TrustMUSE: A Model-Driven Approach for Trust Management

Mark Vinkovits, René Reiners, Andreas Zimmermann

TrustMUSE: A Model-Driven Approach for Trust Management

Mark Vinkovits, René Reiners and Andreas Zimmermann

Fraunhofer FIT
User Centered Ubiquitous Computing
Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{mark.vinkovits, rene.reiners,
andreas.zimmermann}@fit.fraunhofer.de

Abstract. With the increasing acceptance of Trust Management as a building block of distributed applications, the issue of providing its benefits to real world applications becomes more and more relevant. There are multiple Trust Management frameworks ready to be applied; however, they are either unknown to developers or cannot sufficiently be adapted to applications' use cases. In our research, we have defined a meta model to modularize Trust Management, where each element in the model has clearly defined dependencies and responsibilities – also enforced by a complete API. Based on this model, we were able to develop a process supported by a number of tools that enables non-security expert users to find an applicable Trust Management solution for their specific problem case. Our solution – collectively called the TrustMUSE system – has evolved over an iterative user-centered development process: starting with multiple focus group workshops to identify requirements, and having multiple prototypes to conduct usage observations. Our user evaluation has shown that our system is understandable for system designers, and is able to support them in their work.

Keywords. Trust Management, Model-Driven Architecture, User-Centered Design, Meta Model, Usable Security

1 Introduction

During recent years, Trust Management has gained increasing attention and is becoming more and more accepted as an essential building block of distributed systems, especially for the Internet of Things [18]. There is a vast number of research results in the field, and a variety of use cases is covered by these. However, these solutions rarely find their way into real-world applications. One possible explanation for this, as already identified by others, is the general lack of security expertise present during the development of applications [19]. As a result, even though Trust Management frameworks provide sound procedures for common threats, they are not integrated into systems, because developers simply are not aware of them. To avoid this problem, two basic steps are necessary: first, we need to ensure that designers of these applications have a general understanding of Trust Management and of available solutions;

secondly, developers need support with regard to developing or integrating Trust Management frameworks. These are the points we address with our research by developing a model-driven process that helps selecting and integrating applicable Trust Management solutions into specific application use-cases.

We organized our research according a user-centered design methodology [1]: first, we identified our users to be non-security expert developers. Focusing on this user group, we arranged a number of requirement sessions with focus groups using different scenarios as presented in [2, 3]; this way we gained a broad view on the challenges developers had. During the development of our approach, we concentrated on regularly evaluating also intermediate results to ensure that we did not lose focus of our target end users. Within this user-centered design process, this paper presents the state at the end of the first large iteration, where all intermediate results of the individual components have already been evaluated once, and the first evaluation of the whole integrated system took part.

From the first requirements workshops, that were the first step of our user-centered process, we found that there already was a general understanding of Trust Management [2] and that a structured, hierarchical view was requested to gain an overview of available solutions. Therefore, we chose to develop model-driven approach for working with Trust Management, called the TrustMUSE system (Trust Management Usable Software suitE). By applying grounded theory, we first composed a meta model of Trust Management as underlying structure of our system (The definition process, a first version and a validation of the elements of the model have already been presented in [4]). This meta model identified common aspects of different Trust Management frameworks; thereby, it enabled a more structured and focused view on the benefits and characteristics of distinct realizations. To be able to use the conceptual elements of the meta model in a more specified model-driven approach, we improved the concept into the TrustMUSE Model by defining the APIs of each element; these APIs not just abstract the functionality of the individual elements, but also concentrate on integration and implementation issues.

Looking at the TrustMUSE Model as the shaping structure of our approach, as next step towards our complete TrustMUSE system, we defined the TrustMUSE Process as an easily understandable process for interacting with the model. The TrustMUSE Process, as specified in [3], provides the means to systematically browse Trust Management state of the art. It enables developers to think in terms of their application, and in return be provided with a Trust Management framework. The software part of our system is the TrustMUSE Builder, which guides our target users through the TrustMUSE system and automates its processes.

With having a first integrated prototype of the whole TrustMUSE system available, we executed a first evaluation where target end users not just provided feedback about the intermediate concepts, but were asked to solve a specific application design task with our integrated approach. All of our test users were able to find an appropriate Trust Management framework based on TrustMUSE, and also understood what had been suggested to them. Even if this was only a first evaluation, its results were significant as they provided a first valid indication of whether TrustMUSE was able to solve the previously mentioned challenges.

The remaining paper is structured as follows: section 2 contains state of the art in the areas of modelling and usable security; section 3 describes the TrustMUSE Model and presents its APIs; section 4 deals with the TrustMUSE Process and TrustMUSE Builder implementation, which is then evaluated in section 5. Section 6 concludes the paper and outlines future work.

2 State of the Art

There is a large number of Trust Management frameworks available - all designed for different specific domains [5–7]. Similarly, there exist multiple surveys collecting and categorizing Trust and Reputation Management frameworks and identifying common aspects [8, 9]. As evident from these examples, Trust Management has thoroughly been researched, and there are many approaches ready to be used for finding trustworthy service providers in distributed environments. A consequence of this vivid research is the recent attempt to standardize Trust Management, and achieve interoperability between individual frameworks. There are multiple aspects where such standardization has started: common taxonomy [10], generic models [11], meta models [4, 12] and identification of common procedures [13]. However, these standardization approaches are still at their start and are barely applied – also because of their lack of formal application method.

Even if standardization of Trust Management were at its full extent, developers of distributed applications, who are less acquainted with security, would still have a hard time in knowing which solutions are the most appropriate for their challenges. On the one hand, this is due to the well-known problem of users having difficulties understanding security concepts [14]; on the other hand, this is caused by the lack of well-defined processes for finding a Trust Management solution applicable to a problem [3]. In order to solve the first problem, the field of user-centered security had risen [15]. It identifies the need to approach users, and support them in securely using software. This is generally achieved by applying an iterative user-centered design methodology [1], as we also did for our own approach. For achieving the desired understanding of security at the developers, Model-Driven Security (MDS) is one potential solution [16]. MDS aims at creating clear and understandable models of applied security procedures: this helps to clearly separate aspects of the software, improving overview and understanding of functionality. Additionally, during development, MDS enables code to be better structured and stay in accordance with specification and documentation.

Individual smaller steps of our approach had been presented previously: we presented the requirements from the initial workshops in [2]; in [4], we presented the conceptual meta model which is further improved into the TrustMUSE Model in this paper; the TrustMUSE Process that enables non-security experts to find an appropriate Trust Management solution for an application has been shown in [3].

3 TrustMUSE Model

The TrustMUSE Model is a meta model with accompanying APIs for Trust Management frameworks. It is based on the original TrustFraMM concept that has been published in [4]. Over the past years, we have continued to work with this model: we implemented specific Trust Management frameworks based on the model's elements, we have defined generic interfaces over which services can be consumed and interchanged, and we defined design patterns for the integration of the system into arbitrary applications. The experience gained through this process has matured the TrustMUSE Model into the version we present in this chapter. First, we provide an overview of the meta model – a detailed description of the elements can be found in [4] – and the accompanying APIs as seen in Fig. 1; after that, we briefly describe two systems we implemented in accordance with the model.

3.1 Description of the Model

There is no proper way to describe Trust Management functionality in a sequential way; therefore our sequential presentation of the TrustMUSE Model should not be considered as a restriction on the operation of describable frameworks. We start our description from *Trust Evidence*: the raw pieces of data – like observations, recommendations and certificates – that guide the system in deciding whom to trust. *Trust Evidences* have assigned *Trust Scopes* – used similarly at other places of the system – to distinguish received data according to aspects or contexts of entities' behavior. *Trust Discovery and Distribution* defines where, how to search for and eventually also share evidences, and it places them into the *Evidence Storage*: a typed database providing querying functionality to the rest of the system. *Trust Evaluation* takes this data, filtered by entities, and passes it to the associated *Trust Model*. This model describes the rules and procedures that turn raw data into assessed *Trust Values*. The output from this step is placed into the *Trust Storage*, which is the counterpart of the *Evidence Storage*, just for interpreted information. This information is then used by *Trust Enforcement* – the act of trusting: deciding if for a given service an entity is to be trusted, selecting a fitting provider for an action, or simply notifying the application about changes in someone's trust. Compared to using pure assessed *Trust Values*, this act includes considering contexts, risks, alternatives and priorities. After an interaction with another entity, it is possible to provide feedback into the system by the means of *Interaction Evaluation*. It places any feedback into the *Evidence Storage* and additionally, depending on its implementation, initiates *Trust Update*: the procedures that keep the status of the system fresh.

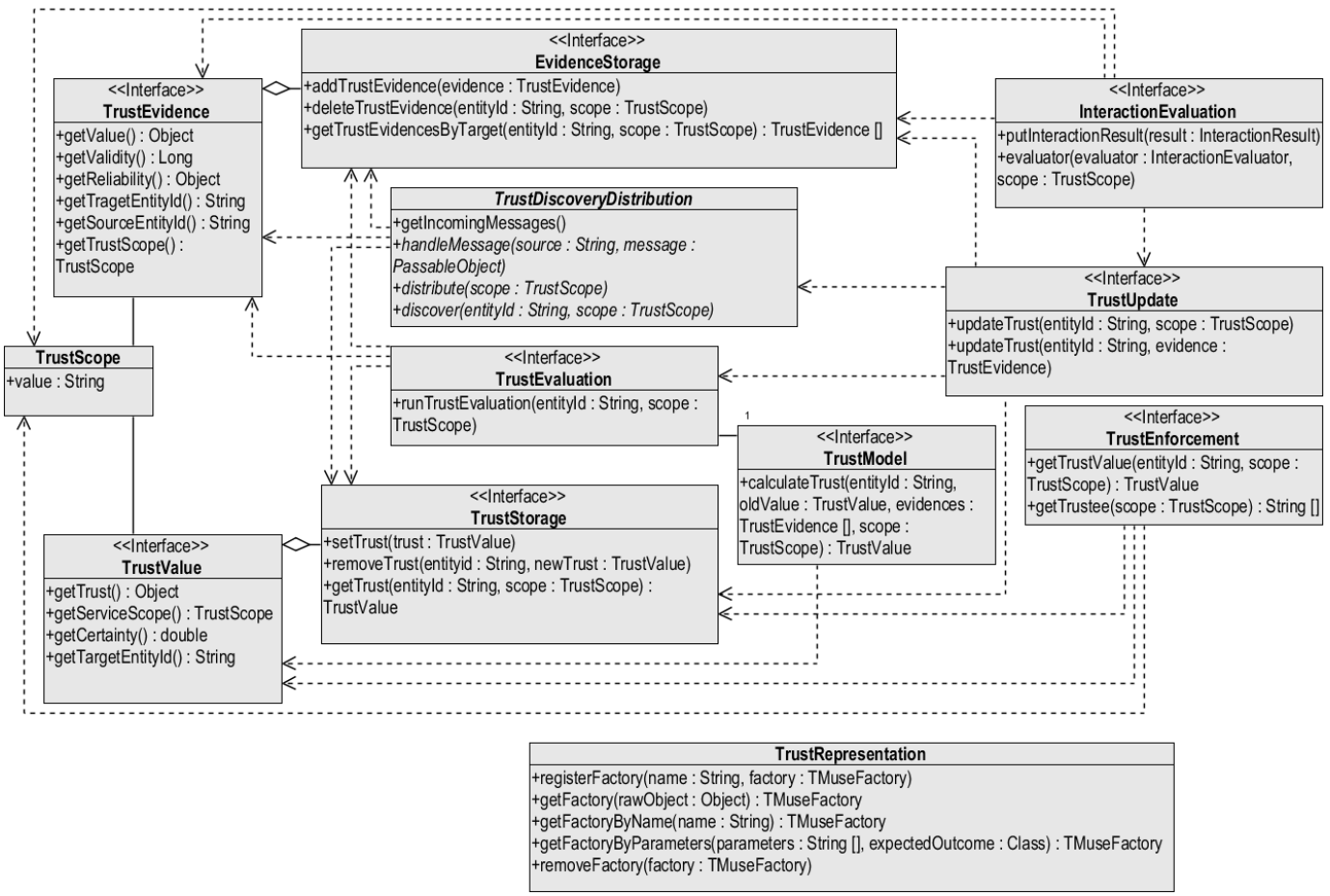

Fig. 1. UML class diagram of the TrustMUSE Model.

The APIs of the elements introduced so far represented the services of the underlying functionality; for this reason we did not discuss them in detail. However, the next element requires more introduction as its behavior is more bound to a specific design pattern: *Trust Representation* is the element holding information related to the formats applied in any of the TrustMUSE Model's elements. We did not indicate this in the class diagram, but almost every component may be dependent on it. It is needed to enable the integration of different procedures with their accompanying formats – as is intended by TrustMUSE. To implement this, *Trust Representation* is a repository for factory objects[1] where every element can register a class that is able to parse its respective formats; these factories are then used by other elements when required. The API of *Trust Representation* reflects this role as it provides methods to find specific factories based on input data, constructor parameters or names.

We implemented the here presented API in Java in form of OSGi[2] Declarative Services; this enforced us to clearly follow the defined dependencies. In the next section, we will see how we used our implementation to integrate Trust Management functionality into two specific applications.

3.2 Developing Software with the Model

In order to validate the APIs of the TrustMUSE Model, we continued elaborating it to see how it can be integrated into applications. We searched for two applications with distinct trust requirements: by this, we aimed to stress that different Trust Management implementations can be modeled and consumed over the same APIs. In this section, we first present the two applications we have selected for this purpose and show what kind of Trust Management functionality we implemented for them; after this, we provide an additional API that is necessary to integrate implementations done in accordance with TrustMUSE into applications.

Reputation Module for Expert Network. This application was an existing product that dealt with connecting human or software experts, providing specific services online, with consumers searching for those services – e.g. a lawyer providing online consultation in multiple fields for possible clients. The owner of this application wished to indicate a reputation score per expert to enable further differentiation of them.

The scenario where the application had been deployed possessed some interesting requirements that needed to be considered while selecting an appropriate Trust Management implementation: reputation scores had to be stored directly at the entity they were about, thus determining how *Trust Discovery and Distribution* had had to be implemented. The *Trust Model* simply took the average of positive and negative ratings; it was executed every time a new rating came in – as defined in *Trust Update.* Reputation scores, as well as the feedback provided about received services, were

[1] http://www.oodesign.com/factory-pattern.html Last visited 22nd January 2014

[2] http://www.osgi.org/Main/HomePage Last visited 22nd January 2014

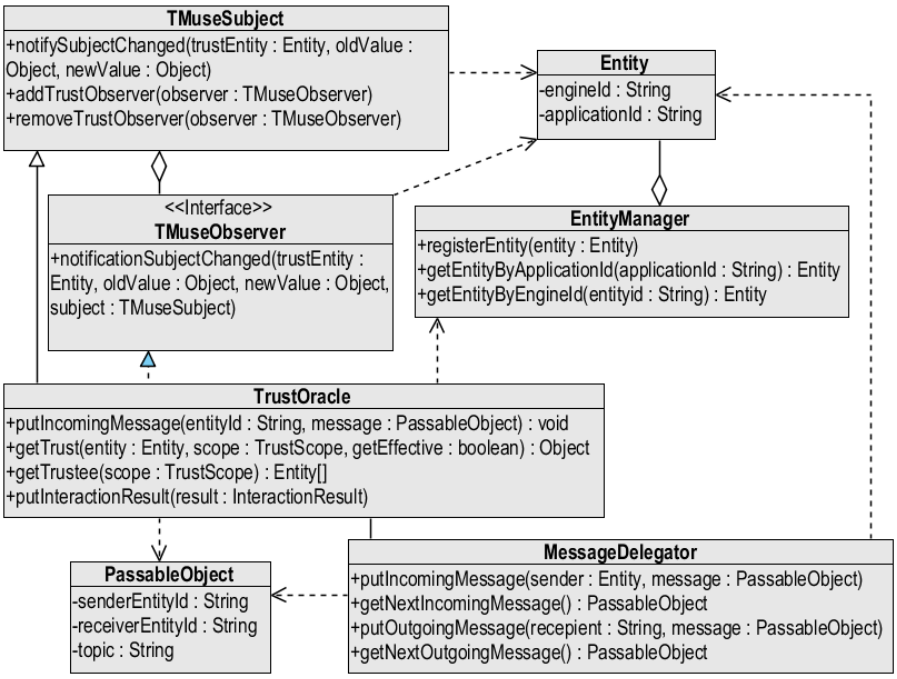

Fig. 2. TrustMUSE integration layer class diagram

presented as a five-star scale – a question of *Trust Representation*. There was no *Trust Enforcement* component, and *Interaction Evaluation* was done by enquiring the user.

Trust Module for Mesh Network. The application had the purpose of monitoring and debugging a mesh network consisting of nodes with different capabilities; it was not intended to actually intervene with the routing. Typically for such applications, our solution was to run on each device, sharing observations across the network. Our Trust Management solution was based on the framework described by [17]; the TrustMUSE division of it was as presented in [4]: *Trust Discovery and Distribution* was implemented to exchange own observations, *Trust Update* initiated this regularly, the *Trust Model* dealt with calculating the parameters of the Beta function, *Trust Enforcement* compared each *Trust Value* to a specified threshold.

Integrating TrustMUSE based solutions into applications. As identified during the requirements process [2], developers would like to mainly build on a basic set of services in their applications – only when required, did they want to customize Trust Management functionality. Therefore we created the TrustMUSE integration layer as seen in Fig. 2: a set of interfaces and utility classes that wrap the Trust Management implementation, and simplify it for the user to consume trust information. The main interface an application works with is the *Trust Oracle*: it consists of the four main

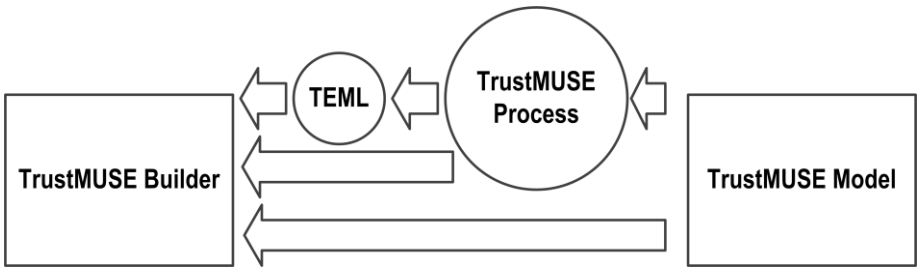

Fig. 3. Overview of the TrustMUSE system

methods that a user expects to receive and that are necessary for the framework to operate.

4 TrustMUSE Process and TrustMUSE Builder

The TrustMUSE Model enables separation of concerns within the Trust Management domain; it helps non-security experts to gain an understanding of its functionalities, and benefit from different implementations. What the TrustMUSE Model cannot provide on its own, however, is the ability to know what implementations work in what environments or for which problems. To provide support in this task, we developed the TrustMUSE Process. In this chapter, we first briefly provide an overview of the TrustMUSE Process and present TEML (TrustMUSE Element Markup Language): the standardized format for describing attributes. The implementation of the TrustMUSE system concepts is the TrustMUSE Builder software, which is described at the end of this chapter. An overview of the components of the TrustMUSE system, and how the TrustMUSE Process and TrustMUSE Builder fit into it, can be seen in Fig. 3.

4.1 TrustMUSE Process

In this section we briefly summarize the TrustMUSE Process, and show how its development fits into the methodology used for the overall TrustMUSE system. The detailed presentation of this can be found in [3].

In the TrustMUSE Process, it is all about attributes: characteristics, conditions or services of Trust Management implementations. For the process, each author providing an implementation, defines the attributes that apply for the developed solution. When executing the process, the user is presented with the set of all author defined attributes, sorted by TrustMUSE Model elements. Developers are now able to focus on one element at a time, select and exclude attributes that seem relevant for the application's scenario – thereby actually selecting between Trust Management implementations that are applicable for it. After having finished the selection of attributes, the TrustMUSE Process suggests those implementations that have the largest overlap with the attributes selected by the user.

Table 1. Example attributes for the TrustMUSE Process

TrustMUSE Model element	Example standard attributes
Trust Discovery and Distribution	Trusted third party; Personalized view on entities; Requires continuous Internet connectivity.
Trust Model	Trusted third party; Uncertainty handling; Continuous forgetting.
Trust Enforcement	Weighted aggregation.
Interaction Evaluation	Rule based feedback; multi-level feedback.
Trust Update	Calendar based; Number of interactions based.

The benefit in the TrustMUSE Process is that developers receive a Trust Management solution without having read the whole state of the art in the field. Even if the first suggestion does not fit completely, developers can quickly change the attributes to receive an alternative candidate; this still significantly reduces the number of frameworks necessary to be read, before finding a fitting one. An example of possible attributes for different TrustMUSE Model elements can be seen in Table 1.

The definition of the TrustMUSE Process had been based on requirements we collected beforehand in focus group workshops. From the requirements, we developed two paper prototypes and compared them during multiple user interviews using a specific development scenario, and manually simulating the process's operation with multiple sheets of prepared paper templates. The feedback collected from these user interviews than finalized the process, and guided the implementation of the TrustMUSE Builder.

4.2 Standardization and Tooling

To achieve the model-driven process we aim for, a certain level of standardization of Trust Management has been necessary; this includes the APIs we have seen but it also includes formats and tools that are presented in this section. As described in the previous section, to enable the TrustMUSE Process, Trust Management experts have to provide attributes for their developed solutions; additionally, these attributes have to be in the same format for all solutions to be able to integrate them into one tool. To enable this uniform representation of attributes we defined the TrustMUSE Element Markup Language (TEML), based on which attributes and their dependencies can be provided, sorted by TrustMUSE Model elements, in machine readable XML format.

A TEML document is composed as follows: the author starts with defining a *trustMUSEElement* with a freely chosen *name* attribute and the *className* of the TrustMUSE Model element the solution is for. As a next step, the characterizing attributes are given to the solution through a number of *attribute* XML elements. In case the solution has no collisions with other elements, the TEML document is finished; else these collisions also have to be defined in the same document. To do so, it is necessary to define further *trustMUSEElements*: these will have no *name* but only a *className* attribute. The attributes, which are placed into these latter *trustMUSEEl-*

```
<?xml version="1.0">
<!DOCTYPE TrustMUSE [
<!ELEMENT TrustMUSE (trustMUSEElement+)>
<!ELEMENT trustMUSEElement (attribute+)>
<!ELEMENT attribute EMPTY>

<!ATTLIST trustMUSEElement name CDATA #IMPLIED>
<!ATTLIST trustMUSEElement className (Scope | TrustEvi-
dence | TrustValue | EvidenceStorage | TrustStorage |
TrustDiscoveryDistribution | TrustUpdate | TrustEvalua-
tion | TrustEnforcement | InteractionEvaluation |
TrustRepresentation) #REQUIRED>
<!ATTLIST trustMUSEElement dependencies IDREFS #IMPLIED>
<!ATTLIST attribute name CDATA #REQUIRED>
]>
```

Fig. 4. DTD of TEML

ements, define the collisions of the solution. Finally, the identifiers of the colliding *trustMUSEElements* have to be provided as dependencies to the original solution's XML element. The DTD of TEML can be seen in Fig. 4.

In order to facilitate the cumbersome process of creating correct XML documents by hand, we developed a GMF[3] based utility in which TEML documents can be generated automatically: the author simply pulls the respective TrustMUSE Model elements onto the canvas, and types attributes into them. Collisions can also be defined by simply connecting two elements. When saved, the utility generates two files: one file containing the diagram layout and one containing the TEML document.

4.3 TrustMUSE Builder Prototype

With the presented concepts at hand, we are able to create the tool that automates the execution of the TrustMUSE Process: the TrustMUSE Builder, implemented as a standalone .NET WPF[4] desktop application. At start-up, it reads its working directory for stored TEML documents and additional informative text files – short descriptions, lists of references and implementation libraries. Parsing these files, the application builds up its model of possible TrustMUSE Model element implementations, their attributes and dependencies; then it prepares and shows the GUI as it has been specified in [3].

The first view of the GUI is called the composition state: this is the state where the user can select and exclude attributes by clicking the checkboxes next to the attributes. After each selection, the application logic checks which candidate implementa-

[3] http://www.eclipse.org/modeling/gmp/ Retrieved 27th January 2014

[4] http://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx Retrieved 27th January 2014

tions are to be excluded, based on the defined collisions, and disables them. For each TrustMUSE Model element, the user can view a short help description; also for each attribute there is a tool tip providing an explanation of the presented term. A screenshot of this view can be seen in Fig. 5. If the user feels happy with the selection and has nothing more to add, she can go to the next view – called the composed state.

The composed state has the same layout as the previous one; the difference is that instead of showing attributes, it shows specific implementations for the elements. An implementation suggestion consists of a name, as specified in the TEML, a short description, and a list of additional references. The user can review whether the suggested framework looks sane for the application's purpose, and decide whether to go back to the composing state and change the attributes, or to acknowledge the framework.

If the suggested framework is acknowledged, which is done by clicking a button in the GUI, TrustMUSE Builder copies the implementation libraries from the respective elements into one specified folder. From this point on, the development process for the user, building on OSGi, looks as follows: as first step, the folder with the implementation libraries has to be set as target platform; second, missing code has to be filled in – like the TMuseFactory for the used trust representation, the network and communication handling, and the consumption of the API's integration layer; finally, the configurations for the different implementations have to be set. If all this is done, the Trust Management framework will be ready to be used.

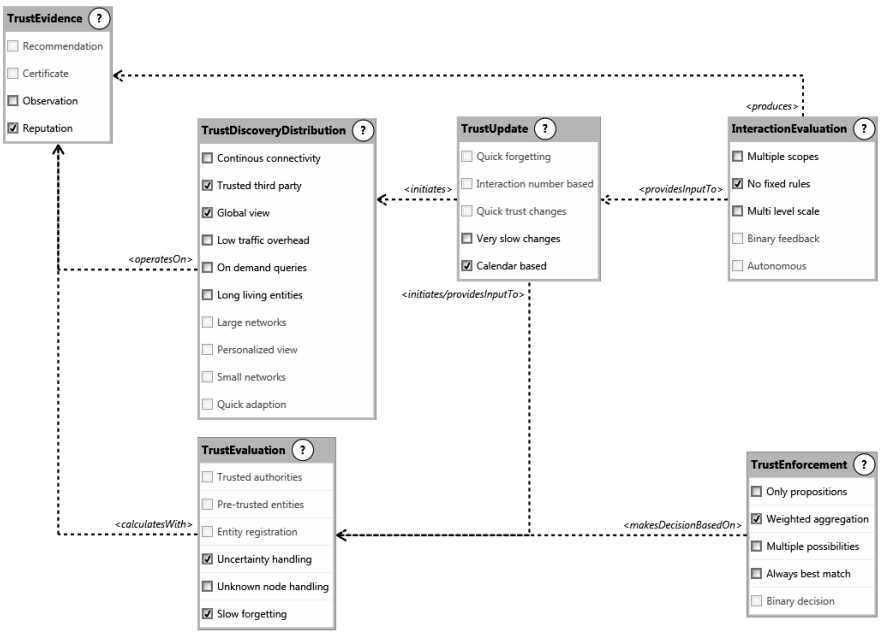

Fig. 5. Screenshot of the composition state of TrustMUSE Builder

5 First Qualitative User Evaluation and Threads to Validity

We evaluated the implementation of the integrated prototypic TrustMUSE system with application developers in a final set of interviews. Our aim was to find out whether our solution is able to support our target users finding a Trust Management solution when given a specific problem: that is, can users based on our system do the transition from specification to solution. Although our system was still in a very early stage of implementation, the evaluation held great significance as it had the potential of providing first valid feedback about the usefulness of the TrustMUSE approach. In this section we present our interview set up and present the collected responses; at the end of the section we interpret the results, and identify lessons learnt and future work.

5.1 Experiment

The main question of the final evaluation was, whether our target users, non-security expert developers, are able find an appropriate Trust Management framework with the support of TrustMUSE. Accordingly, for the evaluation, we first developed a scenario and produced an application specification with very clear Trust Management requirements: our scenario described a company that signed half year contracts with other companies to provide access to its distributed services. To ensure that users did not try to solve the task with regular requests to a server, we included a clause, stating that the company's server was not to be contacted too often, into the specification – out of scalability reasons.

With this scenario, we approached five users: each of them had multiple years of software design experience but little to no security qualification. At the beginning of the test, we presented them with the specification and asked them to draft a solution based on their knowledge, without any support. Following this, we presented them the TrustMUSE Builder software: we provided a brief description of the TrustMUSE Process and explained that the presented software is not fully implemented; therefore the participants were allowed to ask questions during the experiment, however, we only answered if the misunderstanding was caused by the prototypic nature of the tool. For a better understanding of the developers' mindset, we asked our participants to think aloud, explain their decisions and state any ambiguities they encounter. Also, after they received the framework suggestion from the tool, they were asked to describe how they interpret the proposed solutions, and whether they think it is appropriate for their original problem. In the experiment, they were only asked to go as far as to click the framework generation button; we did not intend to evaluate the code integration aspects of the TrustMUSE system at that moment. Finally, we did a structured interview consisting of twelve statements with five-point Likert scale – strongly disagree, disagree, undecided, agree, strongly agree – responses, categorized into three topics: how clear were the components and dependencies of the model, how well was the process able to help them, how much did they benefit from the tool. Additionally, we also collected some open feedback for future extensions of the tool.

Table 2. Results of the user evaluation

Statement	Mean answer	P-Value
You understand the concept of Trust Management.	Agree	0.0002
You understand the sub-processes present in Trust Management.	Undecided	0.0332
You understand the connection between the Trust Management components.	Agree	0.0001
You think the concept of attributes is a good way to describe your scenario.	Undecided	0.0004
You understand the Trust Management solution that has been proposed.	Agree	0.0002
You think the proposed framework is appropriate for your problem.	Agree	0.0001
You could explain the proposed framework to someone else.	Agree	0.0002

5.2 Results

The participants, when asked to provide a solution for the specification based on their own knowledge, all foresaw a certificate based system – except for one user who could not come up with a solution; however, they were not able to provide any more details regarding the realization of their idea. Subsequently, with the support of TrustMUSE, each user was able to find an appropriate solution for the problem, and understood how the proposed framework solved the specification. Additionally, our users stated to have gained a deeper understanding of Trust Management, and to have understood the relations defined in the TrustMUSE Model.

While the above statements remain valid for the overall experiment, there were statements in the interviews where the responses were much diffused and often had outliers. To be able to make sense and exclude invalid results, we applied statistical evaluation methods to the response sets: we checked the interquartile ranges (IQR) and made statistical hypothesis tests. First, we excluded all statements where the responses' IQR was more than 2; this step excluded four statements out of the twelve. Afterwards, for the remaining statements, we made the null hypothesis that our participants disagreed with them; testing our data against this null hypothesis excluded one additional statement. The statements remaining valid after our tests, with accompanying p-values, are presented in Table 2.

5.3 Lessons Learnt and Future Work

Reviewing our results from the evaluation, we found that our approach started on a sound track: developers were able to solve the task with the support of TrustMUSE

that they could not properly handle before. Additionally, a very important benefit of TrustMUSE, as stated by the test participants, was the gained information about elemental components within Trust Management, their relations, and their applicability. However, TrustMUSE still showed to be too technical and users had difficulties dealing with all the new terminology; also, clarity and usability of attributes' terminology in the TrustMUSE Process caused misunderstandings.

The difficulty in defining attributes is that they have to be detailed enough to describe fine operational details of implementations; however, they should not be too technical, so that our target users still understand them. Additionally, as we have learnt from our user tests, attributes have to be unambiguous: even if we provide explanations, users tend to interpret terms to accommodate their own beliefs. Therefore, it will be necessary to execute a separate user-centered design process, where the appropriate set of attributes shall be found through multiple iterations. The overall process shall consist of analyzing multiple frameworks, dividing them by TrustMUSE Model elements, attributing them, and then talking to developers about their understanding.

Our users expressed some additional wishes towards the TrustMUSE Builder tool: they wished to see what effects their decisions had on the final framework suggestion. They sometimes felt lost during the use of the tool, and did not know whether what they did made sense; they could not clearly see the relation between their input and the application's output. Therefore, future developments should address these issues: find visual features that could tackle the lack of transparency, provide better indication of what the tool is doing currently, and generally better involve the user into the decision process.

6 Conclusions

In this paper we presented TrustMUSE (Trust Management Usable Software suitE): a model-driven approach for integrating Trust Management into applications. Building on the experience from the state of the art in user-centered security and model-driven security, our approach aims at supporting non-security expert application designers to find appropriate Trust Management frameworks for their application domains. We first presented the TrustMUSE Model: a meta model for Trust Management with accompanying APIs. Based on OSGi based implementation experience, we also provided an API for an integration layer that wraps Trust Management functionality, and only exposes main services that are needed by the relying distributed application.

Built on top of the TrustMUSE Model, we presented the concept of the TrustMUSE Process and its implementation: the TrustMUSE Builder. This tool first reads different Trust Management implementations that are described using TEML (TrustMUSE Element Markup Language) documents; these implementations are then presented in an abstract format to the users of the tool. They can then describe their application specification by means of attributes, and subsequently receive a Trust Management framework suggestion tailored to their needs.

We closed this paper with the evaluation of our system, where users were asked to solve a Trust Management task based on the TrustMUSE Builder software. Each user was able, within a limited time span, to come up with an appropriate solution; additionally, they felt confident in the validness of the proposed framework for their problem. Based on the collected user feedback after the experiment, we conclude, that we should further increase the abstraction level of the representations used in TrustMUSE. Future work needs to address the design of a more straight forward process that better supports our target end users in incorporating Trust Management into their application designs.

Acknowledgements. This work has been performed within the ALMANAC project, co-funded by the EC within the FP7, theme ICT-2014.1.4 Future Internet and smart Internet of Things, grant agreement No. 609081. Possible inaccuracies of information are under the responsibility of the project. This report reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained therein.

References

1. Gould, J.D., Lewis, C.: Designing for usability: key principles and what designers think. Commun. ACM. 28, 300–311 (1985).
2. Vinkovits, M.: Towards requirements for trust management. Privacy, Security and Trust (PST) 2012. pp. 159–160. IEEE Comput. Soc, Paris, France (2012).
3. Vinkovits, M., Zimmermann, A.: Defining a Trust Framework Design Process. Trust, Privacy, and Security in Digital Business. pp. 37–47. Springer-Verlag, Prague, Czech Republic (2013).
4. Vinkovits, M., Zimmermann, A.: TrustFraMM: Meta Description for Trust Frameworks. ASE/IEEE International Conference on Privacy, Security, Risk and Trust. pp. 772–778. , Amsterdam, Netherlands (2012).
5. Marti, S., Garcia-Molina, H.: Limited reputation sharing in P2P systems. Proceedings of the 5th ACM conference on Electronic commerce - EC '04. pp. 91 – 101. ACM Press, New York, New York, USA (2004).
6. Zouridaki, C., Mark, B.L., Hejmo, M.: Byzantine robust trust establishment for mobile ad hoc networks. Telecommun. Syst. 35, 189–206 (2007).
7. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. Proceedings 1996 IEEE Symposium on Security and Privacy. pp. 164–173. IEEE Comput. Soc. Press (1996).
8. Josang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. 43, 618–644 (2007).
9. Artz, D., Gil, Y.: A survey of trust in computer science and the Semantic Web. Web Semant. Sci. Serv. Agents World Wide Web. 5, 58–71 (2007).
10. Viljanen, L.: Towards an ontology of trust. Trust, Privacy, and Security in Digital Business. pp. 175–184. Springer-Verlag, Copenhagen, Denmark (2005).
11. Kinateder, M., Baschny, E., Rothermel, K.: Towards a Generic Trust Model – Comparison of Various Trust Update Algorithms. Proceedings of the Third international conference on Trust Management. pp. 177–192 (2005).

12. Saadi, R., Rahaman, M.A., Issarny, V., Toninelli, A.: Composing Trust Models towards Interoperable Trust Management. Trust Management V. pp. 51–66. Springer Boston, Copenhagen, Denmark (2011).
13. Gómez Mármol, F., Martínez Pérez, G.: Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems. Comput. Stand. Interfaces. 32, 185–196 (2010).
14. Whitten, A., Tygar, J.D.: Why Johnny Can ' t Encrypt: A Usability Evaluation of PGP 5 .0. Proceedings of the 8th USENIX Security Symposium (1999).
15. Zurko, M.E., Simon, R.T.: User-centered security. Proceedings of the 1996 workshop on New security paradigms - NSPW '96. pp. 27–33. ACM Press, New York, New York, USA (1996).
16. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. 15, 39–91 (2006).
17. Buchegger, S., Le Boudec, J.-Y.: A Robust Reputation System for P2P and Mobile Ad-hoc Networks. Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems (2004).
18. IoT-A FP7 Project: Final Architectural Reference Model for the IoT. (2013), http://www.iot-a.eu/public/public-documents/d1.5/view. Last visited 27th January 2014
19. CISCO: Cisco 2014 Annual Security Report (2014), http://www.cisco.com/web/offers/lp/2014-annual-security-report/index.html. Last visited 6th February 2014