], or engaged scholarship [15, 44].

G-4.4: Artifact contribution Another way of formalizing is to reflect on the contribution of the artifact. Depending on the abstraction level of the solution, we can differentiate between three classes of solutions [14, 59]: a well-developed design theory, a nascent design theory, and a situated implementation. For classical MAL DSLs (e.g., [17, 32]), the contribution is expected to be a situated implementation, while for some approaches (e.g., [16, 31]) one can also argue for a nascent design theory.

G-4.5: Artifact nature The resulting artifact can either be of descriptive or prescriptive nature [1, 14]. The MAL DSLs describe classically known vulnerabilities related to certain assets and their relations. Hence, the contribution is descriptive. However, those descriptive languages can be used to describe possible future configurations and, thus, the contribution can be prescriptive.

G-4.6: Contribution maturity The contribution can be classified to its maturity [14]. If a known solution is applied to a known problem (Routine Design), there is no significant contribution to the work. If an existing solution is extended to a new problem (Exaptation), there is a research opportunity and a knowledge contribution. The same contribution holds for the cases if there is a new solution for a known problem (Improvement) or if there is a new solution for a new problem (Invention). As MAL DSLs are relying on an existing solution (i.e., MAL) and are developing for a new domain that is not covered yet, the contribution is expected to be an exaptation. If a language is redesigned, an improvement is also possible.

Table 1. Addressed categories in documented MAL DSLs

	vehicleLang	coreLang	powerLang
Stage 1	G-1.1 G-1.3	G-1.3	G-1.2 G-1.3
Stage 2	G-2.15	G-2.6 G-2.15	G-2.6 G-2.8 G-2.9 G-2.15 G-2.16
Stage 3	-	G-3.2 G-3.3	-
Stage 4	-	G-4.1	G-4.2 G-4.3

5 Demonstration

To demonstrate how our findings relate to already developed MAL DSL, we will discuss the approach followed for the development of already presented MAL DSLs. Table 1 summarizes shortly the found labels in the different languages.

5.1 vehicleLang

We start with vehicleLang [32], which is a DSL for the automotive domain, and the problem that tries to solve is how to perform cyber-attack simulation on vehicular infrastructures. In the process followed for the development of this language and by following the four stages of the ADR we can identify the following labels.

Stage 1. Regarding the first stage, the main method for information gathering and defining the problem was an SLR of the domain (G-1.1) and some limited input from experts. Then the problem that this language tries to solve can be categorized as an instantiated problem (G-1.3).

Stage 2. Moving on to the second stage, which involves the BIE cycle. The building of the language was heavily based on existing literature and therefore clear language rules were set. Additionally, since MAL was used as the development framework, a metamodel frames the structure of the language. When it comes to stakeholder intervention/involvement, this was minimal in this work since only a few interviews with one domain expert were conducted during the development phase. Then, regarding validation, again one interview with a domain expert was used together with unit tests (G-2.15).

Stage 3. Coming to the third stage, a reflection on the created artifact was done both by writing a scientific paper but also on presenting it in both a conference but also a workshop. Finally, as it mostly happens with all MAL DSLs, a formalization and generalization of the created artifact was not done on this work since an attempt to solve a very specific problem was the main goal of it. Discussing what could have been made better on this work, we first require higher involvement of the stakeholders, which are the automotive domain experts, because this would also allow a higher level of validation of the artifact in stage 2.

Stage 4. Finally, we can argue that processes of the final stage of the ADR process are completely missing from this work. To correct this, the solution would be to try to perform some of them, for example, build the language with future extensions in mind. One more thing that could be done is to also try to elaborate on how the designed artifact could be used towards solving a broader group of problems.

5.2 coreLang

coreLang [31] is another MAL DSL that was designed as a boilerplate language for other MAL DSLs because it includes all the basic and common IT components that are found on IT systems of different domains.

Stage 1. Although how the problem was identified is not clearly stated, it seems that the main developers of that language were aware of the problem of double work needed in other MAL DSLs to cover similar elements of different infrastructures. For that reason, they aimed at creating a fundamental language. The problem can be characterized as an instantiated problem that addresses an implementation gap (G-1.3).

Stage 2. For coreLang to be abstract enough but widely acceptable also, it adopts a common terminology (G-2.6) found on all IT infrastructures. When it comes to stakeholder intervention/involvement, this is an example of high involvement since weekly meetings with domain experts were conducted during the whole development phase for both brainstormings but also for providing feedback back to the developers and suggestions on possible improvements in the language. Finally, regarding the evaluation of the language, both test cases and unit tests (G-2.15) were used.

Stage 3. Because of the aforementioned high involvement of the stakeholders, we can state that this is one of the few languages that covers most of the processes found on this stage (G-3.2 and G-3.3). Additionally, due to the abstraction of the language, a conceptualization of how a broader class of problems can be solved comes as a natural consequence. This was supported, on that work, by trying to evaluate coreLang against the MITRE ATT&CK matrix of possible attacks.

Stage 4. Adhering to the authors' goal, coreLang was built with future extensions in mind, and even more, on top of that, a whole MAL DSL ecosystem was also proposed based on coreLang [16] (G-4.1).

5.3 powerLang

powerLang [17] is a MAL DSL that was designed to enable organizations in the power domain to assess the security

of their IT and operational technology (OT) environments. Therefore, it reuses two existing languages (coreLang [31] and sclLang [49]) to provide assets for office and for substation environments. To bridge the gap between these two worlds, icsLang is proposed that is thought to represent the environment controlling the substation.

Stage 1. It is not explicitly elaborated how the problem founding the need for powerLang was identified. From the background of this language, we can deduce that the language is developed in a research project, and stakeholders (researchers and practitioners) might have been involved in the problem definition during the proposal writing (G-1.2). The problem itself can be categorized as instantiated addressing a design gap (G-1.3).

Stage 2. One main characteristic of powerLang is that it reuses and composes existing language definitions (G-2.9) by using coreLang and sclLang and linking them to each other. To ease the use of powerLang by practitioners, it further adopts their common terminology (G-2.6, G-2.16). However, to find a balance between generality and specificity (G-2.8), icsLang –as a subset of powerLang– uses the terminology of industrial control systems, but is not further tailored to the power domain. Thus, icsLang should allow being used also in non-power domains.

The development process of powerLang is not further detailed. It is solely stated that icsLang is build using MITRE ATT&CK for Industrial Control Systems[1]. Thus, the direct involvement of stakeholders is unlikely. Concerning evaluating the language (G-2.15) unit tests have been developed to ensure the correct functionality of the language and it is demonstrated on a real-world attack.

Stage 3. As there have been no activities conducted to foster the exchange between the language developers and possible stakeholders, there have been also no joint learning activities. Consequently, stage 3 has not been addressed in the development of powerLang.

Stage 4. Similarly, it is not stated that any efforts have been taken to formalize the results beyond solving the given problem. However, there are two contributions to formalization. Firstly, icsLang is designed to cover also other domains than power (G-4.3). Secondly, in the related articles design principles are suggested that are thought to ease the linking of different MAL DSLs (G-4.2).

Reflecting on the development process of powerLang, we can presume that there is an opportunity for improvement. Especially, it is recommended to involve stakeholders to a greater extend. This includes the problem definition, but also the development of the language and the paralleling learning activities. Further, the article presenting powerLang would have benefited from elaborating more extensively on the scientific contribution and trying to formalize the findings more.

[1]https://collaborate.mitre.org/attackics/index.php/Main_Page

6 Discussion

Hitherto, we have developed a method to guide the instantiation of MAL and demonstrated its application based on three published languages. However, several points offer the possibility for improvement.

Firstly, our survey had a quite small number of participants, which was mainly caused by a small population of available persons, who developed a MAL DSL so far. Moreover, we achieved a response rate of 63 %, which is above average for online surveys [10]. Apart from the number of participants, the participants were also from only two different organizations, which are additionally closely linked to each other. On the one hand, there are no other organizations in which MAL DSLs have been developed yet. On the other hand, we could have taught other language developers MAL to gather their experiences. However, we did not opt for this option, because a developer would have developed an entire language from scratch. This would have been too time-intensive for our needs.

Secondly, there are different options related to the design of the surveys. We opted for open-ended questions as we wanted to gather a wide spectrum of answers regarding the design of MAL DSLs and not to steer the answers of the participants in the direction of our thinking. However, the formalization of these answers becomes more challenging and subjective. Alternatively, we could use closed questions, which would ease the formalization and lead to more objective results. Nonetheless, we prioritized the opportunity to gather unexpected results over more objective results, which we will address in our future work.

Another means to ease the formalization could be to apply machine learning techniques. This would be challenging as we were using open questions, but not impossible. Nevertheless, we emphasized human intuition in our formalization of the survey results, which could not be mirrored by machine learning. Yet, we are considering machine learning to analyze the outcomes of surveys with closed questions.

As indicated before, we manually performed the coding of the survey answers. To reach a consensus, we presented our decisions to each other and if necessary, we discussed our results. This is prone to human effects, such as single persons can dominate the discussion and influence the outcome. We tried to create an open space to value all opinions the same, but we cannot guarantee that subliminal tendencies might remain. Therefore, we plan to experiment with more subjective approaches, which rely on numerical measures [45].

Thirdly, applying Venable et al. [61] results in the recommendation to rely on our method on ADR. This was also in line with our expectations as ADR emphasizes the exchange

with stakeholders during the artifact development. However, it might be that other DSR approaches or a combination of them suit the development of MAL DSLs better. To determine this, several applications of our method are necessary that show what needs to be improved.

Finally, the generalization of our approach to other DSLs should be researched. We assume that most parts of our method can be applied to DSLs, too. Just the guidelines that are related to threat modeling (cf. Section 4.2.3) might be removed or replaced by guidelines that are tailored to the respective domain.

7 Related Work

In this work, we provide guidelines for developing MAL DSLs and since MAL is both a domain-specific and a threat modeling language framework, we consider both as related work. First, there are guidelines for framing the design of DSLs. With the same intention as our work to help guide the development of a DSL, Mernik et. al. describe patterns in the different phases of the development [40]: decision, analysis, design, implementation, and deployment. Two of their patterns have been adapted in our work and are the guidelines "Adopt existing domain notations" and "Reuse and compose existing language definition". Mernik et. al. do not, however, define patterns for the implementation phase as they see it as out of scope for their paper. The method of assigning patterns is similar to our method of assigning labels per phase according to the survey responses. Gabor et. al. do not define patterns but instead guidelines according to the categories purpose, implementation, contents of the language as well as concrete and abstract syntax [30]. Three of their guidelines have been adapted in our work, namely "Adopt existing domain notations", "Balance generality and specificity" and "Reuse and compose existing language definitions". Compared to our guidelines, they do not provide any guidelines regarding reflection and learning but focus on the development itself. Second, in a systematic literature review of threat modeling by Xiong and Lagerström [68] several articles are identified that either describe threat modeling or a specific process of developing a threat model. Especially, the latter articles can be compared to our approach. One paper divides the process into first determining scope, then gathering background information, describing the component, and recording any weaknesses [57]. Finally, the author outlines how to gather threats for the model by brainstorming. The process is based on the author's previous experience in developing threat models. This is similar to how we base our approach on previous MAL developer's experience but we fit this into the DSR framework.

Another paper describes a process of four steps [11]. These four steps are to first create a Dataflow Diagram (DFD), gathering attacks with the help of a threat library, assessing the risks, and mitigating the risks. The third article describes the process as iterative and consisting of six stages [28]. The stages are identification of assets, architectural overview, fragmentation of the system, identification of threats, documentation of these threats, and lastly rating them. Overall, these guidelines, are similar to those in stage 2 of the ADR but do not focus on insights into the other three stages.

In addition, there have been developments of artifacts, other than languages, where the ADR process has been used. ADR has been used for developing concept generation and selection methods where the authors determined that using ADR was feasible for their project even though they saw some weaknesses [47]. ADR has also been used for creating an Inter- Organizational Social Networking Information System (IO SNIS) [42] and Organisational Culture Assessment Instrument-Spilter (OCAI-Spilter), which is a culture measurement tool [35].

8 Conclusion

We have investigated whether ADR, a form of DSR tailored to creating IT artifacts that are shaped by their organizational context during development and use, can be used to create DSLs (i.e., instances) of the Meta Attack Language. To this end, we have surveyed experienced developers of such languages. From their answers and literature on DSL development, we extracted guidelines for the development of MAL DSLs using ADR. Through this, we identified many guidelines for the BIE stage of ADR but only a few for the problem formulation, reflection, and formalization of learning stages. At the same time, we recognized that these stages were also neglected in documented MAL DSLs [17, 31, 32]. This suggests that ADR might be a suitable framework for the systematic engineering of (MAL) DSLs. However, it also hints that there are gaps in systematically embedding DSLs in organizational contexts that demand further research, especially concerning the non-BIE stages. Moreover, we aim to research if our approach applies to other DSLs.

Acknowledgments

This project has received funding from the European Union's H2020 research and innovation program under the Grant Agreement No. 832907, the Swedish Centre for Smart Grids and Energy Storage (SweGRIDS), and the Deutsche Forschungsgemeinschaft (DFG) under Grant Agreement No. 441207927.

References

[1] Hazbi Avdiji and Robert Winter. 2019. Knowledge Gaps in Design Science Research. In *International Conference on Information Systems (ICIS) 2019*. https://www.alexandria.unisg.ch/258227/

[2] Ankica Bariic, Vasco Amaral, and Miguel Goulão. 2012. Usability evaluation of domain-specific languages. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE, 342–347.

[3] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. 2011. Quality in Use of Domain-Specific Languages: A Case Study. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and*

Usability of Programming Languages and Tools (Portland, Oregon, USA) *(PLATEAU '11)*. Association for Computing Machinery, New York, NY, USA, 65–72. https://doi.org/10.1145/2089155.2089170

[4] Sven Burmester, Holger Giese, and Matthias Tichy. 2005. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In *Model Driven Architecture*, Uwe Aßmann, Mehmet Aksit, and Arend Rensink (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 47–61.

[5] Arvid Butting, Robert Eikermann, Katrin Hölldobler, Nico Jansen, Bernhard Rumpe, and Andreas Wortmann. 2020. A Library of Literals, Expressions, Types, and Statements for Compositional Language Design. *Special Issue dedicated to Martin Gogolla on his 65th Birthday, Journal of Object Technology* 19, 3 (October 2020), 3:1–16. Special Issue dedicated to Martin Gogolla on his 65th Birthday.

[6] Benoit Combemale, Robert France, Jean-Marc Jézéquel, Bernhard Rumpe, James Steel, and Didier Vojtisek. 2016. *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series.

[7] Stefan Cronholm and Hannes Göbel. 2018. Guidelines supporting the formulation of design principles. In *29th Australasian Conference on Information Systems (ACIS), Sydney, December 3-5, 2018*.

[8] Norman Dalkey and Olaf Helmer. 1963. An Experimental Application of the DELPHI Method to the Use of Experts. *Management Science* 9 (1963), 351–515. Issue 3. https://doi.org/10.1287/mnsc.9.3.458

[9] Defense Use Case. 2016. Analysis of the cyber attack on the ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC). Available: https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf.

[10] Elisabeth Deutskens, Ko De Ruyter, Martin Wetzels, and Paul Oosterveld. 2004. Response rate and response quality of internet-based surveys: an experimental study. *Marketing letters* 15, 1 (2004), 21–36.

[11] Danny Dhillon. 2011. Developer-Driven Threat Modeling: Lessons Learned in the Trenches. *IEEE Security & Privacy* 9, 4 (2011), 41–47. https://doi.org/10.1109/MSP.2011.47

[12] Mathias Ekstedt, Pontus Johnson, Robert Lagerström, Dan Gorton, Joakim Nydrén, and Khurram Shahzad. 2015. Securi CAD by Foreseeti: A cad tool for enterprise cyber security management. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 152–155.

[13] Ulrich Frank. 2013. Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In *Domain engineering*. Springer, 133–157.

[14] Shirley Gregor and Alan R Hevner. 2013. Positioning and presenting design science research for maximum impact. *MIS quarterly* (2013), 337–355.

[15] Robert Wayne Gregory and Jan Muntermann. 2014. Research Note —Heuristic Theorizing: Proactively Generating Design Theories. *Information Systems Research* 25, 3 (September 2014), 639–653. https://doi.org/10.1287/isre.2014.0533

[16] Simon Hacks and Sotirios Katsikeas. 2021. Towards an Ecosystem of Domain Specific Languages for Threat Modeling. In *Proc. of the 33rd International Conference on Advanced Information Systems Engineering (to be published)*. 1–15.

[17] Simon Hacks, Sotirios Katsikeas, Engla Ling, Robert Lagerström, and Mathias Ekstedt. 2020. powerLang: a probabilistic attack simulation language for the power domain. *Energy Informatics* 3, 1 (2020).

[18] Amir Haj-Bolouri, Lennarth Bernhardsson, and Matti Rossi. 2016. PADRE: A Method for Participatory Action Design Research. In *Tackling Society's Grand Challenges with Design Science*, Jeffrey Parsons, Tuure Tuunanen, John Venable, Brian Donnellan, Markus Helfert, and Jim Kenneally (Eds.). Springer International Publishing, Cham, 19–36.

[19] Ahmad Hawasli. 2018. *azureLang: a probabilistic modeling and simulation language for cyber attacks in Microsoft Azure cloud infrastructure*. Master's Thesis. KTH Royal Institute of Technology, Stockholm, Sweden.

[20] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. Design science in information systems research. *MIS quarterly* 28, 1 (2004), 75–105.

[21] Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. 2018. Software Language Engineering in the Large: Towards Composing and Deriving Languages. *Computer Languages, Systems & Structures* 54 (2018), 386–405.

[22] Hannes Holm, Khurram Shahzad, Markus Buschle, and Mathias Ekstedt. 2015. P^2CySeMoL: Predictive, Probabilistic Cyber Security Modeling Language. *IEEE Transactions On Dependable And Secure Computing* 12, 6 (2015), 626–639.

[23] Industrial Defender. 2021. Florida Water Treatment Plant Hit With Cyber Attack. Available: https://www.industrialdefender.com/florida-water-treatment-plant-cyber-attack/.

[24] Sven Jannaber, Dennis M Riehle, Patrick Delfmann, Oliver Thomas, and Jörg Becker. 2017. Designing a framework for the development of domain-specific process modelling languages. In *International Conference on Design Science Research in Information System and Technology*. Springer, 39–54.

[25] Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. 2018. A Meta Language for Threat Modeling and Attack Simulations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM, 38.

[26] Coquessa Jones and John R. Venable. 2020. Integrating CCM4DSR into ADR to Improve Problem Formulation. In *Designing for Digital Transformation. Co-Creating Services with Citizens and Industry*, Sara Hofmann, Oliver Müller, and Matti Rossi (Eds.). Springer International Publishing, Cham, 247–258.

[27] Gökhan Kahraman and Semih Bilgen. 2015. A framework for qualitative assessment of domain-specific languages. *Software & Systems Modeling* 14, 4 (2015), 1505–1526.

[28] R. Kamatchi and Kimaya Ambekar. 2016. Analyzing Impacts of Cloud Computing Threats in Attack based Classification Models. *Indian Journal of Science and Technology* 9 (06 2016). https://doi.org/10.17485/ijst/2016/v9i21/95282

[29] Dongwoo Kang, Jeongsoo Lee, Sungchul Choi, and Kwangsoo Kim. 2010. An ontology-based Enterprise Architecture. *Expert systems with applications* 37, 2 (2010), 1456–1464.

[30] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2009. Design Guidelines for Domain Specific Languages. In *Domain-Specific Modeling Workshop (DSM'09) (Techreport B-108)*. Helsinki School of Economics, 7–13.

[31] Sotirios Katsikeas, Simon Hacks, Pontus Johnson, Mathias Ekstedt, Robert Lagerström, Joar Jacobsson, Max Wällstedt, and Per Eliasson. 2020. An Attack Simulation Language for the IT Domain. In *Graphical Models for Security*, Harley Eades III and Olga Gadyatskaya (Eds.). Springer International Publishing, Cham, 67–86.

[32] Sotirios Katsikeas, Pontus Johnson, Simon Hacks, and Robert Lagerström. 2019. Probabilistic Modeling and Simulation of Vehicular Cyber Attacks: An Application of the Meta Attack Language. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP)*.

[33] Steven Kelly and Risto Pohjonen. 2009. Worst Practices for Domain-Specific Modeling. *IEEE software* 26, 4 (2009), 22–29.

[34] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons.

[35] J. Lee, J. V. Hillegersberg, and K. Kumar. 2015. An Action Design Research on development and deployment of a computer-based group discussion support tool for achieving consensus and culture change at an educational institution. In *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. 1367–1371. https://doi.org/10.1109/IEEM.2015.7385871

[36] Engla Ling, Robert Lagerström, and Mathias Ekstedt. 2020. A Systematic Literature Review of Information Sources for Threat Modeling in the Power Systems Domain. In *Critical Information Infrastructures Security*, Awais Rashid and Peter Popov (Eds.). Springer International Publishing, Cham, 47–58.

[37] Steve Livengood. 2012. Experiences in Domain-Specific Modeling for Interface Specification and Development. In *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*. 1–2.

[38] Giovanni Maccani, Brian Donnellan, and Markus Helfert. 2014. Systematic problem formulation in action design research: The case of smart cities. *ECIS 2014 Proceedings - 22nd European Conference on Information Systems* (01 2014).

[39] Alyona Medelyan. [n.d.]. *Coding Qualitative Data: How to Code Qualitative Research (Updated 2020)*. https://getthematic.com/insights/coding-qualitative-data/

[40] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and How to Develop Domain-Specific Languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[41] Savita Mohurle and Manisha Patil. 2017. A brief study of wannacry threat: Ransomware attack 2017. *International Journal of Advanced Research in Computer Science* 8, 5 (2017), 1938–1940.

[42] Matthew T. Mullarkey and Alan R. Hevner. 2019. An elaborated action design research process model. *European Journal of Information Systems* 28, 1 (2019), 6–20. https://doi.org/10.1080/0960085X.2018.1451811 arXiv:https://doi.org/10.1080/0960085X.2018.1451811

[43] Robert C. Nickerson, Upkar Varshney, and Jan Muntermann. 2013. A method for taxonomy development and its application in information systems. *European Journal of Information Systems* 22, 3 (2013), 336–359.

[44] Peter Nielsen and John Persson. 2016. Engaged Problem Formulation in IS Research. *Communications of the Association for Information Systems* 38 (05 2016), 720–737. https://doi.org/10.17705/1CAIS.03835

[45] Cliodhna O'Connor and Helene Joffe. 2020. Intercoder Reliability in Qualitative Research: Debates and Practical Guidelines. *International Journal of Qualitative Methods* 19 (2020), 1609406919899220. https://doi.org/10.1177/1609406919899220 arXiv:https://doi.org/10.1177/1609406919899220

[46] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. A design science research methodology for information systems research. *Journal of management information systems* 24, 3 (2007), 45–77.

[47] Anna Malou Petersson and Jan Lundberg. 2016. Applying Action Design Research (ADR) to Develop Concept Generation and Selection Methods. *Procedia CIRP* 50 (2016), 222–227. https://doi.org/10.1016/j.procir.2016.05.024 26th CIRP Design Conference.

[48] Roel Popping. 2015. Analyzing Open-ended Questions by Means of Text Analysis Procedures. *Bulletin of Sociological Methodology/Bulletin de Méthodologie Sociologique* 128, 1 (2015), 23–39. https://doi.org/10.1177/0759106315597389 arXiv:https://doi.org/10.1177/0759106315597389

[49] Engla Rencelj Ling and Mathias Ekstedt. 2021. Generating Threat Models and Attack Graphs Based on the IEC 61850 System Configuration Description Language. In *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems* (Virtual Event, USA) *(SAT-CPS '21)*. Association for Computing Machinery, New York, NY, USA, 98–103. https://doi.org/10.1145/3445969.3450421

[50] Bernhard Rumpe. 2016. *Modeling with UML: Language, Concepts, Methods*. Springer International.

[51] Bilal Al Sabbagh and Stewart Kowalski. 2015. A Socio-technical Framework for Threat Modeling a Software Supply Chain. *IEEE Security & Privacy* 13, 4 (2015), 30–39. https://doi.org/10.1109/MSP.2015.72

[52] Maung K Sein, Ola Henfridsson, Sandeep Purao, Matti Rossi, and Rikard Lindgren. 2011. Action design research. *MIS quarterly* (2011), 37–56.

[53] Bran Selic. 2009. The theory and practice of modeling language design for model-based software engineering—a personal perspective. In *International Summer School on Generative and Transformational Techniques in Software Engineering*. Springer, 290–321.

[54] J. P. Shim, Ramesh Sharda, Aaron M. French, Rhonda A. Syler, and Karen P. Patten. 2020. The Internet of Things: Multi-faceted Research Perspectives. *Communications of the Association for Information Systems* (2020), 511–536.

[55] Adam Shostack. 2014. *Threat modeling : designing for security* (1st edition. ed.). Wiley.

[56] Inger Anne Tøndel, Martin Gilje Jaatun, and Maria Bartnes Line. 2013. Threat Modeling of AMI. In *Critical Information Infrastructures Security*, Bernhard M. Hämmerli, Nils Kalstad Svendsen, and Javier Lopez (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 264–275.

[57] P. Torr. 2005. Demystifying the threat modeling process. *IEEE Security Privacy* 3, 5 (2005), 66–70. https://doi.org/10.1109/MSP.2005.119

[58] Anton V. Uzunov and Eduardo B. Fernandez. 2014. An extensible pattern-based library and taxonomy of security threats for distributed systems. *Computer Standards & Interfaces* 36, 4 (2014), 734–747. https://doi.org/10.1016/j.csi.2013.12.008 Security in Information Systems: Advances and new Challenges.

[59] John Venable. 2006. The role of theory and theorising in design science research. In *Proceedings of the 1st International Conference on Design Science in Information Systems and Technology (DESRIST 2006)*. Citeseer, 1–18.

[60] John Venable, Jan Pries-Heje, and Richard Baskerville. 2012. A comprehensive framework for evaluation in design science research. In *International Conference on Design Science Research in Information Systems*. Springer, 423–438.

[61] John R. Venable, Jan Pries-Heje, and Richard Baskerville. 2017. Choosing a Design Science Research Methodology. In *ACIS2017 Conference Proceeding*. University of Tasmania.

[62] Michael Vierhauser, Rick Rabiser, Paul Grünbacher, and Alexander Egyed. 2015. Developing a DSL-Based Approach for Event-Based Monitoring of Systems of Systems: Experiences and Lessons Learned (E). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 715–725.

[63] Markus Völter. 2009. Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology* 8, 6 (2009), 79–102.

[64] Jan vom Brocke and Alexander Maedche. 2019. The DSR grid: Six core dimensions for effectively planning and communicating design science research projects. *Electronic Markets* 29, 3 (2019), 379–385.

[65] Fredrik Vraalsen, Mass Soldal Lund, Tobias Mahler, Xavier Parent, and Ketil Stølen. 2005. Specifying Legal Risk Scenarios Using the CORAS Threat Modelling Language. In *Trust Management*, Peter Herrmann, Valérie Issarny, and Simon Shiu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 45–60.

[66] Robert Walter and Maic Masuch. 2011. How to integrate domain-specific languages into the game development process. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. 1–8.

[67] D Wile. 2004. Lessons Learned from Real DSL Experiments. *Science of Computer Programming* 51, 3 (2004), 265–290.

[68] Wenjun Xiong and Robert Lagerström. 2019. Threat modeling - a systematic literature review. *Computers & Security* 84 (2019), 53–69. https://doi.org/10.1016/j.cose.2019.03.010

[69] Wenjun Xiong, Emeline Legrand, Oscar Åberg, and Robert Lagerström. 2021. Cyber security threat modeling based on the MITRE Enterprise ATT&CK Matrix. *Software & Systems Modeling* (2021). https://doi.org/10.1007/s10270-021-00898-7

[70] Koen Yskout, Thomas Heyman, Dimitri Van Landuyt, Laurens Sion, Kim Wuyts, and Wouter Joosen. 2020. Threat Modeling: From Infancy

to Maturity. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results* (Seoul, South Korea) *(ICSE-NIER '20)*. Association for Computing Machinery, New York, NY, USA, 9–12. https://doi.org/10.1145/3377816.3381741

A Labels

Stage	Category	Label
1. Problem Formulation	Problem definition	SLR Problem of an organization Systematic empirical investigation expert interviews focus groups cause effect diagram
	Stakeholder involvement (who)	Researcher end-users practitioners
	Stakeholder involvment (how)	expert interviews focus groups survey status seminars
	Problem type	abstract problem instantiated problem
	Research gap	Theoretical gap Design gap
2. Building, Intervention, and Evaluation	Improve usability	Use easy-to-understand icons (symbols)
	Define language rules	Reuse an existing language Use an appropriate ontology Use annotations Use graphical modeling language Ad-hoc
	Reuse existing model	Threat library Petri Nets
	Build a knowledge base	Whiteboarding Interview domain experts
	User Error Protection	Define language rules DSL's support for error prevention and model checking Reliability of a DSL is defined as the property of a language that aids producing reliable programs (Guideline includes model checking ability/preventing unexpected relations) Model checking
	Modularity	DSL is composed of discrete components such that a change to one component has minimal impact on other components its elements Modularize and layer the language: particularly if the language is going to support multiple viewpoints for different sub-domains
	Adaptability	Provide for language extensibility Allow for incorporating "foreign" language fragments in models Extensibility: The degree to which a language has general mechanisms for users to add features Support variability on language level
	Reusability	Reuse language definitions Reuse type systems Compose existing languages: Reuse existing languages for the creation of a new one

Stage	Category	Label
2. Building, Intervention, and Evaluation		The language development is based on existing concepts
	Functional Correctness	Always start design with a semantics model The language provides (formal) language semantics
	Operability	Support model reuse on the language level
	Interoperability	Provide Integrability Integrate in an existing IDE support for development of DSLs with high Quality in Use
	Usability evaluation	Evaluate a designed artifact by performing usability evaluation
	Problem definition	Verify model against intended functions
	Validate result	security experts delphi method
	Validate method	security experts delphi method
	Accessibility	DSL conciseness, which refers to what terms can be deleted without compromising the domain artifact representativeness Avoid redundancy Clear language to target mapping: There should be a clear mapping of the language concepts to the concepts of relevant target representations. In an ideal case, all information required by the target representations can be extracted from the model Consistent style everywhere Simplicity: An easy-to-learn language is easy to use Homogeneity: Show related concepts in the same way, and keep them together; show unrelated concepts differently and separately Comprehensibility: DSL language elements are understandable
	Testability	The language is assessable regarding its quality and correctness Perform "Care for usability" evaluations Define the quality criteria to evaluate DSLs Observational methods include case study and field study Perform analytical methods include static analysis, architecture analysis, optimization, and dynamic analysis Perform Experimental, testing, and descriptive methods evaluate "how well the artifact supports a solution to the problem" use computer and lab simulations, field experiments, and lab experiments Quality in Use of a DSL should be assessed experimentally

Stage	Category	Label
2. Building, Intervention, and Evaluation		Use Evaluation method: Cognitive Dimensions (CD) that contains 14 dimensions: viscosity, visibility, compromise, hidden dependencies, expressiveness role, error tendency, abstraction, secondary browsing, mapping proximity, consistency, diffusion, hard mental operations, provisional, and progressive evaluation learnability was measured through the number of errors a subject committed, divided by effort; efficiency was measured by the size of the test set divided by effort satisfaction was measured in four levels: frustrating, unpleasant, pleasant, and pleasurable define quality concerns and associated metrics Compare DSL to already existing standard DSLs compare to language design patterns Maintainability: The degree to which a language is easy to maintain. DSLs can be altered and new concepts and concept extensions can be added. (modularity is considered here) Testing: Test the language design on language users Stakeholder groups are considered during development Define a methodological approach to support the evolution of a DSL's design based on user experience and infer its impact on quality improvement during its lifecycle (e.g. traceability of design decisions)
	Stakeholder involvement (who)	Domain experts Security experts Language developer
	Stakeholder involvement (how)	Brainstorming
	Appropriateness recognizability	Derive concepts from physical structure The language considers modelling pragmatics DSL expressiveness, which refers to in what extend the DSL represents the domain Adopt existing domain notations Derive concepts from Look&Feel Derive concepts from expected output Derive notation from corporate identity Appropriateness: DSL is appropriate for the specific applications of the domain (e.g., to express an algorithm) Users can recognize whether the DSL is appropriate for their needs The concepts of a modeling language should correspond to concepts prospective users are familiar with. There should be a clear mapping of the language concepts to the concepts of relevant target representations.
		Provide sufficient level of detail Balance compactness and understanding

Stage	Category	Label
2. Building, Intervention, and Evaluation	Functional Appropriateness	Multiple levels of abstractions: A modeling language should provide concepts that allow for clearly distinguishing different levels of abstraction within a model. Rationale: Conceptual models may represent different levels of abstraction, e.g., types and – in rare cases – instances. Overloading a model with different levels of abstraction compromises an appropriate interpretation of a model. Beware of overgeneralization The DSPML has a defined scope and purpose The concepts of a language should allow for modeling at a level of detail that is sufficient for all foreseeable applications. A modeling language should provide concepts that allow for clearly distinguishing different levels of abstraction within a model. Completeness: All concepts and scenarios of the domain can be expressed in the DSL Flexibility: Give the user flexibility in how to formulate a domain-specific program Usability of a DSL is the degree to which a DSL can be used by specified users to achieve specified goals
	N/A	Strive for 80% solution: Provide a solution that covers 80% of activities - more time left to deal with the real problems The language is based on requirement analysis
	Learnability	Learnability: The concepts and symbols of the language are learnable and rememberable Ask graphic designers for help Use descriptive notations Use a mixture of language notations
	Usability	Detect recurring patterns in design
3. Reflect and Learn	Learning activities	co-creating knowledge at every stage tight coupling between researchers and stakeholders
	Planning	workshop training session
	Evaluate	prototype implementation in the organization (interaction with stakeholders) continuous evaluation continuous documentation
	Reflection	present results discuss results collaboration
4. Formalization of Learning	Artifact type	IS-related artifact Organizational solution
	Design principles	artifact purpose building process context artifact properties evaluation process

Stage	Category	Label
4. Formalization of Learning	Approach	problem structuring utility theories hypotheses grounded theory heuristic theorizing applied science research action design research engaged scholarship
	Contribution	well-developed design theory nascent design theory situated implementation descriptive knowledge prescriptive knowledge
	Maturity	Invention Improvement Exaption Routine design