Towards Security-Enhanced and Privacy-Preserving Mashup Compositions

Heidelinde Hobel, Johannes Heurix, Amin Anjomshoaa, Edgar Weippl

Towards Security-enhanced and Privacy-preserving Mashup Compositions

Heidelinde Hobel[1], Johannes Heurix[2], Amin Anjomshoaa[2], and Edgar Weippl[1]

[1] SBA Research, Vienna, Austria
hhobel@sba-research.org, eweippl@sba-research.org
http://www.sba-research.org
[2] Institute of Software Technology and Interactive Systems,
Vienna University of Technology, Austria
heurix@ifs.tuwien.ac.at, anjomshoaa@ifs.tuwien.ac.at
http://www.tuwien.ac.at

Abstract. In recent years, there has been an emerging trend towards people building their own sophisticated applications to automate their daily tasks without specialized programming knowledge. Enterprise mashups facilitate end users' development of applications in a business context autonomously or with minimal support from the software engineering staff. Hence, mashup solutions are aimed at exploiting the full potential of end users' software development. However, the use of mashup solutions for business tasks gives rise to several security and privacy-related questions, since sensitive data records could be created even with simple procedures. In this paper, we propose an approach where security rules for mashup compositions can be defined, and submitted mashups are automatically evaluated for compliance with the respective policies.

Keywords: Enterprise Mashups, Semantics, Security, Privacy, Usability

1 Introduction

Web 2.0 comprises a set of new technologies as well as behavior models of end users [1]. One of these technologies is known as *mashup*, which is also becoming popular in an enterprise context as *Enterprise Mashups*. According to [2], Enterprise Mashups are defined as

> "... a Web-based resource that combines existing resources, be it content, data or application functionality, from more than one resource by empowering the end users to create and adapt individual information centric and situational applications."

The basic idea is that existing resources, such as data and services, are used to create a new resource in such a way that even users with limited programming skills are able to fulfill this task. The mashup development is based on mashup editors such as JackBe Mashup [3] or IBM Mashup Center [4] by providing

an integrated development environment (IDE), where programming is accomplished by simple drag-and-drop operations of predefined modules (basically the mashup's operations) and connecting them. Google's Blockly [5] has similar objectives, providing a graphical programming editor that allows programming by puzzling blocks together.

Hence, the application of mashups facilitates "*short-time, situational, ad-hoc, tactical, and individual*" [6] software development and has great potential for various application fields, especially for businesses [7]. However, by shifting the responsibility of application development into end users' hands, several security and privacy-related questions arise, as the end user is given the opportunity to access the enterprise's data and process that data without regulations. In traditional software development, the security of applications is guaranteed by the skill of the software developers, sophisticated test-mechanisms, and reviews. However, these measures are not applicable for mashup solutions, as the expense is usually not considered justifiable due to the short-lived and individual nature of mashups.

In this paper we propose a platform-based approach for establishing rules for mashup design in order to prevent data leakage and distribution of data to unauthorized people and mitigate semantic aggregation, thus empowering enterprises to regain authority over end users' mashup development. This is required since in the last instance, the enterprise is responsible for its own security and for the privacy of owners of the records stored in its databases.

The contributions of this paper are as follows:

- We introduced a platform-based security architecture for designing and enforcing policies for composing mashups in an organizational context.
- We formulated the modeling of mashup compositions in our own notation.
- We implemented a prototype as proof of concept and provided a case study in order to illustrate the usability of the system.

The paper is structured as follows: In Section 2 we provide an overview of the potential of mashup solutions in a business context and the related security and privacy issues, followed by Section 3, where we introduce our platform-based approach for security-enhanced and privacy-preserving mashup compositions. Section 4 evaluates the prototype implementation and a case study, followed by the discussion (Section 5), related work (Section 6), and conclusion (Section 7).

2 Background

According to [6, 8], mashup solutions are the answer to the common problem that only 20% of the required software solutions can be satisfied by the software development staff of a company; the remaining 80%, comprising all "*situational, ad-hoc, tactical, and individual software solutions*" [6], are neglected due to insufficient resources. Mashup solutions aim to solve this shortage by involving the end users in software development, supported by SOA, Web Services, and lightweight compositions [8].

The major advantage is that an end user can easily implement a completely suitable application for any task in a short time. However, this also implies that typical software quality and security measures, such as the skill of the developers, test mechanisms, reviews, and audits, cannot be applied to mashups, as they would be very costly for this kind of development where software is developed in a short time and for every task, leading to the emergence of new bottlenecks.

Anjomshoaa et al. [9] summarized the security, privacy and trust issues that arise with the mashup technology and data processing application: "*(1) Resource Trustworthiness, (2) Content and Feed Copyright Issues, (3) Information Leakage, (4) Distribution to unknown or unauthorized users, (5) Distribution of sensitive information, and (6) Creation of sensitive information through aggregation*". *Information leakage* refers to the mashup's nature of facilitating data processing and publishing, and sensitive information that is not allowed to cross organizational borders, and thus organizations must be prepared to ensure that only permitted data is generated by mashup solutions and published publicly. *Distribution of (sensitive) data* to *unknown or unauthorized users* means that internal borders within organizations also have to be considered in data distribution that is facilitated by mashup solutions. Sensitive information refers to data that contains personal information or information about companies that could be used against the company itself, resulting in a security breach and ultimately affecting the company's competitiveness. *Creation of sensitive information through aggregation* is an intrinsic issue of the mashup technology enabling sophisticated data processing possibilities. Without appropriate regulations on how data may be processed, mashup solutions can be used to process huge amounts of data with the result of disclosing valuable personal or organizational information in an unauthorized way or for malicious purposes. [9]

Hence, the application of mashup solutions requires new measures for secure and privacy-preserving data handling in an enterprise context, where we have to concentrate on data processing rather than access rights.

3 A Platform for Security-enhanced and Privacy-preserving Mashup Compositions

In traditional Enterprise/Web Applications, the functionality is provided by the software engineering staff in the form of methods or functions, where a great part of these methods/functions access the enterprise's data and transform the data according to the implemented logic. We have to consider that the implemented logic is developed by skilled developers by considering organizational policies and weaving them into their implementations. With mashups, the task of implementing functionality is shifted from skilled developers to other end users. However, as allowing end-users unrestricted data transformations may compromise the organization's security integrity, we propose the extension of traditional enterprise or Web platforms by an additional vertical layer that is responsible for validating the security and privacy-preserving characteristics of end users'

mashups and server-side execution of accepted mashups to protect the security and privacy of the enterprise's data.

3.1 System Architecture

The system architecture of the proposed approach is based on the fundamental Enterprise/Web Architecture and extends the existing functionality by a module that (i) validates mashups based on a ruleset and (ii) allows the server-side execution of accepted mashups. While making use of matured and well-engineered security mechanisms of traditional platforms, we can concentrate on mashup validation mechanisms that ensure that the mashups' provided functionality is compliant with the enterprise's policies.

As illustrated in Figure 1, the existing platform is extended with the mashup validation and execution module, where the traditional enterprise functions are granted access to the company's data on demand, as these functions are considered trustworthy.

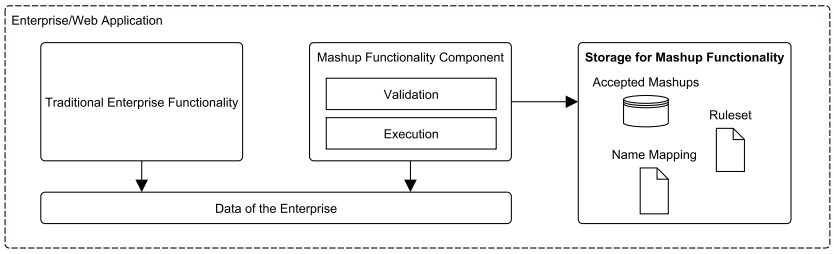


Fig. 1. Proposed architecture of the platform-based approach.

In order for mashup functionality to be executed on the system, the mashup has to pass the validation mechanism of the system, which is based on a ruleset. Additional name mapping is used to map domain-specific names to the naming convention that is used in the ruleset. Accepted mashups are stored in the system, and only these mashup solutions are allowed to be executed and to access data. Other required functionalities are an interface for submitting or designing a mashup solution and an interface for displaying violated rules so that the mashup designer can correct unapproved mashup compositions.

3.2 Mashup Validation

Mashup characteristics are validated by a specific ruleset that defines how the mashups need to be composed. The ruleset incorporates the policies of the enterprise, defining what has to be done, what can be done, and what must not be done. As the names of concepts used depend on the actual domain and there are different mashup languages, a flexible mapping system is required to map

the domain and mashup language-dependent names to the naming convention used in the ruleset.

Ruleset Design A mashup solution is basically a composition of predefined operations, i.e., a set of operations with a specific sequence. In order to validate the mashup's characteristics, we formalize a mashup composition as a directed graph and thus, $M = (O, A)$ where

- M is a mashup solution/graph,
- O is the set of operations/nodes $\{\texttt{op}_1, ..., \texttt{op}_k\}$ that are used in mashup solutions,
- A is the set of directed edges or arrows respectively connecting the operations and determining the successor of a node.

A typical graph that represents a mashup solution is restricted by the following two characteristics:

1. Each operation has at least one predecessor, except the starting operations. Operations with more than one predecessor are table joining/merging operations or other operations like table constructor operations.
2. Each operation has exactly one successor, except the last operation, which constitutes the endpoint.

Furthermore, some operations are customized by parameters that specify what the operations really do, e.g., specifying which data columns are selected by a certain operation. The parameters mostly depend on the fundamental data structure, e.g., the specified operation is dependent on the name of the column, and thus we formalize operations with parameters as: `op[P]`, where P is the set of usable parameters. Concluding, we formalize a specific mashup solution M_i as a graph with the set O notated in the following form

$$M_i := \{\texttt{op[P}_1\texttt{]}_1, \texttt{op[P}_2\texttt{]}_2, \ldots, \texttt{op[P}_n\texttt{]}_n\}$$

and the set A as the sequence of operations. Other context information also has to be considered in the ruleset, such as the role of the user who is the designer and executor of the mashup. The enterprise platform can be used to derive such context information and can therefore be included in the ruleset.

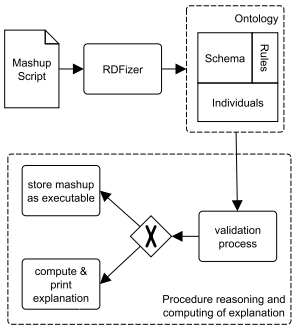
Based on the definitions given above, it can be concluded that the graph of every mashup solution results in a tree structure, more precisely an in-tree, where

- The root node holds the solution, i.e., the endpoint of the mashup.
- The leaf nodes constitute the operations fetching data from the source tables.
- The inner nodes constitute the actual data transforming operations.

The ruleset aims to restrict the relationships between the defined concepts, such as the type and sequence of operations, the relationship between an operation and a specific context, etc. For the implementation of the ruleset checker,

there are several equivalent possibilities such as Object Constraint Language (OCL), rule-based systems, ontologies, etc. In the case of ontologies, we propose the following general relations: (1) Operation `hasParameter` Parameter, and (2) Operation `hasSuccessor` Operation. Obviously, in most cases the role of the user will be important in the execution of the operation, and thus we define the following relation: Executor `performs` Operation. These and other specified relations are restricted afterwards based on the determined effect of the rule. With sophisticated ontologies we can use property, hasValue, and cardinality constraints to achieve the intended effect.

Validation Workflow The actual implementation of the validation process depends on the technology used for the ruleset design. We illustrate the workflow implementation with an ontology-based core system, as shown in Figure 3.2:




```
Data: model <- ontology model with assigned individuals
Result: explanationList <- explanation statements
set USE_TRACING of reasoner on true;
reasoner <- create reasoner instance;
infModel <- create inference model;
infGraph <- create inference graph;
if infGraph is inconsistent then
   explanation <- create explanation from inconsistency;
   forall the statements S of explanation do
      if S is not Axiom then
         add S to explanationList
      end
   end
end
    Procedure reasoning and computation of explanation
```

Fig. 2. Activity diagram and pseudocode of the validation process with an ontology-based core system.

The **mashup script** to be analyzed is parsed for operations, parameters, and the sequence of the operations. This is done by a dedicated **RDFizer** (cf. [10]), which creates individuals from the parsed information and maps them to the **schema** incorporating classes and relations in order to provide machine-understandable meaning for the individuals. The **rules** are implemented by logical restrictions based on the relations of the ontology. The inference model of the ontology is created in the **validation process** and is the result of applying the reasoner to the fundamental ontology. It contains all entailed knowledge about the ontology, e.g., subclass relationships that are inferred from defined classes. In the last step, the reasoner automatically checks whether the inference model of the ontology is consistent. If a predefined rule is violated, the reasoner iterates over the axioms in order to find statements that do not comply with the predefined axioms. However, only the individuals that do not comply with the restrictions and their relations should be printed as explanation for the user, as we consider all axioms to be correct, although certainly, failures in the ontology design could also occur that require manual examination of the rules. Finally,

all statements that concern individuals have to be prepared in human-readable form. If the reasoner identifies no inconsistency, the submitted mashup solution is persisted and can afterwards be executed by an end user with the role and other dependencies that are identical to the submitter of the mashup solution.

We used Pellet [11] as reasoner engine due to its ability to reason in an inconsistent state. The pseudocode for the procedure for creating an inference model, consistency verification, and the computation of the explanation (see Figure 3.2) was derived from a Java implementation with the Pellet [11] reasoner.

3.3 Execution

Some of the rules are dependent on the actual context, and thus a mashup solution may only be executed in the context in which it was validated. For instance, the rules for a mashup are often dependent on the role of the user, and the implemented functionality should not be usable by another user. Obviously, we can use the authentication system of the traditional enterprise platform to specify such rules. Furthermore, the execution of mashup solutions has to take place on the server side due to the following assumptions: (1) The system works as a protection shield between the enterprise's data and the staff, allowing only accepted mashup solutions to access data during the execution. Without a trusted environment, users are able to rewrite accepted mashup solutions and may exploit valuable information as a consequence, and therefore the mashup must not be changed after validation. (2) Even if the mashup solution were to be verified and accepted, the execution of the mashup solution has to take place in a secure environment due to the simplicity of most mashup languages and solutions, which are not specifically designed for secure and trustworthy data transfers.

4 Evaluation

We identified the healthcare sector as a suitable domain for an application example, as it uses highly sensitive data and is a major research field of privacy-preserving data publishing (PPDP) [12]. We evaluated our proposed approach by implementing a prototype, an implementation of a specific ruleset for a fictional healthcare application field, and used this to discuss an application example in the form of a case study where doctors are able to write mashups solutions to analyze data for their research by following the privacy rules of the hospital.

4.1 Prototype Implementation

As proof of concept, we implemented a Web application platform using Java for the business logic and the Apache Wicket Framework for the front-end. We did not use existing functionality methods, considering them as independent from our approach, and concentrated on the implementation of the mashup functionality module. The validation mechanism is implemented based on the proposed workflow for ontology concepts due to their adaptability and characteristics of

semantic solutions, using the Web Ontology Language (OWL) to model the security rules. We limited ourselves in that only mashup scripts written in the Enterprise Mashup Markup Language (EMML) [13] could be validated and, if accepted, executed on the EMML Reference Runtime Engine. The embedded ontology that comprises the ruleset was designed with Protege [14].

The handling of the prototype for the execution of mashups works as follows: A user can access the platform with a browser of her/his choice and upload mashup code using a form. Despite the proposed approach to store only accepted mashups, we stored them as persistent for test purposes so that the user can access her/his uploaded mashup scripts and validate them on demand. If the mashup script adheres to the embedded rules, it is executed and the output of the mashup is displayed, otherwise the violated rule is displayed together with the respective mashup's relations (see Section 3.2, Ruleset Design) that have been computed from the validation system.

4.2 Design of the Ruleset

For the healthcare example, we defined two basic policies for mashup design:

1. A user has to anonymize the transformed data before it is displayed for privacy reasons.
2. A user may only filter the fundamental data according to `Birth`, `ZIP`, or `Sex` of patients.

The set of usable parameters P for the domain comprises the attributes `Name`, `Birth`, `ZIP`, `Sex`, and `Disease`. Furthermore, for our domain we defined the following three usable mashup operations, constituting the operations of the set O for the domain of our case study:

- `Fetch:` This operation fetches a data table. We neglect the resource address for our examples, assuming that only one available data table exists, provided by a Web Service.
- `Filter[`P`]:` A usual filter operation uses an expression such as *Attribute = Value*. However, we simplify the expression by taking only the column into account. As we are using EMML, the mapping system has to decompose the XPath expression.
- `Anonymize:` A Web Service predefined by the development staff anonymizing a data table according to a privacy model.

For evaluation purposes we restricted ourselves to the following mashup formalization:

$$M_i := \{\texttt{op[P}_1\texttt{]}_{(1)} \mapsto \texttt{op[P}_2\texttt{]}_{(2)} \mapsto \ldots \mapsto \texttt{op[P}_n\texttt{]}_{(n)}\}$$

which is possible by exclusing operations like joining/merging tables from consideration, resulting in each operation having only a single predecessor. Therefore, the above informal notation of a mashup solution should be interpreted as an abbreviation of an ordered n-tupel.

We implemented the policies for mashup solutions with the following three restrictions (cf. Description Logic), determining the sequence of operations (cf. Equation (1)-(3)), the attributes that can be used for filtering (cf. Equation (4)), and that in every case an anonymization operation has to be called by the user (cf. Equation (5); in case of ∃ please remember to define a contradiction due to the fact that ontologies work with the Open World Assumption (OWA)):

$$\texttt{Fetch} \sqsubseteq \forall \texttt{hasSuccessor.(Filter or Anonymize)} \quad (1)$$

$$\texttt{Filter} \sqsubseteq \forall \texttt{hasSuccessor.(Filter or Anonymize)} \quad (2)$$

$$\texttt{Anonymize} \sqsubseteq \forall \texttt{hasSuccessor.Nothing} \quad (3)$$

$$\texttt{Filter} \sqsubseteq \forall \texttt{hasParameter.(Birth or ZIP or Sex)} \quad (4)$$

$$\texttt{Executor} \sqsubseteq \exists \texttt{performs.Anonymize} \quad (5)$$

4.3 Validation

Alice is a fictional doctor in the hospital, well-educated in healthcare and interested in research and publication of her results. She is using the hospital's mashup functionality to design personalized mashup solutions for data transformations, facilitating analysis of enormous amounts of data.

In the first case, she just wants to know which effect the date of birth has on a disease and implements the mashup $M_1 := \{\texttt{Fetch} \mapsto \texttt{Filter[Birth]}\}$. In this case, the platform computes an inconsistency due to the missing operation `Anonymization` (cf. Equation (5)) and provides the following notification as failure message along with some type of information about the naming conventions used: `Failure: Executor not performs Anonymization`.

Hence, Alice knows that she has to perform an Anonymization operation. In the next mashup solution, she inserts the operation in the following way: $M_2 := \{\texttt{Fetch} \mapsto \texttt{Anonymization} \mapsto \texttt{Filter[Birth]}\}$, whereupon the platform rejects the submitted mashup solution according to Equation (3) with the following notification: `Failure: Anonymization hasSuccessor Filter`.
Finally, she changes the implementation of the mashup to the following: $M_3 := \{\texttt{Fetch} \mapsto \texttt{Filter[Birth]} \mapsto \texttt{Anonymization}\}$, and the results of the implemented logic are displayed on the screen. Furthermore, as Alice is unfamiliar with the security policies of the enterprise concerning data transformations, she tries to implement a mashup that filters for a specific name: $M_4 := \{\texttt{Fetch} \mapsto \texttt{Filter[Name]} \mapsto \texttt{Anonymization}\}$. However, as it is not her task to establish a link between a patient's name and disease, which might circumvent the security mechanism of the anonymization operation, the system forbids the mashup according to Equation (4) and displays the following notification: `Failure: Filter hasParameter Name`.

A possible and useful enhancement of the restrictions shown here would be the application of another ruleset that controls data published to another research center, so that Alice cannot release datasets that are not approved by the authority.

Due to paper length limitations, we cannot provide a more in-depth use case example, but we believe the basic approach should be clear from our explanations. We also defined more complex use cases, using the full graph of our mashup notation and several operation types as well as parameters. Furthermore, we implemented our system for more than one application domain and used different reasoning engines for evaluation. Due to Pellet's ability to reason in an inconsistent state, we used it for the main part of evaluation. Additionally, we used our formal notation to define objects, enabling us to simulate mashup scripts as test objects and use automated tests to validate their compliance with the defined policies. With automated tests, we were able to test permutations of defined operations, parameters, and sequences, and we analyzed the results of the reasoning system automatically as well as manually in a single review process.

The drawback of our solution is that the formulation and restriction of mashup compositions can be a time-consuming task and it is hard to formulate proper rules for security and privacy purposes in advance. We believe that the best approach is to begin with a small domain. For instance, platforms such as Web-based time-management platforms, where staff members have to enter their working hours, could be enhanced if different users could design their own mashup solutions for personalized statistical evaluations. Of course, trying to model the entire data structure of an enterprise and using mashup solutions for each use case would be a daunting task. The data-processing patterns should be kept as simple as possible. For instance, data aggregations have to take place at the beginning and then it will be stated which operations have to be executed so that the data is cleaned up afterwards before further tasks can be performed on the aggregated set of data.

In our evaluation we neglected performance aspects of reasoning since it is closely related to the reasoner used. Furthermore, we limited ourselves to the in-tree definition of mashup solutions and left mashup solutions with cyclic architecture out of consideration.

5 Discussion

We illustrated our approach on the example of an ontology-based core implementation that is based on pure logic and deductive reasoning and therefore fully comprehensible by machines as well as humans. The proposed platform is designed to return the authority over data-processing to the enterprise, so that the enterprise is able to regulate how and under which circumstances data is accessed and processed by specifying patterns that are modeled in the embedded ruleset. In this section, we analyze our approach at multiple levels.

5.1 Advantages of the Mashup Formalization

As we consider the mashup as an in-tree, where we have sequential $\{\mathsf{op}_i\} \mapsto \{\mathsf{op}_j\}$ relations and joining/merging $\{\mathsf{op}_1, \ldots, \mathsf{op}_n\} \mapsto \{\mathsf{op}_r\}$ relations, we can divide

the whole tree into sub-problems and concentrate on analyzing the single operations, sequences of operations, and joining/merging problems. Testing single operations means validating their parameters and context $\{\texttt{context}\} \mapsto \{\texttt{op[P]}_\ell\}$, testing sequences of operations means validating against malicious aggregation (e.g., SUM, MAX, etc.) $\{\texttt{op}_i\} \mapsto \{\texttt{op}_j\}$, and testing joining/merging operations means validating that culminating previous results does not constitute a possible threat $\{\texttt{op}_1, \ldots, \texttt{op}_n\} \mapsto \{\texttt{op}_r\}$. Furthermore, we can categorize single operations as well as sequences of operations, facilitating that each category can be assigned a specific threat level. The explained sub-problem characteristics together with the usage of categories facilitate a flexible ruleset that allows the end user to program individual and personalized software without the administrator having to adapt the ruleset for each use case.

5.2 Security and Privacy Issues

In the following, we will discuss the effects of our system on security and privacy issues that are introduced in Section 2.

Information leakage and distribution of data/sensitive information to unknown or unauthorized users: We extended a traditional enterprise platform with our security architecture for mashup functionality. As we verify mashup functionality based on a ruleset where the role of the user can be modeled as well, we can protect the enterprise's data from arbitrary access and transformation. Thus, we can define that only authorized people may execute mashup scripts, which must be compliant with the enterprise's data transformation policies. However, as several browser-related attacks exist and the platform implementation acts as trusted environment, the platform has to be secured by appropriate security measures, which go beyond the scope of this work. Furthermore, there is no way to ensure that staff members who have access to the platform do not forward information to unauthorized people.

An important advantage of the proposed security architecture is the adaptive system structure, providing only additional security measures that are independent from other measures, such as regulations for database security. Especially in the case of mashups, it is important that we distinguish between access rights of data and transformation rights, as mashups are aimed to freely access data and process them according to the needs of the user, and we therefore have to concentrate on transformation rules in mashup security. We believe that access rights are not within the scope of mashup security and should be covered by other well-known security measures (see [15]).

Creation of sensitive information through aggregation: Establishing rules that forbid the creation of sensitive information is actually possible, but in reality it is hard to cover all possibilities of sensitive aggregation procedures. However, the enterprise has the option of enforcing data processing patterns, thereby mitigating the threat of sensitive aggregation. An example is our case study in Section 5, where we illustrated rules that permit only a selection of attributes that are allowed to customize a filter operation, thus determining the permitted aggregation methods.

Extracting information by inference attacks: One attack often neglected in security evaluation lies in the extraction of sensitive data by inferencing several well-anonymized data sets. In the case of mashups this could be achieved by generating suitable mashups, where each strictly adheres to the defined rules regarding privacy protection, but the resulting data sets may be linked by unprotected data columns. Since this is an aspect of the anonymization engine in use and is completely independent from the solution proposed in this paper, depending on the anonymization method in use, be it k-anonymity, differential privacy, or other privacy models [12], the problem must be solved there. One solution could be to log what data has been accessed by a single user through mashups and prohibiting additional mashups if the combination with old mashups would be sensitive regarding inference. Still, this may reduce the value of the overall mashup solution drastically.

5.3 Scalability & Performance

The proposed approach can be used on new systems as well as systems that are already in place. Only the ruleset and mapping implementations have to be adapted for the actual domain. The proposed mapping and ruleset-based design that builds the fundamental vocabulary as well as the basis for the definition of the composition restrictions is going to grow rapidly if it is used in large companies. In order to keep track of the dependencies and restrictions, the ruleset can be divided into fine granular classes so that a separate file is loaded for each context that only includes the mashup restrictions for the respective context. Additionally, we have to consider that for security and privacy reasons, the mashups have to be executed on the server side. However, a layered architecture of the proposed platform allows the use of a redundant server structure, and thus it is possible to distribute the workload on several machines.

6 Related Work

To the best of our knowledge, there are only a few publications on security aspects of mashups due to the novelty of the mashup technology. Below, we discuss the works most related to our topic and approach.

Enterprise mashups have great potential for creating value, but the following papers, among others, motivated us to invest time and effort in our proposed approach. In [9], the authors discuss the security, trust, and privacy problems that come with the mashup's architecture and classify the security threats (cf. Section 2). In [7], the authors explain the shift from the purely casual sector to business-supporting applications. In [16], the enterprise mashup technology is introduced in the business domain for improving individual work processes and as the answer to the ever-changing requirements. In [17], the authors give a market overview of different mashup tools and state that although non-commercial tools provide some predefined security solutions, there are still unfulfilled requirements.

The following papers are related to our work in that they propose security-enhancing and privacy-preserving solutions for composition-based application development. In [18], the authors discuss accountability for mashup services and propose a framework to facilitate trust and the resolution of legal requirements. Their proposed framework has an ontology-based approach. The paper proposes models that are meant for information systems developers to understand the entities in mashup service solutions. In [19], the authors propose a privacy-preserving approach for mashup data Web services by using ontologies, metadata, and ontology queries. Their approach is based on rewriting mashup queries to fulfill privacy constraints and modify them for available data Web services. Following these steps, the composition of the mashup in question is computed and, in contrast to our proposed approach, meant for automatically suggested mashup patterns. Instead of computing patterns, our approach is meant for compliance checking so that the enterprise retains the authority over mashup development but allows end users to design their own solutions. In [20], the authors discuss a composability pattern for general service or modular software development that is based on Language-integrated Query (LINQ). The authors explain how specific operations are divided into higher-ordered classes, so that developing is limited to merely chaining together those operations, and building complex applications is accomplished by forming trees of operations.

7 Conclusion

Mashup solutions offer great potential for end users' software development; however, due to their nature, they give rise to several security and privacy-related issues. The security measures on which we rely in traditional software development are insufficient for mashup solutions, which is why we have presented a novel approach in this paper that empowers enterprises to assume authority over end users' mashup development.

We designed and implemented a security architecture where mashup design policies can be defined and enforced. The proposed platform-based approach is designed to be flexible in such a way that only the ruleset and the domain mapping have to be adapted to the actual application system. We used an example to illustrate how security-enhancing and privacy-preserving policies can be modeled and discussed the security-enhancing effects of the proposed security architecture for mashups in an enterprise context, as well as threats suggested in literature that are a consequence of the nature of mashups.

Future work could possibly deal with providing a top level domain for the patterns, thus allowing the users to reuse and extend their customized domain conventions and requirements.

Acknowledgements. The research was funded by COMET K1 and FEMtech 836740, FFG - Austrian Research Promotion Agency.

References

1. Murugesan, S.: Understanding Web 2.0. IT Professional **9**(4) (2007) 34–41
2. Hoyer, V., Stanoevska-Slabeva, K.: Service-Oriented Computing — ICSOC 2008 Workshops. Springer-Verlag, Berlin, Heidelberg (2009) 148–154
3. JackBe - Presto Mashup Composers, http://www.jackbe.com/products/composers.php
4. IBM Mashup Center, http://www-01.ibm.com/software/info/mashup-center
5. Google Blockly - A visual programming editor, http://code.google.com/p/blockly/
6. Hoyer, V., Stanoesvka-Slabeva, K., Janner, T., Schroth, C.: Enterprise mashups: Design principles towards the long tail of user needs. In: Services Computing, 2008. SCC '08. IEEE International Conference on. Volume 2. (2008) 601–602
7. Anjomshoaa, A., Bader, G., Tjoa, A.M.: Exploiting Mashup Architecture in Business Use Cases. In: 2009 International Conference on Network-Based Information Systems, IEEE (2009) xx–xxvii
8. Ogrinz, M.: Mashup Patterns: Designs and Examples for the Modern Enterprise. 1 edn. Addison-Wesley Professional (2009)
9. Bader, G., Anjomshoaa, A., Tjoa, A.M.: Privacy Aspects of Mashup Architecture. In: Proceedings of the 2010 IEEE Second International Conference on Social Computing, IEEE Computer Society (2010) 1141–1146
10. RDFizers, http://simile.mit.edu/wiki/RDFizers
11. Pellet: OWL 2 Reasoner for Java, http://clarkparsia.com/pellet/
12. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Comput. Surv. (June 2010) 14:1–14:53
13. Open Mashup Alliance (OMA) - EMML Documentation, http://www.openmashup.org
14. Protege, http://protege.stanford.edu/
15. Bertino, E., Sandhu, R.: Database security - concepts, approaches, and challenges. Dependable and Secure Computing, IEEE Transactions on **2**(1) (2005) 2–19
16. Pahlke, I., Beck, R., Wolf, M.: Enterprise Mashup Systems as Platform for Situational Applications. Business Information Systems Engineering (2010) 305–315
17. Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools. In: Proceedings of the 6th International Conference on Service-Oriented Computing. ICSOC '08, Berlin, Heidelberg, Springer-Verlag (2008) 708–721
18. Zou, J., Pavlovski, C.J.: Towards Accountable Enterprise Mashup Services. In: Proceedings of the IEEE International Conference on e-Business Engineering. ICEBE '07, Washington, DC, USA, IEEE Computer Society (2007) 205–212
19. Barhamgi, M., Benslimane, D., Ghedira, C., Gancarski, A.: Privacy-preserving data mashup. In: Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on. (2011) 467–474
20. Beckman, B.: Why LINQ Matters: Cloud Composability Guaranteed. Queue **10** (2012) 20:20–20:31