Towards an Ecosystem of Domain Specific Languages for Threat Modeling

Simon Hacks[0000−0003−0478−9347] and Sotirios Katsikeas[0000−0001−8287−3160]

Division of Network and Systems Engineering
KTH Royal Institute of Technology
Stockholm, Sweden
{shacks|sotkat}@kth.se

Abstract. Today, many of our activities depend on the normal operation of the IT infrastructures that supports them. However, cyber-attacks on these infrastructures can lead to disastrous consequences. Therefore, efforts towards assessing the cyber-security are being done, such as attack graph simulations based on system architecture models. The Meta Attack Language (MAL) was previously proposed as a framework for developing Domain Specific Languages (DSLs) that can be used for the aforementioned purpose. Since many common components exist among different domains, a way to prevent repeating work had to be defined. To facilitate this goal, we adapt taxonomy building by Nickerson and propose an ecosystem of MAL-based DSLs that describes a systematic approach for not only developing, but also maintaining them over time. This can foster the usage of MAL for modeling new domains.

Keywords: Ecosystem · Domain Specific Language · Cyber-security modeling · Cyber-security simulations.

1 Introduction

Today, our society is heavily dependent on IT infrastructures and cyber-attacks on them can have disastrous consequences for individuals, regions, and whole nations [28, 29, 35]. Therefore, it is necessary to keep such critical IT infrastructures secure. One approach is the assessment of their cyber-security, which can foster a higher degree of security and resilience. However, such an assessment is difficult as the security-relevant parts of the system must be understood, and all potential attacks must be identified [24]. We can determine three core challenges related to these needs: identification of all relevant security properties of a system; collection of further information on these properties; processing of the information needs to uncover all weaknesses that can be exploited.

Hitherto, we used attack graph simulations based on system architecture models [9] to support these tasks. Our attack simulation tool enables the security assessor to focus on the collection of the information about the system, as the simulation addresses the first and the third challenges. As the previous approach relies on a static implementation, we developed MAL (the Meta Attack

Language) [16]. MAL is a framework for domain-specific languages (DSLs) and used to define which information about a system is required. Moreover, it specifies the generic attack logic. Then, MAL automatically generates attack graphs involving the modeled system. Since MAL is a meta language (i.e., the set of rules that should be used to create a new DSL), no particular domain of interest is represented.

Over the last three years, after MAL was originally proposed on 2018 [16], a number of MAL-based DSLs started being developed. Over the past three years, we can notice that in the first two years the rate of new languages starting to be developed was steady (four new languages per year) but in the last year, that rate was significantly increased (nine new languages in 2020). This increasing trend can be explained by the fact that MAL has gained more recognition through conference paper presentations as well as journal article publications.

We noticed that the developers were reasoning on similar parts among different languages. Thus, we started the initiative to develop a multi-purpose language covering these repeating parts: coreLang [18]. coreLang includes the common concepts that are needed to model IT related networks, but on an abstract level. While specifying coreLang into more concrete languages, we recognized that we were still repeating work in certain domains. Thus, a more systematic approach for developing MAL-based languages is needed, leading to our research question: *RQ1: What are the properties of a MAL-based languages' ecosystem of that reduces redundant work?* Simultaneously, a method is needed how such an ecosystem can be maintained. Accordingly, we formulate our second research question: *RQ2: How can MAL-based languages be developed and maintained to preserve the ecosystem's characteristics?* The resulting ecosystem's purpose is to support the end-user in finding suitable languages for their demands and to reduce the effort for language developers by avoiding redundancy.

The rest of the paper is structured as follows: Next, we present the related work, which is on threat modeling in general and the systematic development of DSLs. To ease the understanding of MAL, we present the idea behind MAL, before we explain the fundamental properties of the ecosystem. This is followed by our vision for the future ecosystem as well as the explanation how single languages should be developed and maintained to fit into the ecosystem. Before we conclude our work, we discuss different insights regarding the ecosystem and possible changes to MAL to improve the ecosystem development.

2 Related Work

MAL languages count towards the domain of model-driven security engineering, in which many domain-specific languages exist [17, 27]. These languages usually facilitate a model of a system, which incorporates its components, the interaction among these, and security properties such as constraints, requirements, or threats. One common formalism for model checking and searching for constraint violations are attack trees [33, 21]. Apart from MAL, which is using this concept, there are several other approaches elaborating on attack graphs [22, 38].

Hitherto, we have united the approaches of attack graphs and system modeling in our previous work [9] by automatically generating probabilistic attack graphs based on a existing system specification. However, the used languages to create the attack graphs were hard-coded. Therefore, we have proposed MAL [16] that allows to create domain specific languages. So far, several languages have been built in MAL like vehicleLang [19], which allows modeling cyber-attacks on modern vehicles, or coreLang [18], which contains the most common IT entities and attack steps. Another approach is the automated creation of MAL languages by translating existing concepts to MAL [11].

As already indicated, the languages created with MAL are DSLs and we aim to develop an ecosystem around these languages. Hence, other related work elaborates on the development of DSLs and hierarchies of DSLs. do Nascimento et al. [25] performed a systematic mapping study on DSLs. Besides an increasing interest in DSLs, do Nascimento et al. notice that security related DSLs receive a lower attention than other DSLs e.g., related to software engineering purposes. Developing DSL for special purposes is a common endeavor in software engineering research. Accordingly, a broad range of DSLs has been developed [25]. Hence, researchers [34, 20] took a closer look at the different DSLs and distilled different reoccurring patterns.

In our work, we create hierarchies between DSLs, which is scarce in existing research. Nonetheless, different authors [15, 32] combine different DSLs in a hierarchical fashion, similar to our idea. In their approaches, each layer is used by different kinds of experts and the upper layer consumes the outputs of the lower layers. Thus, the developers on the lower layers do not need to have the overarching knowledge of the higher layers, while the developers of the higher layers do not need the detailed knowledge of the lower levels. Preschern at al. [31, 30] propose a meta-DSL which is similar to MAL as it provides a framework to develop other DSLs. However, while MAL's purpose is situated in the threat modeling domain, their meta-DSL is used for physical automation. In contrast to our work, they do not consider further dependencies between languages developed with their meta-DSL.

Cleenewerck [8] suggests defining so called "key words" to create components in DSLs that can be reused among other DSLs. This is similar to our idea of creating abstract DSLs that then are reused to create more specific languages. However, his approach is different in the sense that he proposes single fragments that are then reused, while we reuse the entire language. His approach has the advantage that the language designer can explicitly choose what to reuse, while our approach can cover concepts that the designer might have not considered.

3 The Meta Attack Language

Next, we give a short presentation of the MAL. For a detailed overview of the MAL, we refer readers to the original paper [16]. First, a MAL-based DSL contains the main elements that are encountered on the domain under study, those

are called `assets` in MAL. The assets contain `attack steps`, which represent the actual attacks/threats that can happen on 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 which are facilitated when the attack simulation is run. Attack steps 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.

Assets should also have relations between them in order for a model to be constructed, those relations are called `associations` in MAL. Inheritance between `assets` is also possible and each child asset inherits all the attack steps of the parent asset. It should be, nevertheless, mentioned that multiple inheritance is not currently supported in MAL. Additionally, the assets can be organized into categories for purely organization reasons.

In Listing 1, a short example of how a MAL-based DSL is presented. 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, 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 last lines of the example the `associations` between the assets are defined.

```
1  category System {
2    asset Network {
3      | access
4        -> hosts.connect
5    }
6
7    asset Host {
8      | connect
9        -> access
10     | authenticate
11       -> access
12     | guessPwd
13       -> guessedPwd
14     | guessedPwd [Exp(0.02)]
15       -> authenticate
16     & access
17   }
18
19   asset User {
20     | attemptPhishing
21       -> phish
22     | phish [Exp(0.1)]
23       -> passwords.obtain
24   }
25
26   asset Password extends Data {
27     | obtain
28       -> host.authenticate
29   }
30 }
31
32 associations {
33   Network [networks] *
34     <-- NetworkAccess -->
35     * [hosts] Host
36   Host [host] 1
37     <-- Credentials -->
38     * [passwords] Password
39   User [user] 1
40     <-- Credentials -->
41     * [passwords] Password
42 }
```

Listing 1: Exemplary MAL Code

4 Properties of Ecosystems

Jacobidis et al. [14] identified three streams of strategy research elaborating on ecosystems: the first stream focuses on a company and its environment [36]; the second stream concentrates on a particular innovation and the related actors [1]; the third stream centers around technological platforms and the actors interacting around them [6]. MAL and the languages created with it, can be understand as a technological platform and, thus, count towards the third stream.

For ecosystems in the third stream, Jacobidis et al. [14] identified three different types of stakeholders as a common property (P1): the platform sponsors, the complementors, and the consumers. For our envisioned ecosystem, the platform sponsor is the developer team of the MAL compiler, as their decisions on MAL's feature frame the opportunities that the MAL language developers (i.e., the complementors) can work with. Both together provide the final value to the users of the language respectively the consumers. From a complementor perspective, a property of ecosystems (P2) is the ability to reuse existing components and sometimes also combine them [14]. In our ecosystem, the complementor will choose from the different languages those, which are closest to their demands, and even combine different languages to address overarching demands (cf. Section 6). Moreover, this property is mirrored to the consumers (P3), that are free to choose from the ecosystem and combine different components [14].

Another important property of ecosystems (P4) is to provide an alignment structure for the different stakeholders creating the single parts of it [2], while preserving stakeholders' autonomy [14]. This is achieved by a modular architecture [4] and related design parameters can be set by the platform sponsor. Thus, an ecosystem provides processes and rules to solve coordination issues arising along the ecosystem evolution [14].

A fundamental rule of our ecosystem is that at the top, there are languages that cover criteria of a broad range of demands, while the deeper in the hierarchy the more specific the languages are. This reminds of the characteristics of a taxonomy [26], which is comprised by a set of n dimensions that consist of k characteristics. These characteristics are mutual exclusive in each dimension for the object that is classified. From a language development perspective, the mutual exclusivity still holds as the languages should be differentiable from each other. However, we relax the demand for one layer of characteristics to have several levels of concretization. As the relaxation is the only difference, the approach of Nickerson et al. [26] is still applicable to create an ecosystem of MAL-based languages. Thus, we present following our adoption of Nickerson et al.:

First, the meta-characteristic of the taxonomy –or rather of the DSL ecosystem in our case– needs to be determined. Nickerson et al. [26] point out that "The meta-characteristic is the most comprehensive characteristic that will serve as the basis for the choice of characteristics in the taxonomy." Thus, the meta-characteristic guides the development of the ecosystem and should be related to its purpose. On the one hand, it should support the developer of new MAL-based languages to situate their language properly with respect to existing languages. On the other hand, it should serve as aid for the language users to choose the

best suiting language. However, future development of the ecosystem might lead to changes of the purpose, as also mentioned by Nickerson et al. [26].

Second, the ending conditions need to be determined. The development of the ecosystem is a continuous effort. Consequently, there is no general ending condition. If a new language is added to the ecosystem, the ending condition is its successful situation within the ecosystem. However, the addition of a new language may also lead to changes in the ecosystem's structure. Within this work, we temporally extend these ending conditions to make sure that all existing MAL-based languages are included and to include imaginary examples to illustrate the application of the ecosystem.

Next, we develop the ecosystem itself. For the first version of the ecosystem, we follow a conceptual-to-empirical approach [26]. Therefore, we envision a structure that is detailed in Section 5 and incorporate the existing MAL-based languages into it. Following the conceptual-to-empirical approach is motivated by the facts that the number of MAL-based languages is yet not large enough to follow the empirical-to-conceptual approach and that the ecosystem structure shall inspire future language development. However, future alterations of the ecosystem's structure will obviously follow the empirical-to-conceptual approach, as a new object will join, which causes a revision of the structure.

5 A Vision for the Structure of an Ecosystem

Before, we have developed the characteristics that lead to the structure for the ecosystem. Next, we will present the outcome of the application of processes constituted in P4 (cf. Figure 1). We like to note, that this is just a vision and a future structure might look different. However, this vision should serve as an inspiration for the future development of MAL-based languages.

Before diving into the details of Figure 1, we like to discuss shortly the founding ideas. First, the structure follows the principle from general to specific. In other words, we situate languages that cover general domains on the top of the hierarchy and specify the languages to specific domains. This is thought to reduce the effort for creating languages. Second, we indicate cluster of languages. These cluster represent languages that belong to a certain domain. They are thought as help for the end-user to select the best suited language(s). This leads also to aspect three: each language is not planned as a silver bullet. Instead, the end-user chooses several languages that satisfy together the overall demand.

As indicated, we envision the future structure of MAL-based languages in a hierarchical structure. By definition, the origin of all MAL-based languages are the concepts of MAL (cf. Section 3). The second layer defines languages that cover certain overall concepts, like coreLang [18] representing common aspects of IT related networks. So far, coreLang depicts the only existing language on this layer. Another possible direction for future languages is to include business aspects into the threat modelling [10, 3].

The next layer concretizes the general-purpose languages. As a first step of concretization, we envision a differentiation between languages that cover the

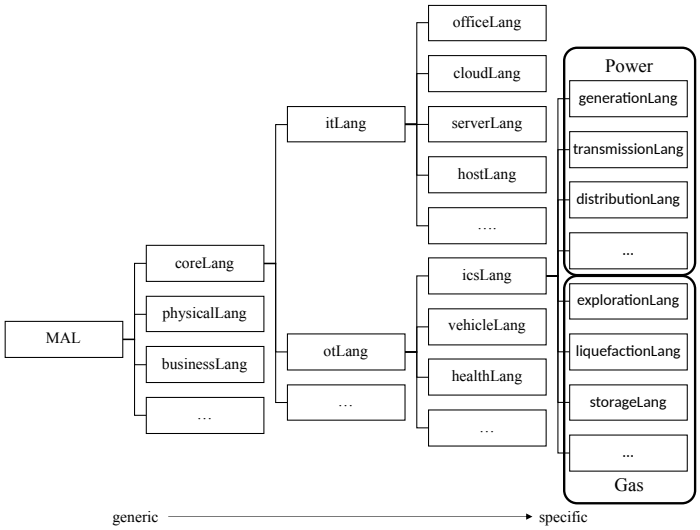

Fig. 1. Vision of an ecosystem for MAL-based languages

specifics related to IT and OT. The languages of the IT branch cover aspects related to classical IT. The OT branch is characterized by a much broader diversification of languages as the spectrum of languages will need to represent very different domains with different terminologies. So far, we have a language that is used to model the internals of vehicles (vehicleLang [19]) and a language that satisfies the need to simulate attackers in industrial environments (icsLang).

We assume that there will not be much need to specify the aforementioned layer of languages further. For example, to provide the power domain with necessary and to their terminology tailored assets, icsLang can be refined into certain languages, inspired by the facets included in the Smart Grid Architecture Model (SGAM) [7]. Similarly, this can be done for the gas domain, inspired by its value chain [37]. These most concrete languages will mainly rename already existing concepts to meet the domain specific terminology.

Hitherto, we have described languages that can be arranged into an inheritance hierarchy. However, we recognized possible languages that do not fit into this hierarchy as they cover orthogonal aspects that might or might not be of relevance for certain languages in the hierarchy. For example, enterpriseLang [39] codifies the techniques of MITRE ATT&CK[1] into MAL.

6 Single Language Development

So far, we have discussed the overall structure of the DSL ecosystem. However, the ecosystem is constituted by its languages. Therefore, we will discuss follow-

[1] https://attack.mitre.org/

ing how new languages can be integrated into the existing ecosystem and how existing languages should be maintained to keep the spirit of the ecosystem alive.

6.1 Developing a New Language

Before developing a new language, the developer needs to determine the requirements towards the language. In other words, which certain domain(s) should be covered by the language. Here, the techniques of domain analysis (e.g., [13]) might be of support for the developer. Based on this recognition, the developer needs to decide if an existing language or a combination of languages within the ecosystem are already (partly) satisfying the requirements.

If the requirements are already fulfilled, there is no need for an additional language. However, it might be the case that a language satisfies the requirements functionally, but the desired domain demands another terminology. In this case, we recommend an inherited language from the existing language, that simply introduces the common terminology of the domain. If the requirements are already fulfilled by a set of different languages, then a new language should be comprised of these languages (see Section 6.3).

Finally, if the requirements are not met, there is a need for a new language. To situate the language properly within the ecosystem, the developer should determine the language that covers the requirements best. Therefore, the developer should traverse the ecosystem's structure from its root by choosing always the best suiting domain. The traversing ends when the developer reaches a language that is too specific and, thus, does not satisfies the requirements anymore. Accordingly, the new language should be placed on the same level as the first language that is not satisfactory. If a further reorganization (cf. Section 4) of the following hierarchies is necessary, needs to be decided case by case. If several languages are identified as suitable for an extension. Then these languages need to be extended by several different new languages that are combined by the mechanisms described in Section 6.3.

6.2 Language Maintenance

Hitherto, we have discussed how developers should situate new languages within the ecosystem. However, it might be the case that an already existing language is covering the domain. In these cases, the reuse of existing languages should be prioritized. Nonetheless, it might be possible that the language does not contain all needed concepts. For example, a certain asset could be missing. Then, the language needs to be maintained.

To minimize site effects on other languages, we recommend adding new assets to languages that are at the lowest level of the ecosystem's structure. If the asset is shared among different languages on the same level of abstraction, then the asset should be moved up to the language of which these languages inherit from. However, if the there are other languages sharing the same parenting language and these languages do not contain the same asset, an additional abstract language could be necessary. This additional language would be then introduced

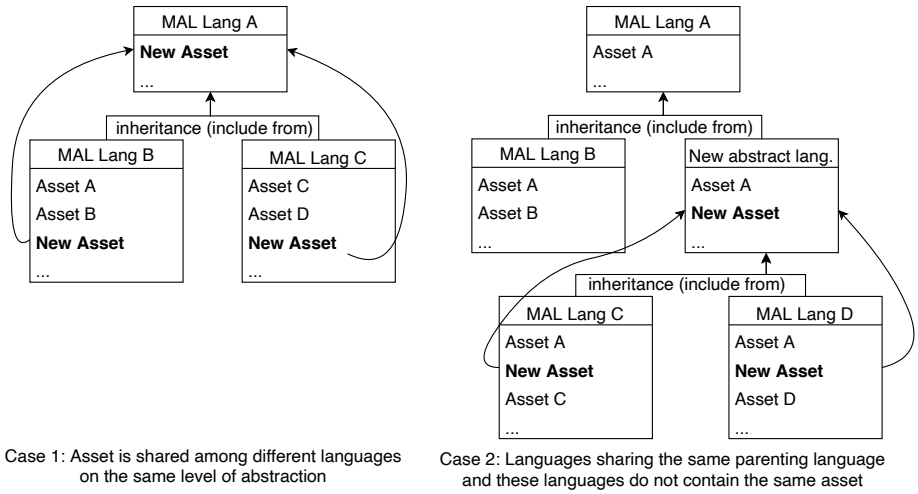

Fig. 2. Two cases of adding new assets in the ecosystem

as parent language for the languages sharing the certain asset. Those two cases are depicted in Figure 2.

Following this pattern will ensure the downward compatibility of the languages and, thus, create a more robust ecosystem. Drawbacks of this application are an increasing number of languages and deep inheritance hierarchies. Additionally, in such a deep inheritance chain, many unnecessary assets can be inherited when creating a new language. A possible solution for these drawbacks is the introduction of multiple inheritance and partial inheritance, respectively.

The deletion or the change of assets on the lowest level is without danger to the integrity of the ecosystem, as this will have no influence on other languages. However, adding, deleting, and changing on higher levels will have direct influence on all related lower languages. Therefore, these actions should be avoided on higher levels to impede undesired site effects. If such changes are necessary, a thorough analysis on the effects will be necessary and even a restructuring of ecosystem parts might be needed. Thus, we highly recommend to perform changes just on the lowest levels.

6.3 Combining Languages

As indicated before, the combination of different languages can satisfy the needs of the end-user. Usually, this will be the case if the end-user wants to model entire organizations that cover different domains. For example, for manufacturing organizations there will be the computing systems that are necessary for the manufacturing parts. Further, there will be other systems that support the administrative parts within the organization, like accounting or human resources. To perform simulations for such kind of organizations, the two languages covering these different aspects are needed. The ecosystem behaves in such cases

similar to the product line pattern [23], as the developer combines the languages from the ecosystem like the features for product lines.

The determination of the best suiting languages follows basically the same process as described in Section 6.1, except that the domain analysis [13] will result in several domains. Thus, the process needs to be performed for each of the domains to identify the best suiting language.

Next, the languages need to be linked to each other. First, the developer needs to identify the desired connection points between the languages. Connection points refer here to certain assets that are on the border (or close to) between two domains. These assets serve as transition for the attacker from one language to the other. Afterwards, it should be checked if there already exist links between the languages due to shared associations inherited from a parent language.

This examination can lead to four kinds of possible findings: First, the existing link is in line and no further actions are needed. Second, the found link is contradictory to our intentions and it needs to be removed. Actually, the languages need to be redesigned to achieve this. However, in future versions of MAL it might be possible to realize this without changing the languages, e.g., due to the specialization patter [34]. Third, there is no link between the asset and there is also non desired. Again, no actions need to be taken. Lastly, there is no link, but there should be one. In this last case, we see different options to create links between the languages that we will discuss next.

The first option is using inheritance between assets of the different languages as we did in powerLang [12]. The advantage is that the changes to the language are minimal, as just an extension between two assets has to be added. However, this is still a change on one language. Additionally, this works only if the asset does not already have an extension, as MAL does not support multiple inheritance (now). Another drawback is that the inheritance might cause unintended behavior if attack steps get overwritten. To sum up, the disadvantages prevail the advantages and, thus, inheritance should not be used to link languages.

The second option is to add a dedicated attack step that leads to another attack step in the other language. This gives a better control on how an attacker moves between languages and no unwanted side-effects arise. But changes to at least one language are needed. Further, a deeper understanding of both languages is needed to determine suitable attack steps for the transition. To sum up, this approach seems better, due to less unwanted side effects, but should also not be preferred as changes to the languages are necessary.

For the last option, the languages of the ecosystem need to be prepared first. Basically, the designer of the language foresees certain attack steps, preferably encapsulated in a designated asset, that serve as incoming and outgoing connection points. As these attack steps are of technical nature, they might be hidden to the end-users. A third language can be created in which the outgoing connection points are linked to the incoming connection points, similar to the adapter pattern [20]. Thus, no changes to the languages of the ecosystem would be necessary (cf. Figure 3). Alternatively, outgoing connection points can automatically be linked to incoming connection points, but we think that a human

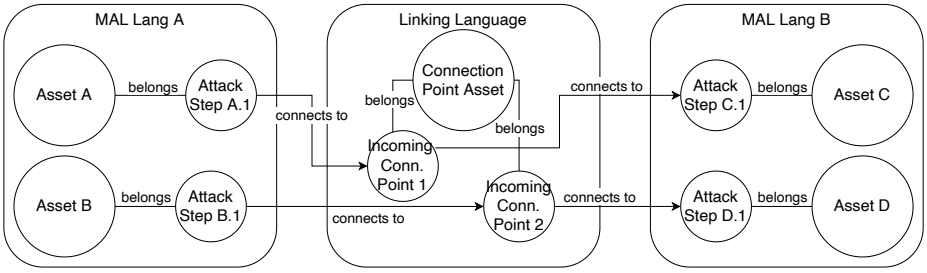

Fig. 3. Combining languages using an intermediate language

evaluation of which concrete points are connected to each other is necessary. A disadvantage of this technique is the additional effort for the language designers to consider and model the connections points. However, the effort will be worth it, as no changes to the languages will be necessary.

When combining languages, the question of responsibility for the maintenance of the linked languages needs to be answered and what happens if changes are introduced on those languages. The main responsibility for the maintenance of the languages falls to the developer but if proper versioning is used on them, then, changes on those languages should not adversely affect the linking languages.

6.4 First Experimentation Insights

As a first evaluation of our proposed approach, we developed icsLang. icsLang's purpose is to provide a set of assets that can be used to model industrial control systems (ICS) such as substations in power grids. Therefore, we considered existing MAL-based languages as starting point. The only feasible language was coreLang [18], as all other languages were in domains that did not fit.

The next step was to determine the needed assets within icsLang. Hence, we used ATT&CK for Industrial Control Systems[2] as inspiration and created for each of the mentioned assets a representation in icsLang. Next, we defined which icsLang assets inherit from which coreLang assets based on their description, where possible. Afterwards, we ensured that behavior, that was not already covered by the inheritance, was implemented in icsLang. We did not recognize the need to move new assets up to coreLang as all assets were specific for the ICS domain (such as physical processes), but we identified several bugs in coreLang that were corrected afterwards.

To check the applicability of icsLang, we tried to model a substation with it, which was no challenge. However, we recognized that the used terminology was slightly different. Therefore, we decided to create substationLang that inherits the concepts of icsLang and renames to the terminology used in substation. Moreover, we came along the concept of signaling that was not implemented in

[2] https://collaborate.mitre.org/attackics

icsLang. As this concept is not mutual exclusive to substation, but also exists in other ICS domains such as power generation, we moved the asset up to icsLang.

Using coreLang, icsLang, and substationLang in combination to model both IT and OT environments that interact with each other, caused no issues due to the inheritance structure. Consequently, we are not able to provide any deeper insights on the means for linking substantial different languages to each other.

7 Discussion

Before, we have presented the properties of the ecosystem, a vision for the future structure of the ecosystem, and the guidelines language developers should follow to create a sustainable ecosystem. Next, we discuss how our ecosystem meets the identified properties (cf. Section 4):

The first property, P1, is related to the stakeholders of the ecosystem. In our case, we have three different groups: the developer team of the compiler (platform sponsor), the language developers (complementors), and the users of the ecosystem (consumers). One might argue that security experts that provide knowledge to the language developers might be also stakeholders. We do not agree directly, since their role as experts does not create an interest in the ecosystem per se. But they might use the ecosystem for their analysis and, accordingly, they belong to the consumer group.

To guarantee the ability to reuse and combine the single components of the ecosystem (P2), we have described a certain set of rules that are thought to ensure these capabilities (cf. Section 6). However, these guidelines may foster a deep hierarchy of inheritance. Even if our first experiences with icsLang have shown that approximately three layers of inheritance might be sufficient, there are means to cope with deep hierarchies. On the one hand, one could introduce the concept of multiple inheritance to MAL. In that case, it would be possible to design language more finely structured and, thus, avoid several levels of inheritance. In contrast, this would lead to a bigger number of different languages, which need to be considered. On the other hand, MAL could be extended so that is possible to deactivate certain relations between assets or attacks, similar to the "language specialization patter" [34]. This would not reduce the inheritance depth but would enable the language developer to reduce the complexity of the language and opt out undesired behavior.

The latter would not only be useful in the context of inheritance, but also by combining languages that have a common parent language. It might be the case that in such settings, relations are inherited that are unintentional. Thus, removing them from the language would improve the language design.

The free choice and combination of the end-user (P3) is realized, due to the fact that each language of the ecosystem works standalone, at least for a certain demand. Moreover, if the languages of the ecosystem follow the guidelines described in Section 6.3, then the user will be easily able to combine different languages with each other. However, this interoperability is restricted by the effort the language developers spend to allow it. One option could be to demand

that every language needs to provide the features that are needed to achieve interoperability.

The last property, P4, is linked to means that achieve an alignment structure. Therefore, we present a process to develop a structure for an ecosystem of DSLs (cf. Section 4). The process is inspired by Nickerson et al. [26], thus the process itself can be considered as quality ensured. However, we cannot state the same for the resulting vision of the ecosystem. A solution to evaluate this, would be to split the existing MAL-based languages into two sets and use one for evaluation of the result [5]. Unfortunately, the number of existing MAL-based languages is too small so far. Thus, the structure of the ecosystem cannot be evaluated, which should be tackled in future work.

Finally, to ensure that future MAL-based languages will be added to our envisioned ecosystem and do not alter its properties, an appropriate tool support is vital to reduce the effort for developers. This includes classical coding features such as automatic code completion or refactoring capabilities. But also, more advanced features are needed, such as a central repository and automated provision of existing languages, similar to the mechanisms of maven.

8 Conclusion

It is very clear today that protecting our IT infrastructures is of great importance. One way of achieving this is by defensive protections but another one is through offensive security. A characteristic example is cyber-attack modeling and simulations and a framework that allows this is the MAL. MAL has been proposed in 2018 but since then it has noted an increase in usage and the number of MAL-based DSLs is constantly increasing year by year.

Because of that and because how closely, in terms of similarity, the different IT infrastructures of different domains are, a systematic way of developing such DSLs and maintaining them needed to be defined. In this paper, we have proposed our vision towards the ecosystem of DSL for threat modeling. This ecosystem, whose properties are the answer to our first research question, contains guidelines, best practices and lessons learned for the process of developing such languages, combining them, but also for the maintenance of them after the development phase has concluded. All those constitute the answers to our second research question, which was how MAL-based languages can be developed and maintained in order to preserve the ecosystem's properties. Then, in the last parts of this paper, a thorough discussion on what the plan for the future of this proposed ecosystem is, was done.

Regarding future work, an evaluation of our proposed ecosystem is on our plans and from the feedback we will get out of it more concrete advises for the development of languages inside the ecosystem could be made. Hitherto, we solely had a look at work contributing to reuse from the DSL domain. However, other possible inputs might be found in using components, method chunks, or in research related to product lines. Another point of future work is improvements to MAL itself, such as the addition of multiple inheritance as mentioned before.

Acknowledgement

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

References

1. Adner, R.: Match your innovation strategy to your innovation ecosystem. HARVARD BUSINESS REVIEW **84**(4), 98–107 (2006)
2. Adner, R.: Ecosystem as structure: An actionable construct for strategy. Journal of Management **43**(1), 39–58 (2017)
3. Aldea, A., Vaicekauskaite, E., Daneva, M., Piest, J.S.: Assessing resilience in enterprise architecture: A systematic review. In: 24th International EDOC Conference. pp. 1–10. IEEE CS, Los Alamitos, CA, USA (2020)
4. Baldwin, C.Y., Clark, K.B.: Design rules: The power of modularity, vol. 1. MIT press (2000)
5. Barbosa, A., Santana, A., Hacks, S., Stein, N.v.: A taxonomy for enterprise architecture analysis research. In: 21st ICEIS. vol. 2, pp. 493–504. SciTePress (2019)
6. Ceccagnoli, M., Forman, C., Huang, P., Wu, D.J.: Cocreation of value in a platform ecosystem! the case of enterprise software. MIS Quarterly **36**(1), 263–290 (2012)
7. CEN-CENELEC-ETSI, Smart Grid Coordination Group: Smart grid reference architecture (2012)
8. Cleenewerck, T.: Component-based dsl development. In: Generative Programming and Component Engineering. pp. 245–264. Springer, Berlin, Heidelberg (2003)
9. Ekstedt, M., Johnson, P., Lagerström, R., Gorton, D., Nydrén, J., Shahzad, K.: securiCAD by foreseeti: A CAD tool for enterprise cyber security management. In: 19th International EDOC Workshop. pp. 152–155. IEEE (2015)
10. Goluch, G., Ekelhart, A., Fenz, S., Jakoubi, S., Tjoa, S., a. T. Muck: Integration of an ontological information security concept in risk aware business process management. In: 41st HICSS 2008. pp. 377–386 (2008)
11. Hacks, S., Hacks, A., Katsikeas, S., Klaer, B., Lagerström, R.: Creating meta attack language instances using archimate: Applied to electric power and energy system cases. In: 23rd International EDOC. pp. 88–97 (2019)
12. Hacks, S., Katsikeas, S., Ling, E., Lagerström, R., Ekstedt, M.: powerlang: a probabilistic attack simulation language for the power domain. Energy Informatics **3**(1) (2020)
13. Hjørland, B.: Domain analysis in information science. J Doc **58**(4), 422 – 462 (2002)
14. Jacobides, M.G., Cennamo, C., Gawer, A.: Towards a theory of ecosystems. Strategic Management Journal **39**(8), 2255–2276 (2018)
15. Johanson, A.N., Hasselbring, W.: Hierarchical combination of internal and external domain-specific languages for scientific computing. In: ECSAW. ACM (2014)
16. Johnson, P., Lagerström, R., Ekstedt, M.: A meta language for threat modeling and attack simulations. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. p. 38. ACM (2018)
17. Jürjens, J.: Secure systems development with UML. Springer Science & Business Media (2005)
18. Katsikeas, S., Hacks, S., Johnson, P., Ekstedt, M., Lagerström, R., Jacobsson, J., Wällstedt, M., Eliasson, P.: An attack simulation language for the it domain. In: Graphical Models for Security. pp. 67–86. Springer, Cham (2020)

19. Katsikeas, S., Johnson, P., Hacks, S., Lagerström, R.: Probabilistic modeling and simulation of vehicular cyber attacks : An application of the meta attack language. In: 5th ICISSP (2019)
20. Keepence, B., Mannion, M.: Using patterns to model variability in product families. IEEE Software **16**(4), 102–108 (1999)
21. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: International Workshop on Formal Aspects in Security and Trust. pp. 80–95. Springer (2010)
22. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Comp Sci Rev **13**, 1–38 (2014)
23. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
24. Morikawa, I., Yamaoka, Y.: Threat tree templates to ease difficulties in threat modeling. In: 14th NBiS. pp. 673–678 (2011)
25. do Nascimento, L.M., Viana, D.L., Neto, P., Martins, D., Garcia, V.C., Meira, S.: A systematic mapping study on domain-specific languages. In: The Seventh ICSEA. pp. 179–187 (2012)
26. Nickerson, R.C., Varshney, U., Muntermann, J.: A method for taxonomy development and its application in information systems. EJIS **22**(3), 336–359 (2013)
27. Paja, E., Dalpiaz, F., Giorgini, P.: Modelling and reasoning about security requirements in socio-technical systems. Data Knowl Eng **98**, 123–143 (2015)
28. Petermann, T., Bradke, H., Lüllmann, A., Poetzsch, M., Riehm, U.: Was bei einem Blackout geschieht: Folgen eines langandauernden und großflächigen Stromausfalls, vol. 662. Büro für Technikfolgen-Abschätzung (2011)
29. Petit, J., Shladover, S.E.: Potential cyberattacks on automated vehicles. IEEE Transactions on Intelligent Transportation Systems **16**(2), 546–556 (2015)
30. Preschern, C., Kajtazovic, N., Kreiner, C.: Efficient development and reuse of domain-specific languages for automation systems. Int. J. Metadata Semant. Ontologies **9**(3), 215–226 (2014)
31. Preschern, C., Leitner, A., Kreiner, C.: Domain specific language architecture for automation systems: An industrial case study. In: 8th ECMFA. pp. 1–12 (2012)
32. Prähofer, H., Hurnaus, D.: Monaco — a domain-specific language supporting hierarchical abstraction and verification of reactive control programs. In: 2010 8th IEEE International Conference on Industrial Informatics. pp. 908–914 (2010)
33. Schneier, B.: Attack trees. Dr. Dobb's journal **24**(12), 21–29 (1999)
34. Spinellis, D.: Notable design patterns for domain-specific languages. J. Syst. Softw. **56**(1), 91 – 99 (2001)
35. Stellios, I., Kotzanikolaou, P., Psarakis, M., Alcaraz, C., Lopez, J.: A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. IEEE Communications Surveys Tutorials **20**(4), 3453–3495 (2018)
36. Teece, D.J.: Explicating dynamic capabilities: the nature and microfoundations of (sustainable) enterprise performance. Strategic Management Journal **28**(13), 1319–1350 (2007)
37. Weijermars, R.: Value chain analysis of the natural gas industry: Lessons from the us regulatory success and opportunities for europe. J NAT GAS SCI ENG **2**(2), 86 – 104 (2010)
38. Williams, L., Lippmann, R., Ingols, K.: GARNET: A graphical attack graph and reachability network evaluation tool. Springer (2008)
39. Xiong, W., Legrand, E., Åberg, O., Lagerström, R.: Cyber security threat modeling based on the mitre enterprise att&ck matrix. submitted to SoSyM Journal (2020)