A-MaGe: Atomic Mashup Generator for the Web of Things

Ege Korkan*[1][0000−0003−4910−4962], Fady Salama[1][0000−0001−9225−6625], Sebastian Kaebisch[2][0000−0002−0544−4204], and Sebastian Steinhorst[1][0000−0002−4096−2584]

[1] Technical University of Munich, Munich, Germany {ege.korkan, fady.salama, sebastian.steinhorst}@tum.de
[2] Siemens AG, Munich, Germany sebastian.kaebisch@siemens.com

Abstract. Individually, Internet of Things (IoT) devices are often not able to achieve complex functionalities and, therefore, need to be composed together into useful mashups. However, given the current fragmentation of the IoT domain, designing a mashup is still a manual task that is time-consuming and error-prone. The introduction of the Thing Description (TD) from the World Wide Web Consortium (W3C) is meant to facilitate the interoperability between IoT devices and platforms by providing a standardized format to describe the network interfacing of entities, called Things, participating in the Web of Things (WoT). Furthermore, the System Description (SD) extension introduces the notion of Atomic Mashups (AMs), small mashup building blocks that are easier to design. However, designing AMs remains a manual task and, given the rising complexity of IoT devices, manually exploring the resulting design space is infeasible. In this paper, we introduce A-MaGe: a method and its open-source implementation that takes the TDs as an input and uses predefined templates, user-configurable rules, semantic annotation filtering and natural language processing to automatically explore and reduce the design space. SD-compliant UML Sequence Diagrams of the resulting mashups are presented to the human agent for further selection to generate the SD of the mashup as well as implementation code based on the W3C WoT Scripting API. We show that the generation process is fast, allowing multiple iterations by the human agent to increase reduction and we evaluate the filtering power of different filters and constraints. Thus, in combination with the TD standard, our method ensures easy composition of services in heterogeneous environments.

Keywords: Web of Things · Mashup Composition

1 Introduction

The domain of Internet of Things (IoT) has been rapidly growing, with the number of connected devices projected to increase to 50 billion devices by 2030 [1]. With this vast increase comes the challenge of connecting devices from different vendors. To facilitate it, vendors offer IoT platforms, software that handles

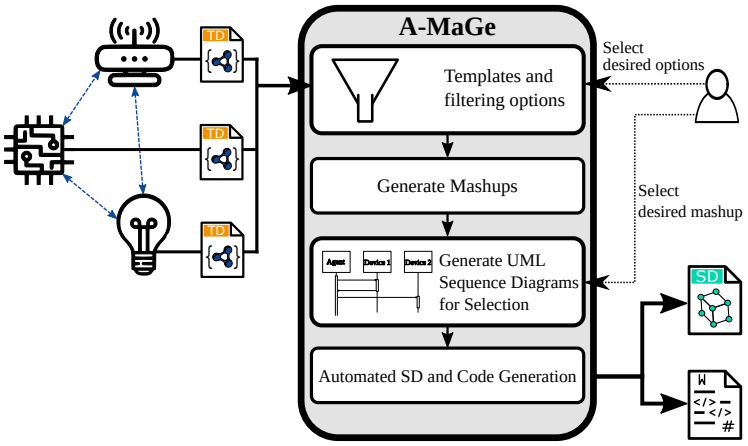

Fig. 1: A-MaGe takes Thing Descriptions (TDs) and based on constraints and filters, Atomic Mashups are generated and presented to the human agent in form of UML Sequence Diagrams. The human agent can then choose to generate the System Description (SD) and the code for the selected mashup.

the communication of different devices and exposes the functionalities. However, there are currently over 620 different IoT platforms on the market [2] and this causes a high fragmentation in the IoT domain as well as a difficulty in developing applications that leverage functionalities from the resulting silos.

To address this problem, the World Wide Web Consortium (W3C) proposed the Web of Things (WoT) architecture as a standardized means to allow the interoperability of different IoT platforms [3]. The main building block of the WoT architecture is the Thing Description (TD) [4], which is a JSON-Linked Data (JSON-LD) document [5] that is both machine- and human-readable and describes the network-interfacing of the interaction affordances offered by any IoT entity, called a *Thing* in the context of this paper.

However, TDs have no means to describe how a system of *Things* interacts together to offer some functionality. To this end, the System Description (SD) was proposed [6], a superset of the TD that offers additional keywords for describing such systems, called mashups in the context of this paper. The SD also specifies a second representation format for mashups using a subset of the Unified Modeling Language (UML) Sequence Diagrams, as well as an algorithm for converting one representation to the other. To describe complex functionalities, the SD uses a sequence of building blocks that together form an execution sequence called a Path. The smallest building block of a Path is an Atomic Mashup (AM), in which a mashup controller performs a specific number of interactions, waits asynchronously for the results of these interactions and, based on these inputs, performs a series of asynchronous output interactions.

1.1 Problem Statement

While the TDs offer an abstraction level that eases the process of designing and creating AMs, the process is still a manual task in which the developer needs to go through the whole collection of TDs to find the interaction affordances that are needed and suitable for the desired functionalities. Furthermore, a better written and annotated TD that exposes more metadata about the TD and its interaction affordances improves the understanding, but the added metadata introduces more information that a human agent has to manually process and consider when designing mashups. And finally, the resulting design space to be explored increases exponentially with the total number of interaction affordances in a system. Thus, the increasing complexity and capabilities of IoT devices, the increasing complexity of the written TDs as well as the increasing complexity of the desired mashups translate into the manual exploration of the design space, being both time-consuming and error prone. There are solutions to automate the generation of mashups which are discussed in Section 4, but to the best of our knowledge, none are centered around the TD standard without extending its standardized core vocabulary. Hence, the generation of mashups using the core TD vocabulary remains unexplored.

1.2 Contributions

In this paper we introduce A-MaGe, a method and a corresponding implementation as a solution for system designers to automatically reduce the design space that needs to be explored manually as well as automate the creation of AMs as illustrated in Figure 1. In particular, we make the following contributions:

- We introduce a method that takes TDs as an input and generates AMs that conform to predefined templates as well as user-defined constraints and filters leading to a reduction in the design space, introduced in Section 2.
- We propose a tool that uses the above-mentioned method to generate SD-compliant UML Sequence diagrams, allowing further selection of the AMs for automatic SD and code generation based on SD-algorithms.
- We show that the above-mentioned method achieves a design space reduction of several orders of magnitude, while being sufficiently fast for a human agent to allow multiple iterations of filtering to further reduce the design space, explained in Section 3.

Section 4 explores other approaches and related work for mashup composition and Section 5 concludes this paper.

2 A-MaGe Methodology

A-MaGe: an Atomic Mashup Generator is a method that is able to automatically explore and reduce the possible design space of Atomic Mashups (AM) given a set of TDs as an input with minimal direct intervention from a human agent. It relies on the AM abstraction defined in the SD which we describe in the following paragraph.

Atomic Mashup: A unique building block defined by the SD is the AM, which describes an undividable execution sequence that performs a specific functionality, similar to atomic operations in programming. An AM is defined as an unordered sequence of interactions performed by a mashup controller, called receive/input interactions (*readproperty*, *observeproperty*, *subscribeevent* or *invokeaction*), followed by an unordered sequence of interactions performed called send/output interactions (*writeproperty* or *invokeaction*). This makes it possible to describe synchronous and asynchronous sensing-actuating behaviors of a system and can then be combined using the aforementioned building blocks such as loops and conditional execution to achieve any desired system behaviour.

Given a system of *Things*, we can define the set all interaction affordances that can be considered as inputs as In_{tot} and similarly Out_{tot} for output interactions. For AMs with a specific input length l_{in} and specific output length l_{out}, we can calculate the resulting design space using the following equation:

$$C(|\mathrm{In}_{tot}|, l_{in}) \cdot C(|\mathrm{Out}_{tot}|, l_{out}) = \frac{|\mathrm{In}_{tot}|!|\mathrm{Out}_{tot}|!}{l_{in}!l_{out}!(|\mathrm{In}_{tot}| - l_{in})!(|\mathrm{Out}_{tot}| - l_{out})!} \quad (1)$$

with $C(a, b)$ denoting the combination formula.
On the other hand, the design space of mashups in general can be used with the permutations function:

$$P(n, k) = \frac{|A|!}{(|A| - k)!} \quad (2)$$

with $|A|$ denoting the number of interaction affordances in a system and k the mashup length respectively.

Looking at an example of a system with four *Things* exposing five input and five output interactions each and a desired mashup with two input and two output interactions, meaning that $|A| = 40$, $|In_{tot}| = |Out_{tot}| = 20$, $k = 4$, $l_{in} = l_{out} = 2$. Using these parameters, we can calculate using Equation 1 that the maximum number of AMs that can be generated is 36100, in contrast to 2193360 mashups in total as per Equation 2, which means that in this case there is a 98.35% reduction in the design space that needs to be explored manually.

2.1 Design Space Reduction Using Templates and Constraints

A human agent designing a mashup may have some prior expectations and constraints on how the mashup should operate or such constraints may arise during the design phase, which can be added incrementally. A computer can take advantage of these constraints and expectations to further reduce the design space and generate mashups that adhere to them, making it easier for a human agent to review and evaluate the results. With A-MaGe, we propose:

1. Filtering the considered mashup space by limiting the number of Things or interactions considered for input or output
2. Matching input and output interactions who use the same vocabulary
3. Semantic context matching of input and output interactions, meaning only interactions with annotations from the same vocabulary are considered.

4. Data type based filters to match input and output interactions based on their Data Schemas or to filter out an interaction based on its type
5. Template rules to choose how the controller receives its inputs in order to limit the *InputTypes*
   - **Subscription-driven template**: The mashup controller starts by subscribing to events or observing properties from input *Things* and waits asynchronously for the data pushes.
   - **Read-driven template**: The mashup controller starts by reading a set of properties from input *Things*.
   - **Action-driven template**: The mashup controller starts by invoking a set of actions in input *Things* and receives the interactions' outputs.
   - Allowing mashups that mix the above-mentioned templates or not. Mixed template mashups include multiple input interaction types.
6. matching input and output interactions filters using Natural Language Processing (NLP) based on the similarity of their names using a similarity score Word2Vec model [7] and based on the similarity of their descriptions by augmenting the Word2Vec approach using Word Mover's Distance algorithm [8].
7. filtering mashups based on specific semantic annotations and/or interactions, which can be described using Linear Temporal Logic (LTL) formulas $\mathbf{F}\phi$ and $\mathbf{G}\neg\phi$ respectively. Our method proposes three path variables ϕ:
   (a) ϕ_1: An interaction from a TD, that was annotated on the top-level with a specific semantic annotation, was performed.
   (b) ϕ_2: An interaction with a specified semantic annotation was performed.
   (c) ϕ_3: A specific interaction was performed.
   To allow a granular selection, these constraints can be specified individually to input, output, and input/output *Things* in case of ϕ_1 as well as to each type of input and output interactions in case of ϕ_2 and ϕ_3, respectively.

Based on the AM concept, the filters and constraints provided by the human agent, our method generates all the possible mashups. These are then presented to the human agent in the form of an SD-compliant UML Sequence Diagram. The human agent can view them and further adjust the filters and constraints. When the desired mashup is found, the human agent can then choose to generate the equivalent SD. The Sequence Diagram is then converted to an SD document using the SD conversion algorithm and the human agent can then automatically generate executable code according to the WoT Scripting API [9].

3 Evaluation

To evaluate A-MaGe, we implemented our proposed method in the **W**oT **A**PI **D**evelopment **E**nviroment (WADE) [3] [10]. However, our method does not rely on any specific programming language or framework to function. We evaluate the viability of our approach by looking at the execution time of our method and explore the filtering power of different user-defined constraints. Therefore, we perform two different tests[4], which are described in detail in this section.

[3] https://github.com/tum-esi/wade

[4] Both tests are done using a computer with an Intel© Core™ i7-8750H Processor, 8 GB of DDR4-2666 memory, Windows 10 Home 64-bit operating system

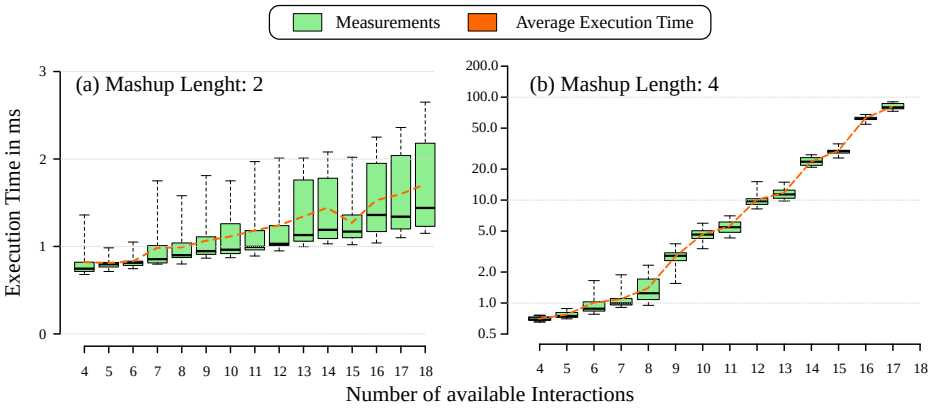

Fig. 2: We perform a number of measurements for mashups with the lengths two, four, six, eight and for interaction pools with sizes between the size of the mashup and 20 interactions. Note: The y-axis of Figure b is in logarithmic scale.

Testing execution time: In this test, we estimate the upper bound of execution time needed to generate all mashups given a specific mashup length and number of available interactions. Hence, we run A-MaGe with all templates enabled, as well as allowing mixed template mashups, but without any further constraints or filters to be able to generate the maximum number of mashups. We perform test runs for mashups with the lengths two, four, six, eight and for interaction pools with sizes between the size of the mashup and 20 interactions. The execution time for each specific test is measured 20 times to account for execution time fluctuations and the results of measurements for the mashups length of two and four can be viewed in Figure 2.

Given that intended scope is small AMs and given these findings, we conclude that our method is viable and is able to generate an exhaustive list of all mashups conforming to certain constraints with acceptable speeds. The process of the human agent further adjusting the filters and constraints and re-running the code takes at most a few seconds, allowing for multiple iterations of filtering and generation in a small span of time.

Testing filtering power: We also perform a set of measurements to test the filtering power of different filters and constraints in different scenarios. We selected a number of filters on three different systems from three different domains: smart agriculture, smart home and smart industry. Each system differs in the devices used as well as the variety in the input and output interactions or multiplicity of Things. For each of these setups, we apply a selected number of filters and constraints one at a time and record the number of generated mashups. The results of this experiment can be viewed in Figure 3.

4 Related Work

There are multiple approaches for semi- and fully automated (web) service compositions in literature. [11,12] proposes an approach based on the RESTdesc

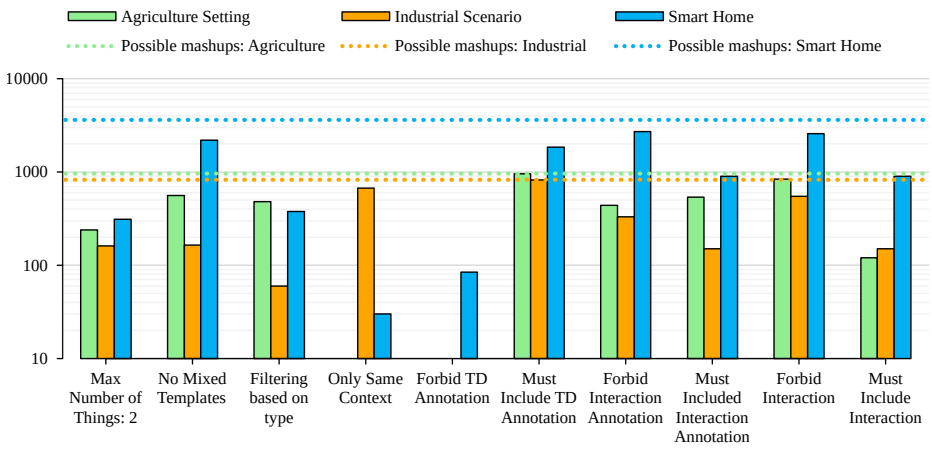

Fig. 3: We evaluate the filtering power of different filters and constraints in different scenarios. The results show that coarse forbidding using annotations is more powerful than granularly forbidding specific interactions, but the opposite is true for enforcing an annotation or interactions to be included.

ontology [13], that is able to describe REST APIs and the relationship between them. Both approaches allow the user to define a set of goals to be achieved and use a semantic reasoner that is able to parse and logically chain APIs based on semantic reasoning to achieve this goal, but they differ in how they represent the goals. [12] uses goals similar to LTL formulas used in our method, where a specific API should be performed and the reasoner finds the chain of APIs that can be connected together that lead to the desired API. On the other hand, [11] allows the user to define the desired state that a mashup should achieve. Therefore, [11] augments the RESTdesc with a semantic description of states and state transition to allow for semantic reasoning about states. Compared to both of these approaches, our method utilizes the TD ontology, which is not restricted to any specific protocol or architecture, as long as the protocol bindings are defined. Hence, our method is more universally applicable. Thus, to the best of our knowledge, no other method was proposed that leverages the TD and the AM abstraction for mashup design space exploration and automatic mashup composition.

5 Conclusion

In this paper, we proposed A-MaGe, a method that takes a set of TDs, as well as multiple filters and constraints as an input, and is able to automatically generate an exhaustive list of all possible Atomic Mashups (AMs) that adhere to the specified constraints. We started by formally defining the design space of mashups in general and showed that by focusing on AMs, we decrease the design space by several orders of magnitude. Subsequently, we introduced our method that uses a set of pre-defined templates, as well as filters and constraints that allow a human agent to further decrease the design space. We showed that

our method is capable of generating AMs in maximum a few seconds and that filtering power of different filters and constraints work in different application domains. Both evaluations show that our method a viable approach while being universally applicable to all WoT devices.

References

1. Mercer, D.: Global Connected and IoT Device Forecast Update (May 2019), https://www.strategyanalytics.com/access-services/devices/connected-home/consumer-electronics/reports/report-detail/global-connected-and-iot-device-forecast-update, accessed on 26.11.2020
2. Lueth, K.L.: IoT Platform Companies Landscape 2019/2020: 620 IoT Platforms globally. Website: https://iot-analytics.com/iot-platform-companies-landscape-2020/ (December 2019), accessed on 27.11.2020
3. Kovatsch, M., Matsukura, R., Lagally, M., Kawaguchi, T., Toumura, K., Kajimoto, K.: Web of Things (WoT) Architecture. Tech. rep. (April 2020), https://www.w3.org/TR/2020/REC-wot-architecture-20200409/
4. Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M.: Web of Things (WoT) Thing Description. Tech. rep. (April 2020), https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/
5. Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P.A., Lindström, N.: JSON-LD 1.1 (July 2020), https://www.w3.org/TR/2020/REC-json-ld11-20200716/
6. Kast, A., Korkan, E., Käbisch, S., Steinhorst, S.: Web of Things System Description for Representation of Mashups. In: 2020 COINS. pp. 1–8 (2020). https://doi.org/10.1109/COINS49042.2020.9191677
7. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
8. Kusner, M.J., Sun, Y., Kolkin, N.I., Weinberger, K.Q.: From Word Embeddings to Document Distances. In: Volume 37. p. 957–966. ICML'15, JMLR.org (2015)
9. Kis, Z., Peintner, D., Aguzzi, C., Hund, J., Nimura, K.: Web of Things (WoT) Scripting API (November 2020), https://www.w3.org/TR/2020/NOTE-wot-scripting-api-20201124/
10. Schlott, V.E., Korkan, E., Kaebisch, S., Steinhorst, S.: W-ADE: Timing Performance Benchmarking in Web of Things. In: Web Engineering. pp. 70–86. Springer International Publishing, Cham (2020)
11. Mayer, S., Verborgh, R., Kovatsch, M., Mattern, F.: Smart Configuration of Smart Environments. IEEE Transactions on Automation Science and Engineering **13**(3), 1247–1255 (2016). https://doi.org/10.1109/TASE.2016.2533321
12. Ventura, D., Verborgh, R., Catania, V., Mannens, E.: Autonomous composition and execution of REST APIs for smart sensors. In: CEUR Workshop Proceedings. vol. 1488, pp. 1–12 (2015), http://ceur-ws.org/Vol-1488/paper-02.pdf
13. Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Mannens, E., Van de Walle, R., Vallés, J.G.: RESTdesc—a Functionality-Centered Approach to Semantic Service Description and Composition. In: Proceedings of the 9th ESWC, Crete, Greece. pp. 27–31 (2012)