Towards Distribution Options in the End-User Development of Multi-device Mashups

Oliver Mroß and Klaus Meißner

Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany
{oliver.mross, klaus.meissner}@tu-dresden.de

Abstract. In recent years several approaches addressed the development of applications for multi-device environments. Due to the mobility of devices, the volatile availability of their resources, e.g., limited battery life or communication quality, and changing user requirements, the application developer cannot anticipate every situation, e.g., new or leaving devices, their capabilities and interaction resources, at the application's design time. Hence, end-users should be able to create and customize multi-device applications at run time. Available solutions already provide user-centered development tools, but they are not sufficient and lack intelligent assistance to offer possible design options, in particular in terms of the application's distribution. In this paper, we present our ongoing work to provide assistance in the end-user development of *multi-device mashups* (MDM) by recommending distribution options with respect to available devices, their capabilities and resources to ease the user-driven mashup development. Furthermore, we investigate the use of aggregation rules and composition ranking rules to rate distribution options regarding their quality properties and their context fitness, e.g., when the MDM is used at home or in an automotive environment.

Keywords: Multi-device Applications · Multi-device Mashups · Mashups · Distribution Options · End-user development

1 Introduction

With the increasing availability of web-enabled (mobile) devices and the paradigm shift towards the Web of Things (WoT) [12, 9] the Web has become the ubiquitous platform for a new type of applications, in which several devices and their capabilities are combined and used together simultaneously according to the application domain. Following the *mashup* approach in [9], such multi-device applications are build by composing devices and their services on the application layer via Web APIs hiding heterogeneous platforms and service implementations. Device services are connected with each other using Web communication protocols as well as standard media types [2]. We call such applications *multi-device mashups* (MDM) that are associated with new challenges in their development process. Considering the *heterogeneity* and *volatile availability* of mobile devices and their capabilities or limited resources, e.g., energy capacity, developers cannot anticipate every situation at the application's run time during its design phase. Furthermore, it is impossible to meet the changing user requirements without having a concrete understanding of the current situation an application is used in. Hence, strictly separating the design and run time leads to limited and inflexible applications. Accordingly, end-users without programming knowledge and limited experiences in adaptation cases should be supported to create/modify MDM at run time.

To the best of our knowledge, available *end-user development* (EUD) tools for multi-device scenarios lack concepts to guide users by providing hints or design options regarding the current usage context, e.g., presenting possible or most likely needed adaptation rules with respect to available device features. Furthermore, users should understand the consequences of each recommended design option on the application quality. This is illustrated in the following scenario. In a smart home environment the user wants to create a MDM to control the house's heating system depending on his/her position in the house. To this end, a *guidance system* presents several design options considering resources and capabilities of the smart home and its devices. Fig. 1 presents an overview of the scenario. The first option (left-side) wires together following software compo-

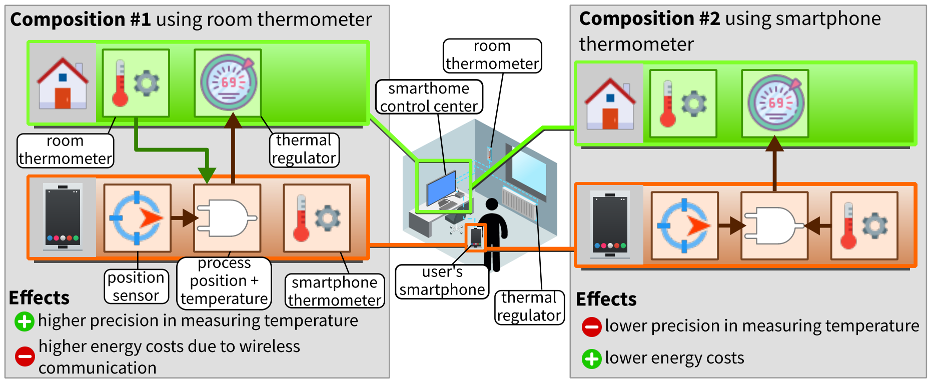

Composition #1 using room thermometer
room thermometer
thermal regulator
position sensor
process position + temperature
smartphone thermometer
Effects
higher precision in measuring temperature
higher energy costs due to wireless communication
room thermometer
smarthome control center
user's smartphone
thermal regulator
Composition #2 using smartphone thermometer
Effects
lower precision in measuring temperature
lower energy costs

Fig. 1. Overview of several composition options in smart home scenario

nents encapsulating devices or digital processes: the user's smartphone indoor positioning sensor, the digital thermometer and the control interface of the thermal regulator within the smart home and a software component executed by the smartphone processing the user's position and room temperature to control the thermal regulator. Another design option (right-side) encompasses an alternative composition. Instead of using the room thermometer, the smartphone's build-in thermometer is wired together with the indoor positioning sensor and the processing component to control the thermal regulator. Both design options are associated with different *effects* on the application's precision and energy demands. To highlight the effects of each distribution option, mechanisms are needed that provide users with information on probable application states.

The *contribution* of this paper is a new architecture of the proposed end-user guidance system based on novel meta-models to ease the user-driven development by recommending *distribution options* context-sensitively to create/modify MDMs at run time. In addition, distribution options are rated semi-automatically considering their effects on the application quality and context suitability to assist end-users in choosing optimal design decisions, generally making the development process more transparent. Moreover, distribution options provide solutions to the problem of adapting black-box components in the context of dynamically distributing composition fragments between heterogeneous devices and their platforms. In our approach, we address the problem by replacing mashup components with functional alternatives but usable on the target device. The remaining of this paper is structured as follows. In Section 2 we discuss related work. Subsequently, in Section 3 we introduce fundamentals of our approach and characterize relevant aspects of the user-driven development of MDM. Furthermore, we present novel meta-models for distribution options and quality aggregation rules. Afterwards, we present the overall architecture of the end-user guidance system in Section 4 and explain relevant features to assist end-users during their development process based on the distribution options concept. Finally, in Section 5 we draw conclusions from our approach and discuss future work.

2 Related Work

In the EUD domain of Internet of Things (IoT) applications, different metaphors and programming styles can be distinguished [15]. On the one hand, *graphical mashup editors* abstract devices, their services, UI elements or application logic as visual components that are wired together by experienced developers or end-users, such as Node-RED[1] or glue.things [11]. Moreover, approaches such as AppInventor[2] or openHAB [7] are using EUD tools based on the *jigsaw puzzle metaphor*. As discussed in [5, 6], such approaches are mainly suitable for experienced developers knowing fundamental programming concepts. They do not provide solutions to the problem that end-users lack experiences in modifying applications in continuously changing multi-device environments. Not only using a suitable metaphor is sufficient, but additional assistance is required. On the other hand, *rule-* and *language-oriented toolkits* [4, 6, 8] target end-users without programming experiences to create multi-device applications. For instance, Coutaz and Crowley present the AppGate approach [4] that supports end-users in the specification of smart home applications by syntax-oriented rule suggestions. This allows end-users to create *event-condition-action* (ECA) rules exploratively, but they have to anticipate application scenarios initially. This is difficult for inexperienced end-users. In this context, we assume that predefined applications as well as context information (e.g., usage history, preferences etc.) can be used to recommend design options.

In the domain of *mashup* applications, Daniel and Matera consider in [5] assistance capabilities as dimension for EUD mashup tools. They present quality-aware and pattern-based component recommendation features to reduce the burden in the development of mashups. Our approach is based on similar models to specify component interfaces and compositions, but goes beyond the single-device paradigm assumed in [5]. Although the model-based recommendation may be adopted to MDM, it lacks meta-model concepts on the component and composition layer to specify distributed composition fragments, for example, component descriptors are missing context requirements to derive potential target devices. However, we consider the quality aware recommendation as solution to the problem that end-users are not able to anticipate the effects of recommended adaption options. In our approach, quality-aware recommen-

[1] https://nodered.org/
[2] http://www.appinventor.org/

dations should estimate quality properties of the multi-device mashup and provide an insight to the likely future application state.

In the previous approaches, the application's distribution state and its adaptation is not taken into account explicitly, but this is of particular interest in the case of dynamic multi-device environments. In the research domain of distributed mashups, end-users are able to control the UI distribution at run time. For instance, the MultiMasher approach of Husmann et al. [10] supports the EUD of distributed mashups and allows to distribute and synchronize arbitrary web UI elements across several devices. However, due to the focus on the discovery and distribution of visual web UI elements, the integration of "headless" services such as web or device services without an UI is limited. Moreover, MultiMasher and other EUD approaches [8, 14] in the distributed UI (DUI) research domain lacks assistance capabilities that consider context properties, e.g., target device resources. Thus, they are not able to draw the user's attention to the consequences of distribution state changes on the application quality. Thus, user-driven UI redistribution activities may result in error-prone application states, e.g., when mashup components are migrated to low-resource devices. In our approach, we address this problem of insufficient user experiences in dealing with UI redistribution activities by calculating quality properties at the (distributed) composition layer. Mashup quality properties can be used to rank design options or filter out critical ones.

In summary, there are already approaches to support the EUD of multi-device applications, but they lack concepts to guide end-users during the development considering the following problems: (a) the end-users' limited abilities to foresee potential situations in the multi-device environment, e.g., volatile device resources, and (b) the end-users' insufficient experiences in creating or adapting MDMs at run time as well as (c) the users' missing capabilities to anticipate adaptation effects on the application quality.

3 EUD of Multi-device Mashups

In this section, first we outline fundamentals of our approach and aspects of the user-driven development of MDM. On this basis, we explain concepts regarding distribution options of multi-device mashups. Moreover, since users should be able to evaluate them and to understand their effects on the application's quality, we present a new meta-model for specifying rules to aggregate quality properties of MDM context-sensitively. Finally, we discuss the process of providing distribution options as possibilities to design or adapt MDM by end-users at run time.

3.1 Fundamentals

In our approach, a *multi-device mashup* consists of several *black-box components* that are loosely coupled together and are distributed across several devices at run time. The mashup's composition, its data flow relations and its distribution state are specified in the *conceptual*, *communication* and *distribution model* respectively as entities of the same *mashup composition model*. Further details regarding the composition meta-model are described in [13].

Each mashup component may encapsulate web services, application logic, UI elements or device features and adheres to the Semantic Mashup Component Description

Language (SMCDL) [16] providing the following concepts to specify the component interface: *properties*, *events*, *operations* and *capability* descriptions. In addition, domain ontology concepts are referenced in order to specify data and functional semantics as well as context requirements, e.g., relevant sensors or software APIs. Context requirements are used to determine a component's executability regarding device profiles.

3.2 EUD as Decision-making Process

We consider the distributed runtime environment of MDMs as a *proactive assistant* guiding end-users by providing useful design options with respect to the usage context (devices, user profile or environment). Users act as *decision-makers* with regard to the application's structure, functionalities and their distribution based on design options provided by the system. Thus, *observability, traceability* and *comprehensibility*, as introduced by Coutaz [3], are relevant aspects of the end-user development of MDM.

Observability: This aspect implies to have an overview of included components, their features, data flow relations and their distribution. Based on this information, users are able coordinate their design decisions.

Traceability: This aspect addresses the need to understand why an adaptation is relevant in the current situation. For instance, the critical battery level of a smartphone is the cause of the application's redistribution. This requires to *present the adaptation's cause*. Due to the different relevance of distinct adaptation causes, not every design option has the same importance from an application's perspective. In the previous example, the mashup's redistribution caused by a critical battery level is more relevant in comparison to the addition of optional mashup components when new devices become available at the same time. This requires *prioritizing distribution options*.

Comprehensibility: Since, there could be multiple adaptations per situation, users should be able to comprehend the effects on the application's quality for each design option. In this context, the challenge is the multi-dimensional nature of the mashup quality concept and its context dependencies. For instance, the lifespan and responsiveness of an MDM depends on available energy resources and the current workload of each device. This results as requirement for *distributed quality aggregation* features that consider the static and dynamic nature of mashup quality properties.

3.3 Distribution Options Meta-model

Considering the previous requirements, we specified the meta-model of *distribution options*, as presented in Fig. 2. Primarily, it supports the dynamic integration and distribution of composition fragments (single/multiple components and data flow channels) in the context of MDM. It does not address UI layout or styling adaptations or modifications on the component interface layer (e.g., reconfigure state or non-functional properties). Several *distribution options* are the result of a context-sensitive composition redistribution algorithm. Algorithm details are beyond the scope of this paper. The redistribution is triggered by generic or domain-specific *event-condition-action* (ECA) rules. They are integrated as (a) preinstalled generic rules of the MDM runtime environment or are (b) specified by users *implicitly* using domain-specific *mashup templates* or (c) *explicitly* using authoring tools, such as described by Ghiani et al. [8]. Each distribution option is caused by a *context event*, for instance, the change of device resources

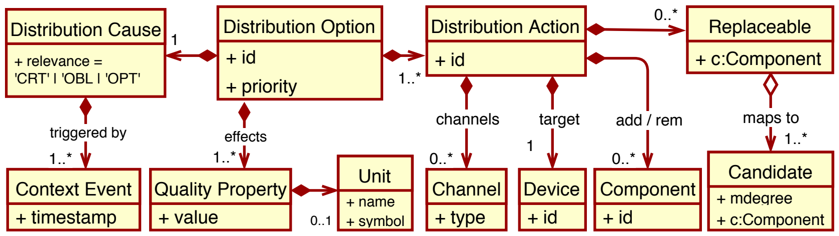
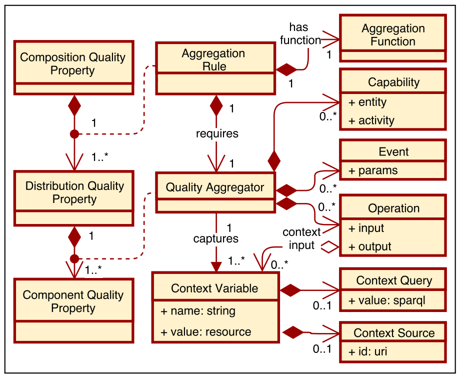


Fig. 2. Overview of the adaptation option model

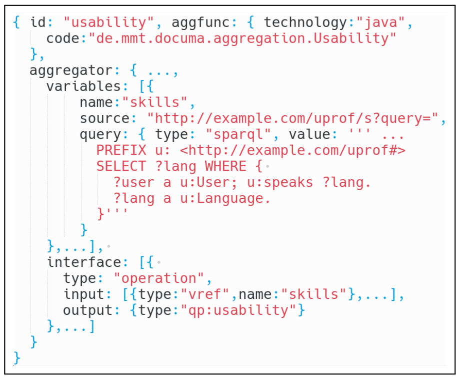
or environmental conditions. It is described as *distribution cause* associated with a *relevance* attribute. From the application's perspective, a context change has a specific relevance. On one hand, it can be *critical* in the case of existential device resources, such as limited energy or network bandwidth. On the other hand, there can be *obligatory* or *optional* distribution causes. A distribution option is obligatory in terms of the mashup's life cycle when it loads an application and its substantial functionalities. It is optional in such cases when further composition fragments may be integrated, e.g., when new devices become available and UI components could be replicated for flexibility purposes or to support multi-user scenarios. Moreover, a distribution option may be associated with multiple *quality properties*. They are the predicted effects on the application's quality. An example is the estimated energy-consumption as described in Section 1. Considering quality properties of distribution options, users are able to evaluate each option subjectively. However, they are also ranked objectively regarding their "context fitness". We discuss this in more detail in Section 4. Finally, a distribution option comes with the integration, removal or replacement of composition fragments. Thus, it has several *distribution actions*. Each action addresses a *target device* and is associated with components to be added, removed or replaced. The latter aspect is especially important for UI migration scenarios using heterogeneous devices. In the example of a smartphone's low battery, a distribution option is provided to migrate UI components between a smartphone and a tablet or desktop PC. Since, not every mashup component can be executed on any device, due to missing I/O-capabilities, platform APIs etc., we assume there are functional alternatives, which are usable in the context of the target device instead. Hence, each distribution action may encompass additional component replacement activities as "sub-adaptations in the adaptation". Component replacements are modeled using the *replaceable* item. Each item maps a source component to alternative *candidates*, whereby each candidate is executable on the action's target device. Here, we consider several component alternatives to support higher flexibility in future use cases. For instance, when presenting an option to migrate UI components, power users can choose from several alternatives regarding their quality, e.g., cheap vs. high-efficient components. In case of novice users, the best matching candidate (derived by its matching degree with respect to the source component) is selected automatically.

3.4 Aggregation Rules

The *comprehensibility* aspect implies to support end-users in rating their design decisions and therefore to increase the transparency of the user-driven development process. This leads to the requirement for a *distributed mashup quality aggregation mechanism* that should address the following challenges: (i) Each application quality property has its specific aggregation function. For instance, the estimated application lifespan is the minimum of device-specific lifespans. (ii) An application quality property depends on static or dynamic quality features of components distributed on heterogeneous and dynamically available devices. (iii) Dynamic quality properties may depend on specific context information. For instance, the application's usability depends on user language skills, which are specified as part of the user profile. It may be provided either by a device and its platform, by a web service or not at all. (iv) During the recommendation of distribution options, each estimated quality property value is a forecast regarding the mashup's future state. Therefore, the resulting quality value is error-prone and its estimation should to be precise as possible, even under condition of noisy data as in the case of the application's energy-consumption. With regard to these challenges, in our approach application quality properties are derived using generic or domain-specific *quality aggregation rules*. Generic rules are an integral part of the MDM runtime environment. Domain-specific ones can be added by the user dynamically using authoring tools that are beyond the scope of this paper. In Fig. 3a the *meta-model of aggregation rules* is presented, which we explain afterwards.



(a) Aggregation rule meta-model



```
{ id: "usability", aggfunc: { technology:"java",
    code:"de.mmt.documa.aggregation.Usability"
  },
  aggregator: { ...,
    variables: [{
        name:"skills",
        source: "http://example.com/uprof/s?query=",
        query: { type: "sparql", value: ''' ...
          PREFIX u: <http://example.com/uprof#>
          SELECT ?lang WHERE {
            ?user a u:User; u:speaks ?lang.
            ?lang a u:Language.
          }'''
        }
    },...],
    interface: [{
      type: "operation",
      input: [{type:"vref",name:"skills"},...],
      output: {type:"qp:usability"}
    },...]
  }
}
```

(b) Sample usability aggregation rule

Fig. 3. Overview of the aggregation rule model

The meta-model is based on the assumption that every *composition quality property* is calculated using a corresponding *aggregation rule*, which is related to a single *aggregation function*. The latter is specified as source code reference and results in the quality value at the application layer. Moreover, an application quality property depends on the same quality feature of each distributed sub-composition executed by its associated device. In the meta-model, each device-specific quality property at the local composition level is represented by the *distribution quality property* concept. In relation to a device the latter concept results from the aggregation of local *component quality properties*. Analogous to the *aggregation function* at the application level, the *quality aggregator* calculates the *distribution quality property* by considering the same component quality features as well as the usage context that may influence the quality as discussed previously in the estimated run time example. Due to the dynamic availability of heterogeneous devices and the property-specific nature of aggregation rules, *quality* *aggregators* should be integrated in the application context dynamically as individual software components responsible for the acquisition of quality-related context information. Hence, their interface elements (*events, properties, operations* or *capabilities*) are described as part of the aggregation rules. After they were integrated on each client at the application's run time, quality aggregators do not directly interact with application components, and they are not visible from the end-user's perspective. Considering the context-dependency aspect of quality properties, quality aggregators may access platform-specific APIs or external *context sources* (e.g., social networks) to retrieve relevant information, e.g., the consumption of energy resources or the user profile. In the case of using external context information by querying remote web services, the rule author can specify the aggregator's operation signature and hereby specify *context variables* as input parameters. This is illustrated in the example in Fig. 3b. Context variables can be resolved either by the client-side runtime or by any *context source* on the web using its associated *context query*.

After presenting basic concepts to support the user-driven development of MDM, in the next section we will focus on actors as well as key features in the overall *distribution options recommendation* system.

4 Recommendation of Distribution Options

In this section, we present the overall architecture of the *MDM assistance system* (MAS) guiding end-users during their development process based on the *distribution options* concept. As presented in Fig. 4, we distinguish following actors: ***Composi-***

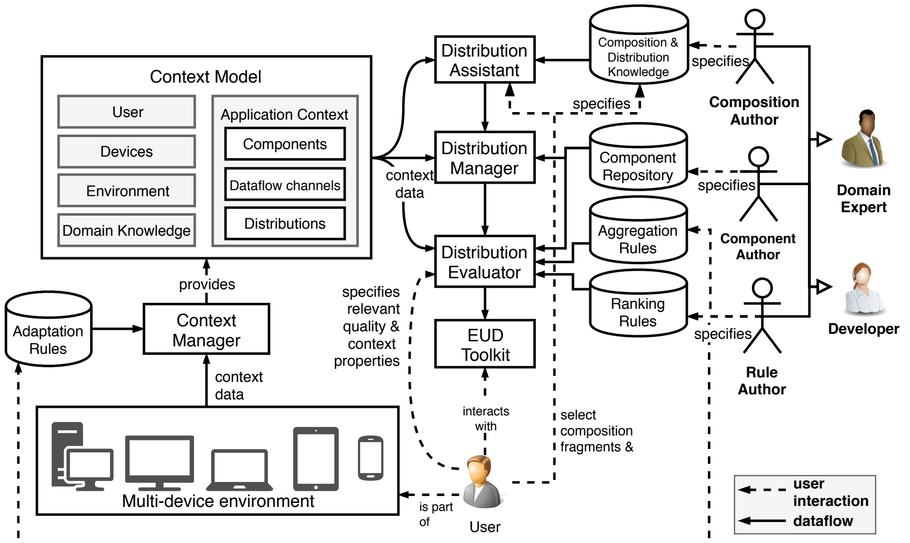


Fig. 4. Overview of the distribution options recommendation architecture

tion authors model entire MDM or *distribution patterns* based on the concept of *co-occurence patterns as introduced by Chowdhury et al. [1]*. Distribution patterns are frequently used composition fragments annotated with distribution knowledge, e.g., the data flow between a remote control component used on mobile devices and a video player executed on a presentation device (e.g., smart TV etc.). ***Component authors*** design and implement application logic, UI elements or device services as mashup components as well as their interface descriptions using web technologies and domain-specific ontologies. ***Rule authors*** specify generic or domain-specific adaptation, aggregation and ranking rules. The latter rank arbitrary compositions (incl. *distribution options*) with respect to their usage context. For instance, distribution options that contain voice I/O features are ranked higher when the user enters her car together with her mobile devices. ***Users*** with domain knowledge and no programming expertise develop or configure MDM using EUD tools provided by the client-server runtime environment.

Features of the MAS are the following, which we explain in more detail afterwards: (i) Based on predefined or automatically derived composition and distribution knowledge, the MAS recommends distribution options at run time to integrate or distribute composition fragments semi-automatically. (ii) Moreover, the effects of distribution options are calculated automatically in terms of quality properties on the application layer. As result, users can filter out undesired adaptations or may easier coordinate their design steps. (iii) Next, the MAS ranks distribution options regarding their current "context fitness". To this end, users can reuse predefined *ranking rules*.

Semi-automatic distribution. Developing or modifying MDMs, users are guided by recommended distribution options derived from context properties, e.g., available devices and their features. The *distribution assistant* matches context model entities (e.g., devices, environment or user profiles) with MDM models or *distribution patterns*, each specified as tuple $(\mathcal{C}, \mathcal{P}, \mathcal{D}, \mathcal{F})$ where several *components* $C_i \subseteq \mathcal{C}$ are associated to *context properties* $P_j \subset \mathcal{P}$ as *distribution* relations $d \in \mathcal{D}, d(C_i, P_j)$. Set $\mathcal{F}$ specifies *data flow* relations $f \in \mathcal{F}$ between components. The result of the "context-pattern" matching operation are composition fragment *candidates* ranked according to their *matching degree* and their average user rating of previous sessions given by other users. Based on top-*k* candidates, the *distribution manager* filters out candidates whose components cannot be executed on any registered device and have no functional alternatives, which can be used instead. For this, the *distribution manager* examines the interface description of each component referenced by every candidate. In relation to a device profile, each component's usability is evaluated according to its context requirements specified as part of the interface description. The result is a *component usability matrix* that includes valid component-device, component-alternatives or alternative-device pairs. Afterwards, the *distribution manager* calculates the *distribution graph* G_D that represents devices D, components C and component alternatives C' as vertices V_D, V_C and $V_{C'}$ respectively. Further, for each valid component-device or component-alternative pair there is an edge $E_{(C,D)}$ or $E_{(C,C')}$ connecting corresponding vertices. Based on graph G_D, the *distribution manager* derives further redistribution options that are considered as optional distribution recommendations per candidate. Finally, the *distribution manager* creates at least a *distribution option* for each candidate. At this, every *distribution* unit $d \in D$ of a candidate results in a *distribution action* accordingly (cf. Fig. 2).

Distribution Rating. In this step, the *distribution evaluator* tries to calculate quality properties for each *distribution option* provided by the *distribution manager*. First, the *distribution evaluator* retrieves valid aggregation rules by matching *quality aggregator* interface elements of each rule (cf. Fig. 3a) with currently available device profiles. The latter result from the *target* devices of the distribution options (cf. Fig. 2) together with directly registered devices of the application context, e.g., the owner device. If every *target* device either provides appropriate *device services* or there are mashup components with compliant interfaces, and they can be executed on the *target* device regarding their context requirements, the aggregation rule is considered as valid. In summary, the *distribution evaluator* filters out usable rules. Second, each valid aggregation rule results in a corresponding quality property at the application layer. To this end, for each rule following sub-processes are performed: (a) In the context of the current rule, for every target device a *quality aggregator* instance is created and integrated on the client-side runtime environment. Further, the *aggregator instance* calculates the quality property on the local composition layer of the associated device as described in Section 3.4. (b) Afterwards, locally aggregated quality properties are sent from each target device to the central *distribution evaluator*. Here, they are used as input to the rule's *aggregation function*. The result of its application is a quality value at the application layer. (c) Finally, the calculated quality value of the current aggregation rule is assigned to the distribution option. In summary, each valid aggregation rule of every distribution option result in a corresponding quality value, which users may perceive as guidance to choose desired distribution options.

Context-sensitive ranking. Since each distribution option may be related to several quality properties, there is no general solution to rank distribution options regarding distinct user preferences. Therefore, our approach is based on the following solutions: First, users may rank distribution options interactively according to common quality properties and their values. Second, distribution options are ranked automatically considering their quality properties and "context fitness". The first solution has to be supported on the EUD toolkit's UI layer, e.g., representing distribution options and its quality values in tabular form as well as providing interactive sorting features. Regarding the second solution, the MAS provides *context-specific ranking rules*, which are specified by experts or are predefined as part of mashup templates. For instance, when distribution options are recommended in an automotive environment, those that provide voice I/O capabilities are preferred and thus ranked higher. A corresponding example rule is presented in Fig. 5. With help of the *when*-clause, valid *ranking rules* are de-

```
{
  when: {
    match: ''' ...
     prefix e: <http://example.com/documa/env#> .
     ask { ?env a e:Environment;
     owl:sameAs(e:Automotive). }
     '''
  },
  select: {
   type:"complex", op:"OR", children:[
    { type:"cmp", caps:[{ a:"ac:out",e:"m:voice"}],...},
    { type:"cmp", caps:[{ a:"ac:in" ,e:"m:voice"}],...}]
  }, rating: 4.5
}
```

Fig. 5. Sample ranking rule

termined in consideration of the current context model by the *distribution evaluator*. The *select*-clause specifies necessary composition fragments and their capabilities. As shown in Fig. 5, composition patterns can be represented as combination of logical

OR and AND statements to specify alternative and necessary component capabilities respectively. The rule's *rating* value is assigned to the distribution option when context and composition conditions are fulfilled. Here, we assume a five-point rating scale. Moreover, several *ranking rules* are applied to each distribution option and the mean rank per option is calculated from the mean quality value and the average context rating. Finally, the distribution options are sorted automatically according to their rank in descending order and are presented to the end-user by the EUD toolkit.

5 Conclusion and Future Work

Due to the mobility and limited resources of devices, developers (incl. end-users) of multi-device applications have to cope with the problem of dynamic context changes. In this context, we argue it is not sufficient to specify adaptation rules at design time only or at run time. Developers, especially end-users, cannot anticipate every situation at design time or are able to perform needed adaptations at run time, e.g., when a control device falls short on energy resources. Thus, they should be guided by an intelligent assistant to create/adapt applications at run time. To address this challenge, we introduced the concept of *distribution options* to modify MDMs dynamically. However, the limitations of our approach are (a) its dependence on predefined distribution, composition and rule models, the lack of assistance for (b) functional testing of resulting MDM and (c) detailed configuration of distribution options. Considering the first limitation, the quality of distribution options result from available applications, distribution patterns or rules. In this context, we assume that expert developers or the runtime administrator are responsible for sufficient quality that comes with a substantial initial effort. But this may be compensated by the community of expert users. Second, distribution options are intended as supplemental design recommendations in association with a graphical mashup editor end-users may apply to create or modify MDMs. Currently it lacks testing capabilities for distribution options that require a sufficient infrastructure to simulate multi-device scenarios and virtual devices based on user requirements. In this context, additional research is needed. Moreover, we question whether non-technical savvy end-users are not overwhelmed by such simulation capabilities as well? The last limitation addresses missing features to adjust distribution options in more detail, e.g., assigning events of a remote control component executed on a smartphone to another video playback feature executed on a smart TV. This might be of interest for power users. However, this comes with the need to track modifications to recalculate their effects on the application quality. Here, further investigations are needed.

In our future work, we focus on a wizard-like EUD toolkit to capture user requirements more precisely and to specify the application's distribution state based on abstract and distributable device capabilities. This allows to derive suitable composition fragments and distribution options more precisely and enables end-users to control recommended design options even more actively.

Acknowledgments

The work of Oliver Mroß is funded by the European Union and the Free State of Saxony within the EFRE program.

References

1. Chowdhury, S.R., Daniel, F., Casati, F.: Recommendation and weaving of reusable mashup model patterns for assisted development. ACM Trans. Internet Techn. 14 (2014)
2. Ciortea, A., Boissier, O., Zimmermann, A., Florea, A.M.: Responsive decentralized composition of service mashups for the internet of things. In: Schneegass, S., Schmidt, A., Michahelles, F., Kritzler, M., Ilic, A., Kunze, K. (eds.) IOT. pp. 53–61. ACM (2016)
3. Coutaz, J.: Meta-user interfaces for ambient spaces. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) Task Models and Diagrams for Users Interface Design, Lecture Notes in Computer Science, vol. 4385, pp. 1–15. Springer Berlin Heidelberg (2007)
4. Coutaz, J., Crowley, J.L.: A first-person experience with end-user development for smart homes. IEEE Pervasive Computing **15**(2), 26–39 (2016)
5. Daniel, F., Matera, M.: Mashups - Concepts, Models and Architectures. Data-Centric Systems and Applications, Springer (2014)
6. Desolda, G., Ardito, C., Matera, M.: Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. ACM Trans. Comput.-Hum. Interact. **24**(2), 12:1–12:52 (2017)
7. openHAB e.V. Foundation: openhab: Empowering the smart home (Dec 2017), https://docs.openhab.org/addons/uis/habmin/readme.html
8. Ghiani, G., Manca, M., Paternò, F.: Authoring context-dependent cross-device user interfaces based on trigger/action rules. In: Holzmann, C., Mayrhofer, R. (eds.) MUM. pp. 313–322. ACM (2015)
9. Guinard, D., Trifa, V., Wilde, E.: Architecting a mashable open World Wide Web of Things. Technical Report 663, Institute for Pervasive Computing, ETH Zurich (Feb 2010), http://www.vs.inf.ethz.ch/publ/papers/WoT.pdf
10. Husmann, M., Nebeling, M., Pongelli, S., Norrie, M.C.: Multimasher: Providing architectural support and visual tools for multi-device mashups. In: Benatallah, B., Bestavros, A., Manolopoulos, Y., Vakali, A., Zhang, Y. (eds.) WISE (2). Lecture Notes in Computer Science, vol. 8787, pp. 199–214. Springer (2014)
11. Kleinfeld, R., Steglich, S., Radziwonowicz, L., Doukas, C.: glue.things: a mashup platform for wiring the internet of things with the internet of services. In: WoT. pp. 16–21. ACM (2014)
12. Mathew, S.S., Atif, Y., Sheng, Q.Z., Maamar, Z.: The web of things — challenges and enabling technologies. In: Bessis, N., Xhafa, F., Varvarigou, D., Hill, R., Li, M. (eds.) Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence, Studies in Computational Intelligence, vol. 460, pp. 1–23. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-34952-2_1
13. Mroß, O., Meißner, K.: Towards user-centered distributed mashups. In: Proceedings of the 2014 Workshop on Distributed User Interfaces and Multimodal Interaction. pp. 11–14. DUI '14, ACM, New York, NY, USA (2014)
14. Nebeling, M.: Xdbrowser 2.0: Semi-automatic generation of cross-device interfaces. In: Mark, G., Fussell, S.R., Lampe, C., m. c. schraefel, Hourcade, J.P., Appert, C., Wigdor, D. (eds.) CHI. pp. 4574–4584. ACM (2017)
15. Paternò, F., Santoro, C.: A design space for end user development in the time of the internet of things. In: Paternò, F., Wulf, V. (eds.) New Perspectives in End-User Development, pp. 43–59. Springer International Publishing (2017)
16. Radeck, C., Blichmann, G., Meißner, K.: Estimating the functionality of mashup applications for assisted, capability-centered end user development. In: Majchrzak, T.A., Traverso, P., Monfort, V., Krempels, K.H. (eds.) WEBIST (2). pp. 109–120. SciTePress (2016)