ASTo: A Tool for Security Analysis of IoT Systems

Orestis Mavropoulos, Haralambos Mouratidis, Andrew Fish, Emmanouil Panaousis
School of Computing, Engineering and Mathematics
University of Brighton, Brighton, UK
{o.mavropoulos,h.mouratidis,andrew.fish,e.panaousis}@brighton.ac.uk

***Abstract*—In this paper, a software tool for security analysis of IoT systems is presented. The tool, named ASTo (Apparatus Software Tool) enables the visualization of IoT systems using a domain-specific modeling language. The modeling language provides constructs to express the hardware, software and social concepts of an IoT system along with security concepts. Security issues of IoT systems are identified based on the attributes of the constructs and their relationships. Security analysis is facilitated using the visualization mechanisms of the tool to recognize the secure posture of an IoT system.**

***Index Terms*—IoT Security, IoT Software Tool, IoT visualization**

I. Introduction

Internet of Things (IoT) systems are expected to be composed of thousands of millions of interconnected objects. IoT objects could be humans, devices or cloud based services. In order to perform a type of analysis in a system of that magnitude and complexity, engineers require domain specific methodologies and tooling.

In this paper, we present a software tool for security analysis of IoT systems. The software tool is named ASTo [1] (Software Tool for Apparatus Framework). It was developed in order to support the Apparatus framework [2]. ASTo is an open source project under the MIT license. It was developed using the Electron framework [3] and the Cytoscape.js [4] library. The tool is based on the metamodels of Apparatus framework in order to express IoT systems. The tool is used to identify the assets along with the threats on an IoT system. Using that information a security engineer is able to propose security controls to reduce the attack surface of the IoT system.

The paper is organized as follows: Section II presents the related work that is made in IoT software tools. Section III presents the architecture of the tool. In section IV the GUI of the tool is introduced along with the functionality of the tool. Finally, Section V concludes the paper and proposes future work.

II. Related work

The literature provides us as with a number of IoT related tools that were developed to support specific methodologies and frameworks.

ThingML is developed as a domain-specific modeling language which includes concepts to describe both software components and communication protocols. The formalism used is a combination of architecture models, state machines, and an imperative action language [5]. ThinkML is supported by a set of open source tools that are built using the Eclipse Modeling Framework. Apparatus was developed for security analysis of IoT systems. As such it requires concepts that can be used to express "things", services, threats but also the social aspects of IoT systems. On the other hand, ThingML was developed to model the hardware, software components and communication protocols of IoT systems. It does not have concepts to model social or security components of IoT, such as users, stakeholders, threats or vulnerability.

ASSIST is an agent-based simulator of Social Internet of Thing (SIoT) [6]. The idea behind SIoT is that smart objects will connect with each other to form social networks. ASSIST uses an agent-based approach by defining three types of agents. Device Agents, Human Agents, and Task Agents. While ASSIST can be used to express the social components, it was not designed with security analysis in mind. As such it cannot be used to express security components.

SenseSim is an agent-based and discrete event simulator for IoT [7]. It can be used to simulate heterogeneous sensor networks and observe the changing phenomena. The simulator using a perception model understands phenomena such as weather changes, fires or car traffic and can react according to them. SenseSim was developed to augment the perception of sensor networks. While those networks can be configured to react to security threats, the tool was not designed to facilitate security analysis.

III. Tool modeling language

The Apparatus framework defines two metamodels to describe IoT systems. The first metamodel describes an IoT system at the design phase. During the design phase a security engineer models how the IoT should be without specifying the hardware architecture of the system. At that phase, the engineer is able to identify the assets of the system and the threats that impact them. The second metamodel describes an IoT system at the implementation phase. During the implementation phase, a security engineer has more information about the hardware architecture and the topology of the system. At that phase, the security engineer can identify vulnerabilities on the services or the devices of the system. Using this information, specific security controls can be proposed in order to secure the system. Each phase of analysis is defined using a metamodel. The metamodel enforces semantic rules on how model instances are created and is presented via a UML class diagram. Each class represents a concept that can be used to describe specific objects of IoT systems. Each concept has a number of properties that further describe it in the system.

A. Design phase metamodel

The design phase metamodel is used to express the early model of an IoT system. It represents the early development stages, where a system is being designed. During that stage, engineers require high-level concepts to represent a system. The design phase concepts of the metamodel are divided into different modules based on their thematic approach. All the concepts of the metamodel unless otherwise stated have the property of *description* which describes the concept in the system. The modules of the design phase metamodel along with their concepts are the following:

Design Network module: is used to model network objects of IoT systems. The Network module is considered the core module of the metamodel. Every other module is designed as an extension to the Network module. This modeling choice was made to give emphasis to the interconnecting nature of IoT systems. The Network module is represented with the color blue in Fig. 1. It is composed of the following:

1) **Thing:** is an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks [8].
2) **Micronet:** represents environments that a security engineer can configure in terms of their security. A Micronet is a managed environment that constitutes a collection of Things necessary for an IoT system to perform a function. Examples of Micronets are a smart home, an agricultural network of sensors or company's internal network. The property of the Micronet is: *(1) purpose:* describes the functionality of the Micronet.
3) **Net:** While Micronet represents environments that have their security configured by a security engineer, Net represents environments that their security configuration is not known. Since their security configuration cannot be verified, Net is considered compromised. Communication between Micronet and Net is never trusted and must always be verified. Examples of the Net are external networks to the IoT system that a security engineer has little knowledge of, such as a third party cloud infrastructure or hostile deployment environments.
4) **Data:** information that is produced or stored by a Thing. Examples of Data are information stored in a database, user passwords or environmental data collected by Things.

Design Social module: extends the Network module in an object-oriented manner with social concepts. Social concepts are used to model users and stakeholders. The social module is represented with the color gray in Fig. 1. It is composed of:

1) **Actor:** is used to represent people or groups of people that interact with an IoT system [9]. An Actor may never be malicious. To represent malicious Actors the Security module concept of *Malicious Actor* is used. The concept of Actor can be used to represent groups of people with different privileges, such as root users or the administration personnel of a University. The property of the Actor is the following: *(1) intent*: describes actor's intention by interacting with the IoT system.

Design Security module: extends the Network and Social modules with security concepts. The security concepts are used to model threats, assets, security controls and attackers in an IoT system. The concepts used by the Security module are heavily influenced by Secure Tropos security concepts [9]. The security model is represented with the color purple in Fig. 1. The security module is composed of:

1) **Asset:** any Actor, Thing or Data of the system that either (1) is considered valuable by the stakeholders and needs to be protected; of (2) a malicious actor wants; or (3) acts as a stepping stone to further attacks. Examples of Assets are the access credentials known by an actor, sensitive user information stored in a database or a sensor that has read/write privileges to a server.
2) **Threat:** a malicious function or system that has the means to exploit a vulnerability of a legitimate system. A Threat can only target an Asset of the IoT system. The property of the Threat is: *(1) threatType:* represents the classification of the Threat according to the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Elevation of Privilege) [10]. It takes an enumerated value.
3) **Constraint:** a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of a multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives [9]. The Constraint has the following property: *(1) propertyType:* how the Constraint is classified according to the extended CIA (Confidentiality, Integrity, Authentication, Authorization, Non-repudiation, Availability) [11]. It takes an enumerated value.
4) **Malicious Actor**: is a person with malicious intent. Malicious Actors are used to representing attackers or insider threats. The concept of the Malicious Actor is a generalization of the concept Actor.

B. Implementation phase metamodel

During the implementation phase, a security engineer has more detailed knowledge of an IoT system. For example, during implementation, the security engineer knows the type of network protocols that are used by the system or the version of the software applications that provide services to the system. The implementation phase metamodel expands the design phase metamodel's concepts with more properties and introduces several new concepts. The design phase concept of Thing is translated into the concepts of Device and Network Connection. The concepts of Vulnerability and Mechanism are

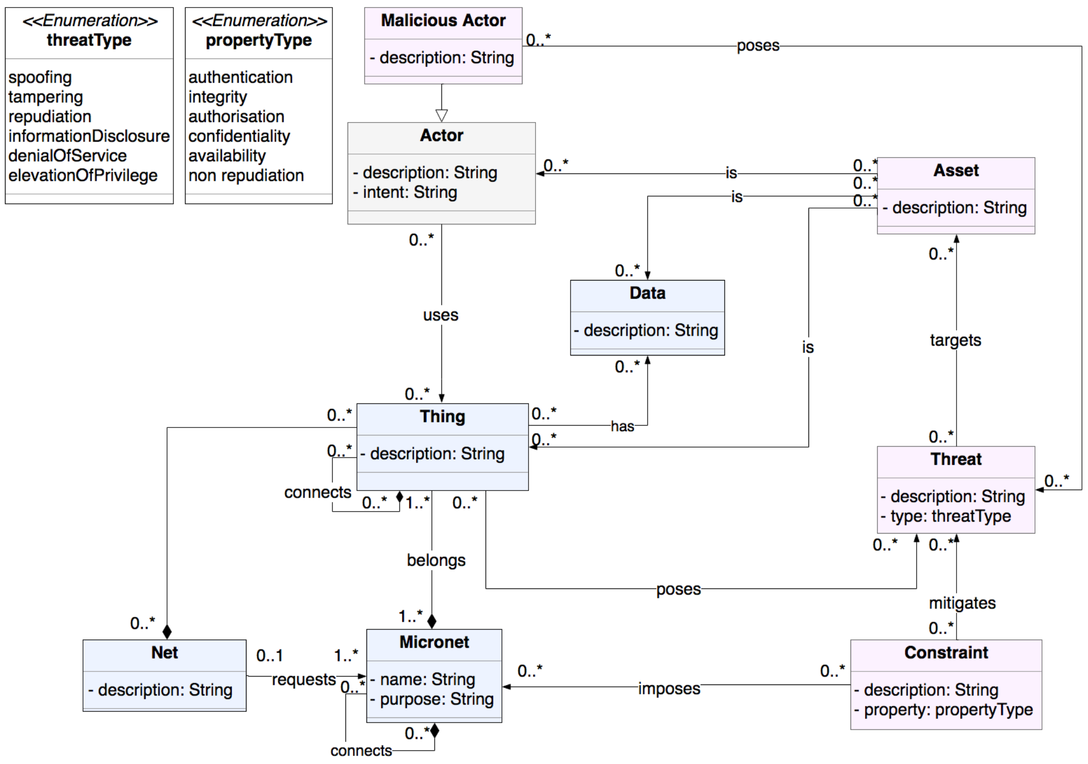


Fig. 1. Design phase metamodel

introduced. The metamodel of the implementation phase is shown in Fig. 2.

Implementation Network Module

1) **Device**: is an object of the physical world (physical things). Objects of the information world (virtual thing) are represented as functions of the Device. A restriction on the model is that Devices can only have a single functionality. If a Device has more than one function, each function has to be represented as a different Device. For example, a laptop running a server (1st function) and client (2nd function), has to be expressed as two separate *virtual* devices that belong to a parent *physical* device that is the laptop. The properties of the Device are: *(1) aspect*: declares whether the Device is a single node, or composed of sub-nodes. The aspect *physical* means that a Device is a parent Device and may be composed by more *virtual* Devices. *(2) layer*: the conceptual layer of the IoT architecture to which the Device belongs. APPARATUS uses a three-layer architecture that consists of the Application Layer, Network Layer and the Perception Layer [12], [13]. *(3) type*: defines the hardware type of the Device. For example, a Device type may be a sensor, a mobile phone or a server; *(4) service*: is the type of role or operation that the Device performs for the system. This value may include network services such as *ssh*, *ftp*, data processing filtering and relaying of data; *(5) input*: what is required in order for the node to perform its role or operation. It takes an enumerated value as an input that is *dataEnvironmental, dataDigital, command, action, notification, trigger*; *(6) output*: is the result of the Device operation or role. It may take the same values as the *input* property. *(7) update:* how the Device is having its software updated. The updates can be automatic, require a specific action or false.
2) **Network Connection**: the type of network communication used between the Devices. The properties of the network connection are: *(1) description*: the type of connection, it can either be *wireless*, signifying a connection using a wireless protocol or *cable*, signifying a connection using a wired medium. It takes an enumerated value as an input; *(2) listOfProtocols*: is a list of the supported network protocols by the network connection. It takes an array of string values as an input, each value in the array represents a supported network protocol.
3) **Net**: identical concept to the design phase of Net.
4) **Micronet**: similar concept to the design phase concept of Micronet. It is expanded with the property of *state*. *(1) state:* is the nature of a Micronet in terms of the IoT system's gateway layer. The *state* can either be *dynamic*, meaning that the Devices in the system change network domains during their usage or *static* meaning that the Devices in the system do not change network domains. Examples of *dynamic* IoT systems are networks of vehicular fleets and mobile devices since devices in such networks change their geographical location. Examples of *static* IoT systems are smart homes and industrial IoT

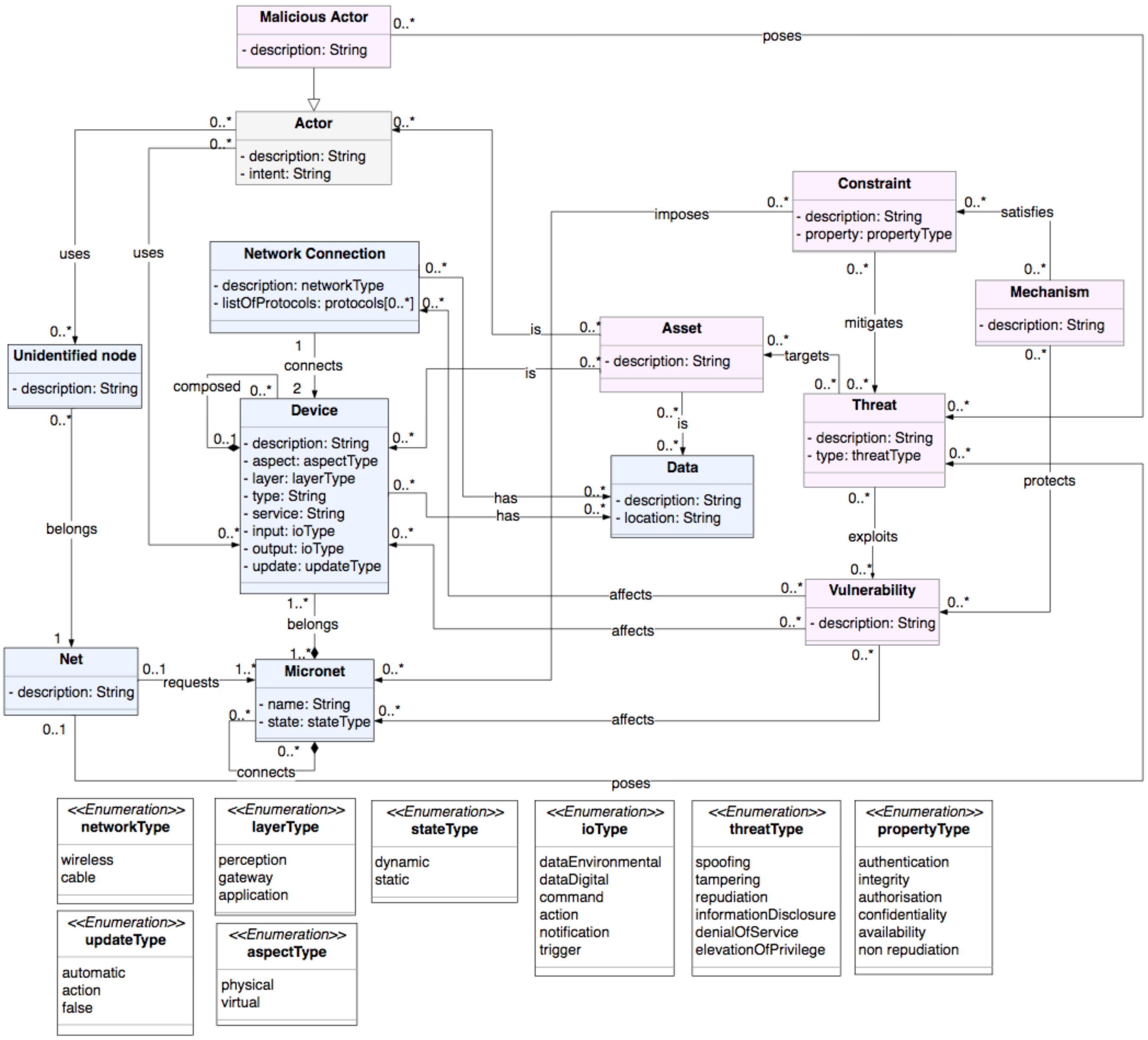


Fig. 2. Metamodel of the implementation phase

systems since devices in such networks are stationary during their life cycle.

5) **Unidentified Node**: is a Device that is not directly connected to a Micronet. It may be a malicious Device or a legitimate Device that is not authenticated in the system. For example, it can be an unauthenticated laptop from a legitimate user trying to connect to an office network or it can be a laptop operated by a malicious attacker trying to compromise the network.
6) **Data**: similar concept to the design phase concept of Data. The concept is expanded with the property of *location*: corresponds to the geographical location of the Data stored in the Device. It can be used to represent if Data are physically stored inside a network or are hosted by a third party service in another geographical region.

Implementation Social Module

1) **Actor**: identical concept to design phase concept of Actor.

Implementation Security Module

1) **Asset**: identical concept to design phase concept of Asset.
2) **Threat**: identical concept to design phase concept of Threat.
3) **Vulnerability**: a software, hardware or usage policy weakness that can be exploited by an adversary towards compromising a system. Hardware and software Vulnerabilities can be identified using techniques such as

penetration testing.

4) **Constraint**: identical concept to design phase concept of Constraint.
5) **Mechanism**: a security mechanism that implements one or more security constraints. A Mechanism, when implemented, protects against one or more Vulnerabilities.
6) **Malicious Actor**: identical concept to design phase concept of Malicious Actor.

IV. Presentation of ASTo

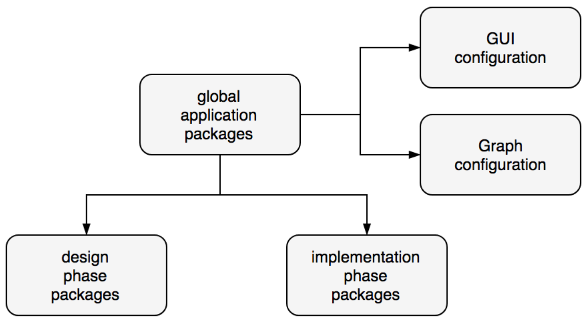


Fig. 3. ASTo architecture

The front-end representation of the APPARATUS model instances is based on graph diagrams. Each graph is made up as a cluster of *nodes* connected with *edges*. Nodes represent the concepts of the metamodel, while edges represent their relationships.

The GUI layout of the tool during the implementation phase is shown in Fig. 4. The main window of the tool is divided into three parts. The first part is the Control Tab. In the Control Tab, a user can select the functions of the tool by pressing buttons. Functions of the Control Tab are the addition/deletion of nodes and edges, highlighting specific node groups and validation of the system. The second part of the window is the Graph Tab. In the Graph Tab, the IoT system under analysis is presented as a graph diagram. The Graph Tab is dynamic to user interaction. The third part is the Action Tab. In the Action Tab, any type information about the state of the graph is displayed. The user has access the command prompt of the application. The command prompt can be used to input search queries for the graphs or type in commands. Valid commands include the functionality of buttons of the Control Tab.

A. Functionality of ASTo

The tool is developed using a modular approach. Each module contains everything necessary to execute only one aspect of the desired functionality. The packages as shown in Fig. 3, are divided into Global packages, Design phase packages and Implementation phase packages. The Global packages are shared between the design and implementation phase analysis. The Design phase packages can only be used for the design phase analysis, while the Implementation phase packages are used only for the implementation phase analysis. Design phase and Implementation phase packages provide similar functionality to the tool but use different metamodels as an input for their analysis.

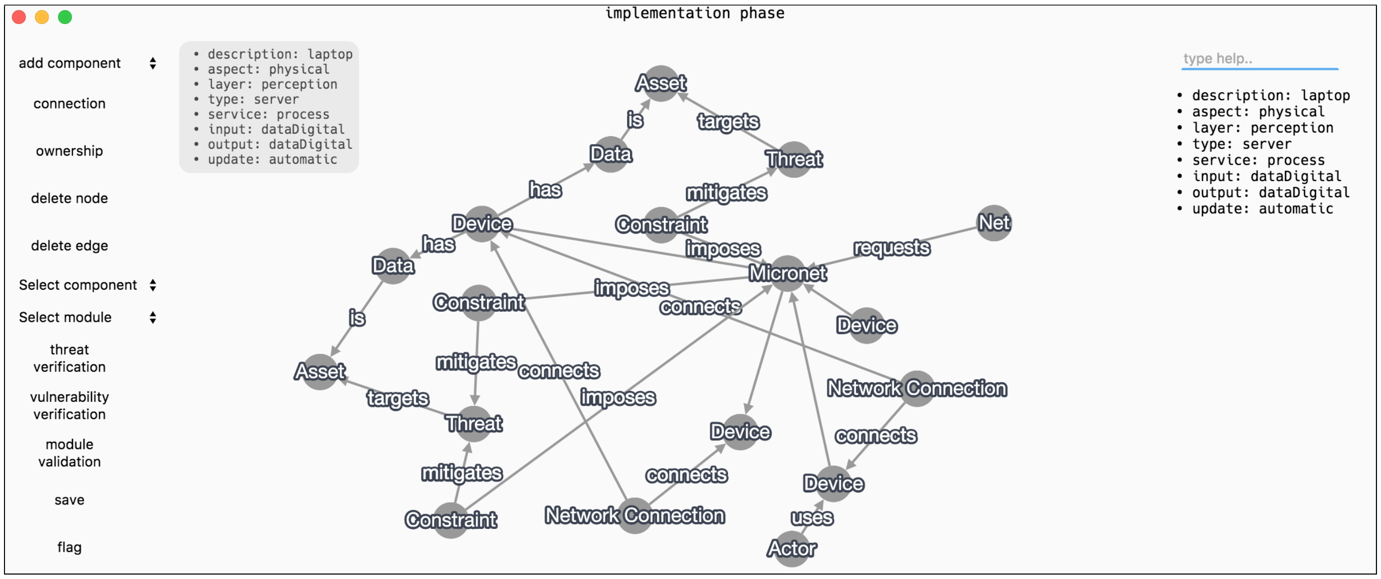
1) Global packages: Global packages provide the majority of the functionality of the Tool. Global packages are divided into two parts. Style and Analysis packages. Style packages are used to configure the appearance of the tool. This was made to enable users with disabilities to configure the tool according to their needs. Analysis packages are used to perform analysis on the IoT graphs.

Style packages

1) *GUI configuration:* every element in the GUI of the application is configurable. A user can change the color values of the elements, the font style, and size or choose to which elements of the GUI to display.
2) *Graph configuration:* the style of the graph is configurable. A user can change the size, color, and shape of the nodes. The size, style, and color of the edges, along with the font style, size and color of graph's labels can be changed.

Analysis packages

1) *graph manipulation:* the tool supports manipulation of the graph in a graphical manner. The user can move nodes, add nodes and edges and make changes on the properties of the nodes.
2) *highlight nodes:* the desired nodes are highlighted while the rest of graph has its opacity reduced.
3) *highlight modules:* the nodes that are part the same metamodel module are highlighted. For example, the engineer can choose to highlight only the network module nodes or the security module nodes.
4) *highlight neighbors:* the neighbors of the selected node are highlighted.
5) *attribute search:* the nodes with the selected attribute are highlighted.
6) *flag properties:* the nodes that have the specified properties are highlighted. This functionality is useful when looking for patterns in the graph. For example, we might be interested in all the *Devices* that use *wireless network connection* which support the *telnet* communication protocol.
7) *hover node information:* while hovering a node, its properties are displayed in an adjacent container.
8) *layout placement:* the layout of the graph can be configured using placement algorithms. Those algorithms are provided by the Cytoscape library [4].
9) *export/import models:* the graphs can be exported or imported as JavaScript Object Notation (JSON) files.
10) *Threat verification:* the tool can verify if the identified Threats are mitigated by Constraints. The package displays an overview of the number of Threats of the graph along with the mitigated Threats number.
11) *model checking:* graphs are validated according to the rules of metamodels of the tool in an asynchronous manner.



- description: laptop
- aspect: physical
- layer: perception
- type: server
- service: process
- input: dataDigital
- output: dataDigital
- update: automatic

- description: laptop
- aspect: physical
- layer: perception
- type: server
- service: process
- input: dataDigital
- output: dataDigital
- update: automatic

Fig. 4. Implementation phase GUI

12) *imported model checking:* the tool can check if imported models comply with the rules of the chosen metamodel.
13) *Vulnerability verification:* this package is only supported on the Implementation phase. The Vulnerability verification package returns an overview of the total Vulnerabilities of the graph and which Vulnerabilities are mitigated.

V. Conclusion and Future work

The present paper introduces ASTo, a visualization tool that enables security analysis of IoT systems during the design and the implementation phase. The tool was developed to support the Apparatus framework. A user can analyze the security of an IoT system using the concepts from the metamodel of the Apparatus framework. The security analysis is based on identifying the Assets of an IoT system and then defining the attack surface of the system using Threats and Vulnerabilities. To reduce the attack surface, users introduce security controls. The tool enables users to choose different visualization functions in order to analyze large systems. Users can assess the validity of the security controls along with the percentage of mitigation they introduce in the attack surface of the system.

To further improve our tool, we plan to develop a security assistant bot to incorporate in the tool. The security assistant will provide security suggestions and other relevant information to the security engineer based on the information of the IoT system. The security suggestions will be linked to specific properties of the system, that the engineer will be able to configure. The security assistant will be used to contribute domain specific security knowledge that a security engineer may lack. Once the security assistant is implemented, we aim to publish an evaluation survey. The survey will be performed by users of different security expertise to evaluate both the visualization of the tool and the benefit of the security assistant.

References

[1] O. Mavropoulos, “Asto,” https://github.com/Or3stis/apparatus, 2016.

[2] O. Mavropoulos, H. Mouratidis, A. Fish, E. Panaousis, and C. Kalloniatis, “Apparatus: Reasoning about security requirements in the internet of things,” *Advanced Information Systems Engineering Workshops*, vol. 249, pp. 219–230, 2016.

[3] GitHub Inc, “Electron,” http://electron.atom.io/, 2015.

[4] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, “Cytoscape.js: a graph theory library for visualisation and analysis,” *Bioinformatics*, vol. 32, no. 2, p. 309, 2016.

[5] A. Vasilevskiy, B. Morin, O. Haugen, and P. Evensen, “Agile development of home automation system with thingml,” in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016.

[6] P. Kasnesis, L. Toumanidis, D. Kogias, C. Z. Patrikakis, and I. S. Venieris, “Assist: An agent-based siot simulator,” in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 2016, pp. 353–358.

[7] M. Dyk, A. Najgebauer, and D. Pierzchala, “Sensesim: An agent-based and discrete event simulator for wireless sensor networks and the internet of things,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015.

[8] ITU, *Global Standards Initiative on Internet of Things Recommendation ITU-T Y.2060*, ITU Std., 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559

[9] P. Giorgini and H. Mouratidis, “Secure tropos: A security-oriented extension of the tropos methodology,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 02, pp. 285–309, 2011.

[10] A. Shostack, *Threat modeling: Designing for security*. Indianapolis, IN: John Wiley & Sons, 2014.

[11] J. Andress, *The basics of information security: Understanding the fundamentals of Infosec in theory and practice*. United States: Syngress Media,U.S., 2014.

[12] Z. Yang, Y. Yue, Y. Yang, Y. Peng, X. Wang, and W. Liu, “Study and application on the architecture and key technologies for iot,” in *2011 International Conference on Multimedia Technology*, 2011, pp. 747 – 751.

[13] W. Miao, L. Ting-lie, L. Fei-Yang, S. Ling, and D. Hui-Ying, “Research on the architecture of internet of things,” in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5. Chengdu: IEEE, 2010, pp. 484–5.