A Goal Decomposition Approach for Automatic Mashup Development

Lin Bai[1], Dan Ye[1], Jun Wei[1]

[1] Technology Center of Software Engineering,
Institute of Software, Chinese Academy of Sciences,
Beijing, China
{bailin, yedan, wj}@otcaix.iscas.ac.cn

Abstract. Automatic mashup aims to discover desired mashlets according to user goals automatically and combine them into an entirely new application. However, the user goals are usually high-level and coarse-grained while the mashlets are low-level and fine-grained. How to fill in the gap becomes a challenge when addressing automatic mashup development. This paper proposes a novel goal decomposition and refinement approach to handle this problem. We defined a goal model based on which we proposed a history heuristic based algorithm to build a Mashup Goal Ontology repository to enable the auto-decomposition of user goals. Then mashlets which are matching with the refined user goals can be found out and mashed up. We evaluate our approach through experimental results which demonstrate acceptable performance of the decomposition.

Keywords: goal decomposition, Mashup Goal Ontology, mashup, mashlet

1 Introduction

Mashup is an emerging application development paradigm and has gained much attention in recent years. Wikipedia explains mashup as a web page or application that uses and combines data, presentation or functionality from two or more sources to create new services [1]. For its easy and fast integration of online resources, mashup becomes another attractive technology for enterprises who are following the SOA (Service Oriented Architecture) paradigm.

Just like service discovery in SOA, mashlet discovery is also an indispensable part in mashup. Mashlet is a general term of online resources, including data, functions and presentations, which are ready to be combined and reused to coin new applications. The difference lies in that the executors of service discovery are professionals who are familiar with service description language (e.g. WSDL), service communication protocol (e.g. SOAP), etc, while the executors of mashlet discovery are usually end users without any knowledge or experience in development. This brings great challenges to mashlet discovery. For example, when we are going to transport good from one chain store to another, we can merge the following information together: the exact addresses of the two stores, the possible routes between them, and the distance of each route. In fact, there may be more than one mashlet that provides the same or similar services with different quality, performance, or user preference. Novice users usually do not know their differences and thus confused to select the one which is appropriate for their current situations most. Further, novice user may lose some important services sometimes. For example, to plan a shorter and faster route, traffic is another necessary aspect (except for distance) to be concerned, which is prone to be omitted by novice users.

To release the users from the hard and error-prone mashlet searching, automatic mashup platform aims to discover mashlets automatically according to user goals. However, the fact is that user goals are generally high-level and coarse-grained, like "transport good as fast as possible", while the mashlets which are registered in the repository to be combined are usually low-level and fine-grained with explicit functional descriptions, like "get distance", "get directions", "get traffic volume", etc. How to find out the concrete mashlets for the general user goals becomes a challenge when addressing automatic mashup development. This paper proposed a novel goal decomposition and refinement approach to fill in the gap between them. We defined a goal model based on which we proposed a history heuristic based algorithm to build a goal-ontology repository to enable the auto-decomposition of user goals. Then mashlets which are matched with the refined user goals can be found out and mashed up. Experimental results demonstrate that our approach is effective for user-goal decomposition and thus gives helpful guidance to novice users on developing their own mashup applications.

The remainder of the paper is organized as follows: We will summarize and analyze some existing solutions to the decomposition and refinement of user goals in current automatic goal-driven mashup development in section 2. Then, section 3 presents a novel history heuristic based algorithm leveraging collective intelligence of historic users to guide the decomposition of current user goals. In section 4, we evaluate our approach by a group of experiments, and section 5 concludes this paper with some discussions and future works.

2 Related Work

Goal-driven development is the main strategy adopted by current automatic mashup platform. To resolve the mismatch between the general user goals and the concrete mashlets, two kinds of approaches can be summarized from current research work.

Eric Bouillet et al [3-7] eliminate above mismatch by restricting users to specify their goals with registered mashlet descriptions, which are expressed in tags. Since registered mashlet may be numerous (considering ProgrammableWeb.com [21] as an example, there have already been 6602 mashups and 5757 APIs registered by 26/4/2012), finding out the exact mashlet description tags as user goals are tedious and toilsome. To facilitate end users to specify their goals, Eric Bouillet et al, on the one hand, categorize tags into facets which mean a category of tags with some common features. Therefore, a hierarchical goal structure is built and end users can refine their goals step by step in a navigation manner. On the other hand, the authors propose a goal refinement strategy, prompting users with possible goals which are generated by a customized AI planner, to guide users to refine their current goals. However, as the authors claim that this approach is proposed to be applied in flow-based information processing systems. The flow planning based goal refinement mechanism is limited for other mashup scenarios, such as event-triggered mashup which contains no evident data flows or logic flows but glues mashlets together through a set of discrete events in a "wiring[18]" manner (Compared with the "wiring" manner, there is a "piping[18]" manner which means mashlets are linked with a explicit data flow or logic flow). Furthermore, the navigation from high-level goals to lower level goals lacks of necessary guidance and depends totally on users' own subjective experience. In this paper, we also adopt a hierarchical goal structure, which we call goal-ontology, to guide users to decompose and refine their initial goals. The difference is we defined rich semantic association (including "is-a", "has-a", etc.) between the father goal and the child goal rather than just subordinate relationship like in [3-7]. Referencing to the goal-ontology as a shared vocabulary on goal concepts, users are more knowledgeable and instructed when choosing sub-goals and therefore feel better experience.

Different from above user-involved goal decomposition, Jian Cao et al [25] integrate the concept of goal-ontology into customized web service models. By defining the relationship among goal concepts (i.e. specialization and decomposition), the general user goals can be easily decomposed into more concrete and fine-grained sub-goals. However, this paper doesn't mention how to create such a goal-ontology and this is the key issue in the ontology-driven goal decomposition problem. Hua Xiao et al present an ontology based automatic goal decomposition approach in [10-13]. The high-level user goals are extended automatically into a set of sub-goals according to a collection of ontologies which can be achieved by dedicated ontology search engines. For the reason that no human interaction is involved, the results of goal decomposition are totally dependent on the quality of pre-chosen ontologies. However, the ontologies searched by ontology search engines are usually too general to guide mashup development. Considering the example discussed in section 1, the initial goal "fast transport" can be extended, for example according to AKT Reference Ontology [14], into sub-goals like "things act on", "receipt agent", "loc@start", "loc@end", "means of transport", which make a radical departure from our expectation by providing redundant sub-goals like "things act on" and missing necessary sub-goals like "get directions", "get traffic volume", etc. Evren Sirin et al also adopt ontology-based approach to deal with automatic goal decomposition in [15]. The difference is that the ontology, which is represented as "domain" in HTN planning problem, is translated from existing OWL-S process definition and therefore more instructive on development than that searched from network by search engine. However, the process-dependent (i.e. "piping") ontologies are not applicable to process-independent (i.e. "wiring") mashup, such as event-triggered mashup mentioned before. Different from above works, in this paper, we will present a historic heuristic based algorithm to mine knowledge and experience from those already developed mashup applications (including both "piping" manner and "wiring" manner) and build up a goal-ontology dedicated to mashup development. Referenced to the goal-ontology, sub-goals, which have higher popularities for example, can then be recommended to the user with a higher probability to be choosen.

3 An Ontology-Driven Approach to Goal Decomposition

Ontology is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts [17]. It is widely used as a shared vocabulary in semantic web, artificial intelligence, etc. In this paper, we utilize it as the knowledge to direct end users to decompose and refine their goals by recommending possible sub-goals and other correlated goals. For example, if a user indicates “fast transport” as his initial goal, “get directions”, “get traffic volume” may be recommended as its sub-goals and “cars rent service” as a possible complement, which is usually accompanied by the transport-related concept.

Ontology-based modeling and reasoning enables automatic and intelligent software development. But at present, the approved united and consistent ontology specific for mashup development is still missing. Manually creation of such a normative ontology is not only time-consuming and expensive, but also error-prone. In this paper, we propose to leverage collective intelligence that web 2.0 encapsulates and advocates, and present a history heuristic based approach to create goal-ontology for mashup development by reversely analyzing existing mashup applications.

Mashup Goal Ontology, in this paper, means the formal representation of knowledge which is used to guide the decomposition and refinement of user goals in mashup development. It can be viewed as a hierarchical goal tree, along which the high level goal can be decomposed into sets of sub-goals. Based on our observation that the architecture of mashup application also presents a hierarchical structure for the reason of nested composition of mashlet, and moreover the concepts of Mashup Goal Ontology (i.e. user goals) can be viewed as functional descriptions of the components of mashup architecture, we define the Mashup Goal Ontology model by referring to the architecture of mashup application. The goal decomposition process is essentially the design procedure of mashup architecture, and the sub-goals are actually the requirements for mashup constructs, namely mashlets.

3.1 Mashup Goal-Ontology Model

In this section, we analyze the architecture of mashup application, based on which we define the Mashup Goal Ontology model for mashup development.

Mashup Architecture. In [18,19], the authors present an enterprise mashup stack like “resource-page-component-mashup” from the point of view of programming. However, as the authors state, the “page” designers should be “characterized by basic programming skills in order to bind the resources to user interfaces”. This is far beyond the capability of novice end users. In this paper, we will look at the mashup architecture from a requirement perspective which may be more acceptable for end users.

As shown in Fig. 1, a mashup application is a collection of mashlets which are loosely coupled through event mechanism. Further, we distinguish two categories of mashlets, infoMashlet and actMashlet, based on their different purpose and logical structures. infoMashlet is a kind of mashlet used to display a group of data. The result data to be displayed and its presentation (i.e. Graphical User Interface) are the most concerned factors for users. Usually, the result data are calculated by a flow of data processing operations on multiple data sources, and the procedure how the data is processed is not cared about by users. Thus, from the requirement respect, infoMashlet is comprised of data items to be displayed and their presentation in our mashup architecture. Correspondingly, actMashlet is a kind of mashlet used to execute a series of interactive actions, such as "making an internet phone call" involves a sequence of actions of "calling" and "hanging up". Each interactive action, such as the "calling" in an internet phone call procedure, is actually implemented by a series of services, e.g. "connecting with the call server", "routing", "creating speech channels", etc. From the users' respect, it is transparent how these services are bound together and how they are implemented in the background. What the users care about is what functions the composed services can provide, i.e. the operations covered by an actMashlet.

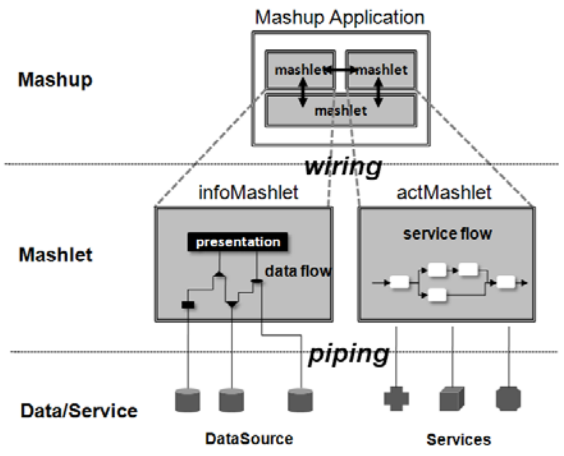


Fig. 1. Architecture of Mashup Application.

Mashup Goal Ontology Model. Referenced to above mashup architecture, we define mashup goal-ontology model as a tree structure shown in Fig. 2.

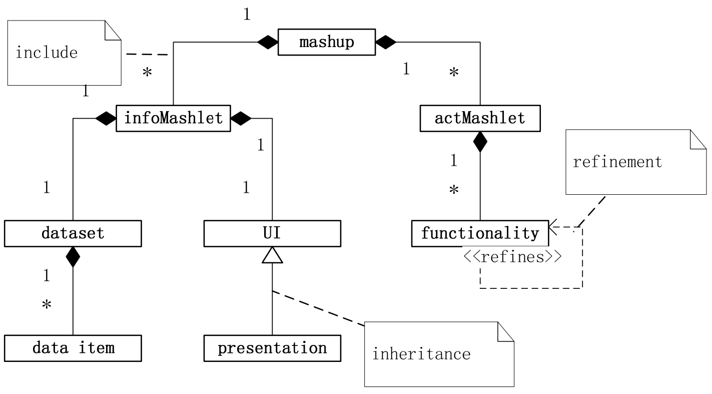


Fig. 2. Mashup Goal Ontology Model.

The concepts of Mashup Goal Ontology are derived from the components of mashup architecture. The hierarchical relationship between the super-concept and the sub-concept means "containing" or "involving". For example, a mashup application may contain 1 to n mashlets which are complementary to each other, while an actMashlet, providing certain functions, may involve 1 to n implementations which are mutually exclusive. Further, we characterize three types of relationships among sub-concepts. For the complementary relationship, we mark it as OR, meaning each sub-concept is optional to be a content of the super-concept and the super-concept is called OR-Concept in this paper. For the coordinative relationship, marked as AND, meaning each sub-concept is indispensable to its super-concept which is called And-Concept. For the mutually exclusive relationship, marked as XOR, meaning the candidate sub-concepts are functionally equivalent or similar and can be replaced with each other. We call their super-concept as an XOR-Concept.

To specify the contribution of each sub-concept to his parent, we label each sub-concept with a weighted value, *W*, indicating its importance and popularity. We assume that one sub-concept contribute more to his parent if it has been appeared in more existing applications. We will discuss it in detail in the following section. To be an exception, the sub-concepts of infoMashlet, "dataset" and "UI", have no weighted value attached for the reason that dataset and UI are the two static constructs of infoMashlet. They will not be instantiated in the model instantiation procedure.

User Goal Definition. User goal can be viewed as an instance of Mashup Goal Ontology model. Based on the model, we define user goal as a set of mashelts, being infoMashlet or actMashlet.

Goal{Mashlet$_i$| i=1…k, Mashelt=infoMashlet|actMashelt}

We define infoMashlet as a tuple of Dataset and UI where Dataset means a set of date items and UI means a set of presentations. "***OR***", "***XOR***", "***AND***" are the three types of relationships between data items and presentations which we have discussed above.

infoMashlet <Dataset, UI>

Dataset{dataitem$_i$ | i=1…n, dataitem = dataitem ***OR*** *dataitem | dataitem* ***XOR*** *dataitem | dataitem* ***AND*** *dataitem}*

UI{presentation$_i$ | i=1…m, presentation=presentation ***XOR*** *presentation}*

We define actMashlet as a set of functionalities. Besides "***XOR***" and "***AND***", relationship of "refinement", marked as "->", is also defined, which means the sub functionality is more specific than its father. For example, "get Traffic Volume" -> "get Traffic Volume of Beijing".

actMashlet{functionality$_i$|i=1…t, functionality=functionality -> functionality | functionality ***XOR*** *functionality | functionality* ***AND*** *functionality}*

An instance of user goal can be described as:

"Fast Transport"{

MapView < "labels of source&destination" ***AND*** *"street names"* ***AND*** *"live traffic", "maps mode"* ***XOR*** *"satellite mode">,*

PlanView { "planning" -> ("time-shortest planning" ***XOR*** *"distance-shortest planning") }*

}

3.2 A History Heuristic Based Approach to Mashup Goal Ontology Creation

Mashup Goal Ontology is the shared knowledge which can be used to guide end users to create their own situational mashup applications. In this section, we present a history heuristic based approach to create Mashup Goal Ontology. We think that the mashup application which has already been implemented and published contains some knowledge that can be reused when addressing the same kind of problems. We analyze each mashup application and parse its core elements out according to the Mashup Goal Ontology model we defined in the last section. That is we translate each mashup application into a Mashup Goal Ontology instance.

For the reason that the Mashup Goal Ontology instances derived from mashup applications may overlap in semantics, we take a semantic merging algorithm on those instances in an iterative manner to form a Mashup Goal Ontology repository for automatic mashup development.

Ontology Merging.

Algorithm: Ontology_Merging

```
Input: <H<>, N>,      //H<>: existing set of ontology instances,   N: new ontology instance to be merged
Output: H'<>,          //H'<>: set of ontology instances after merging
1    for each H in H<>
2        if (Similarity(MashupH, MashupN)>= threshold )   // MashupH and MashupN are similar, then merge them
3            for each infoMashletN
4                for each infoMashletH
5                    if (Similarity(infoMashletH, infoMashletN) >= threshold )
6                        infoMashletH.presentations=infoMashletH.presentations $\cup_{semantic}$ infoMashletN.presentation
7                        infoMashletH.data = infoMashletH.data  $\cup$semantic   infoMashletN.data
8                if no similar infoMashletH exist, infoMashletsH=infoMashletsH  $\cup$ infoMashletN
9            for each actMashletN
10               for each actMashletH
11                   if (SynSet(actMashletH, actMashletN))   // actMashletH and actMashletN are synonymous
12                       actMashletH.weight++
13                   if (Hypernym(actMashletH, actMashletN))   // actMashletH is more generic than actMashletN
14                       actMashletH.functions=actMashletH.functions  $\cup_{semantic}$ actMashletN
15                   if (Hyponym(actMashletH, actMashletN))   // actMashletH is more specific than actMashletN
16                       continue
17                   if (Coordinate(actMashletH, actMashletN))   // actMashletH and actMashletN are coordinate
18                       actMashletH'= actMashletH
19                       actMashletH = GetHypernym(actMashletH, actMashletN)
20                       if (actMashletH'.functions = = null)
21                           actMashletH.functions=actMashletH'  $\cup$  actMashletN
22                       else
23                           actMashletH.functions=actMashletH'.functions  $\cup$semantic actMashletN
24               if no similar actMashletH exist, actMashletsH=actMashletsH  $\cup$actMashletN
25           H'<> = H<>
26       else   H'<> = append(H<>, N)   // append N into H<>
```

The inputs of the algorithm are two ontology instances to merge. *H<>* means the existing set of instances, while *N* means the current instance to be merged. The output is the new merged instance set, marked as *H'<>*. In the body of the algorithm, we take each ontology instance, *H*, in *H<>* to compare with *N*. First, we calculate the semantic similarity between the root nodes of *H* and *N*. If it falls in an acceptable threshold, it means that the goals of mashup *H* and *N* are similar and can be merged together to form a more complex ontology instance. Otherwise, it means *H* and *N* are two unrelated instances and we just put *N* into *H*<> as a new set element without any merging operations.

When merging the two ontology instances *H* and *N*, we first compare their infoMashlet nodes. For the infoMashlets which have high semantic similarities, we merge them by taking a ∪semantic operation on their data and presentation nodes respectively. For the remained infoMashlets of *N* which do not have high semantic similarity with any infoMashlets in *H*, we just put them into the infoMashlet set of *H*. Likewise, we compare actMashlet nodes of *H* and *N* (marked as actMashletH and actMashletN respectively). We characterize four relationships between actMashletH and actMashletN:

- **Synonym**: if they are synonymous, we just need to plus one to the weight of actMashletH node, meaning the contribution of this actMashlet node of *H* is increased by one.
- **Hypernym**: if they are hypernymic, meaning actMashetH is more generic than actMashletN, we join actMashletN into actMashletH as one of its sub-nodes.
- **Hyponym**: if they are hyponymic (i.e. actMashletH shares an "is-a" relationship with actMashletN), we just discard actMashletN for it is too generic to make contributions to refine the actMashlet node of *H*.
- **Coordinator**: if they are coordinate, we shall first calculate the mutual hypernym of both actMashletH and actcMashletN as the new actMashlet node of *H*, and then join actMashletH and actMashletN together as the sub-nodes of the new node.

To calculate the similarity between OR-Concepts and AND-Concepts, we define:

$$\text{Similarity(word1, word2)} = \begin{cases} \alpha/(\alpha+d), & \text{d<threshold} \\ 0, & \text{otherwise} \end{cases} \quad (\mathbf{1})$$
$$d = \text{Distance(word1, word2)} .$$

Distance(word1, word2) is the distance between word1 and word2 in the WordNet [26] architecture. α is an adjustable parameter which we take 1 in this paper.

For the XOR-Concepts, we leverage the open APIs provided by WordNet to determine the semantic relation between ontology concepts, including synonym, hypernym, hyponym and coordinator.

When merging subnodes of two concepts, we define a ∪semantic operation which includes three sub-operations: 1) union set *A* and set *B* to form set *C*, 2) remove synonyms in set *C* and meanwhile 3) modify the weight of remaining elements of set *C*. For example, if $c1 \in C$, $c2 \in C$, *c1* is synonym of *c2*, after ∪semantic operation, only *c1*(or *c2*) is remained in set *C* with its weight plus one, while *c2*(or *c1*) is removed.

3.3 Recommendation-Based Goal Decomposition

Referenced to Mashup Goal Ontology, an initial user goal may be decomposed into a set of sub-goals. To facilitate users to pick out their most interested sub-goals, a recommendation mechanism is used to rank those sub-goals according to their contributions to their parent and push the top-k [22,23] sub-goals to the users.

Specifically, except for above contribution metric, the ranking metric may also involve user preference which can be achieved by either pre-defining or mining through the history of user operations or feedbacks. In the current implementation, we just take the frequency of sub-goal in the existing mashup applications as the ranking metric. Fig. 3 shows the whole procedure of goal decomposition.

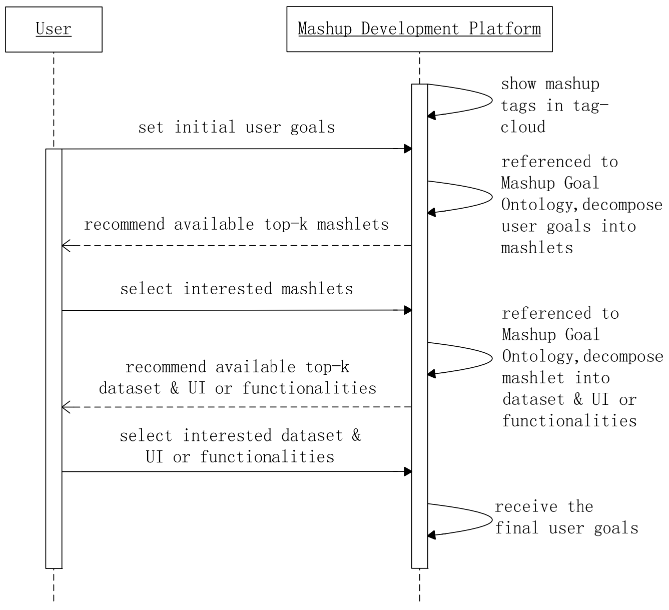

User
Mashup Development Platform
show mashup tags in tag-cloud
set initial user goals
referenced to Mashup Goal Ontology, decompose user goals into mashlets
recommend available top-k mashlets
select interested mashlets
referenced to Mashup Goal Ontology, decompose mashlet into dataset & UI or functionalities
recommend available top-k dataset & UI or functionalities
select interested dataset & UI or functionalities
receive the final user goals

Fig. 3. Procedure of Goal Decomposition.

4 Evaluation

We designed and carried out a set of experiments to evaluate the effectiveness of our Mashup Goal Ontology for decomposing and refining user goals in mashup development.

For the evaluation metric, we adopt precision and recall which are widely used to measure user satisfaction in searching or recommendation field.

$$\text{Precision: } P = (A \cap B)/A \qquad (2)$$

$$\text{Recall: } R = (A \cap B)/B \ .$$

In (2), *A* is the set of sub-goals that are decomposed according to the Mashup Goal Ontology, *B* is the set of sub-goals that the user really required or interested in, *A*∩*B* represents the "hit" sub-goals which are correctly decomposed into. Precision is the ratio of "hit" sub-goals to the total decomposed sub-goals according to the Mashup Goal Ontology. Recall is the ratio of "hit" sub-goals to the total required sub-goals.

We download 6000 mashups and 2500 APIs from ProgrammableWeb.com and divide the mashups into two groups: one is used to construct the Mashup Goal Ontology repository, the other is used to evaluate the performance (i.e. precision and recall) of our Mashup Goal Ontology for decomposing and refining user goals.

Step_1: construct Mashup Goal Ontology repository

Select randomly 5000 mashups and parse out 5000 Mashup Goal Ontology instances. After performing Ontology_Merging algorithm on those ontology instances, we then get a united Mashup Goal Ontology repository, marked as Mashup_Goal_Ontology_5000.

Step_2: evaluate Mashup Goal Ontology on the remained mashups

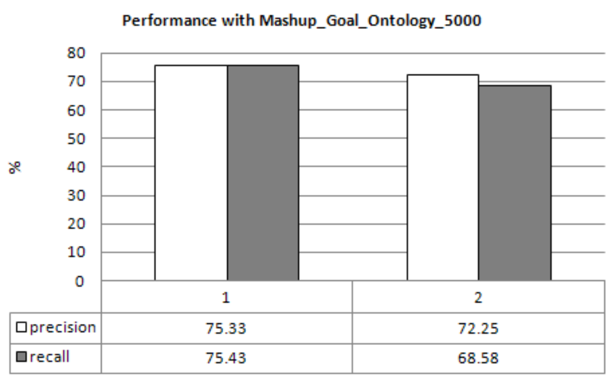
Select randomly 100 mashups in the remained 1000 mashups as the benchmark for evaluating the precision and recall of our decomposition approach, based on the belief that the mashup which has been developed and published represents the real goal of the developer. We parse the benchmark mashup and take its functional description as the user's initial goal. Then we apply our decomposition approach on those initial goals and get a set of sub-goals, i.e. set *A* in (1). Further, we translate the benchmark mashup into a Mashup Goal Ontology instance and take it as the desired user goals, i.e. set *B* in (1). Specifically, to improve the efficiency of the experiment, we will simulate user interactions by clustering algorithm. For the OR-Concept, we cluster its sub-goals in groups by their weights and pick up the group with the highest weight as the sub-goal set that the user may select. For the XOR-Concept, we just need to choose the sub-goal that has the highest weight.

We repeat above steps for twenty times and calculate the mean of precision and recall respectively. The results are shown in Fig. 4-7.

In Fig. 4, Y-axis represents the ratio of precision and recall. X-axis represents the level of decomposition in which 1 means the 1st-level decomposition (i.e. initial user goals decompose into mashlets) and 2 means the 2nd-level decomposition (i.e. mashlets decompose into dataset & UI or functionalities). From the results, we see that after applying our approach, we can get a mean precision (and recall) beyond seventy percent, which means our approach plays a positive effect in guiding the decomposition of user goals. Specifically, we note that the precision and recall of the 2nd-level decomposition are both lower than that of the 1st-level. It can be explained that we simulate user interactions by clustering algorithm in the experiment and the error of 1st-level decomposition is propagated into 2nd-level decomposition.

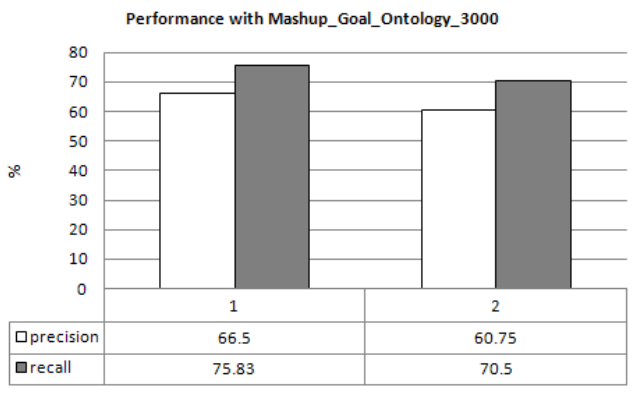
We then change the mashup numbers in step_1 to be 3000, 1000 (the corresponding goal-ontologies are marked as Mashup_Goal_Ontology_3000 and Mashup_Goal_Ontology_1000) and repeat above experiment. We have the following results in Fig. 5 and Fig. 6.

Fig. 7 shows the comparative results with different Mashup Goal Ontology repository. We can see that the precision and recall both reduced with the size of Mashup Goal Ontology repository decreased. The reason is obvious that decreasing Mashup Goal Ontology instances means weakening the knowledge that is used to guide the goal decomposition. This may lead to some goals failed to be decomposed or miss some important sub-goals because of the limited knowledge.



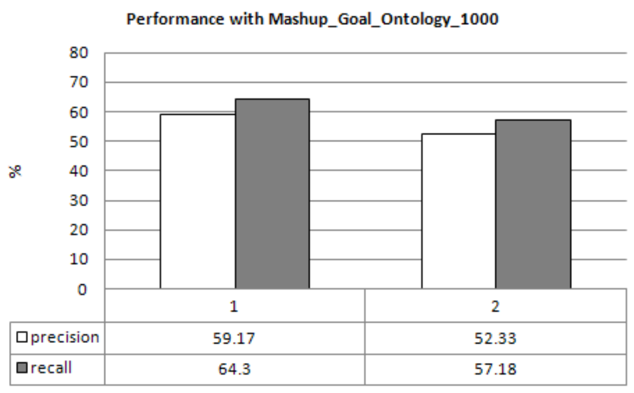
	1	2
□ precision	75.33	72.25
■ recall	75.43	68.58

Fig. 4. Precision and Recall of Decomposition with Mashup_Goal_Ontology_5000.



	1	2
□ precision	66.5	60.75
■ recall	75.83	70.5

Fig. 5. Precision and Recall of Decomposition with Mashup_Goal_Ontology_3000.



	1	2
□ precision	59.17	52.33
■ recall	64.3	57.18

Fig. 6. Precision and Recall of Decomposition with Mashup_Goal_Ontology_1000.

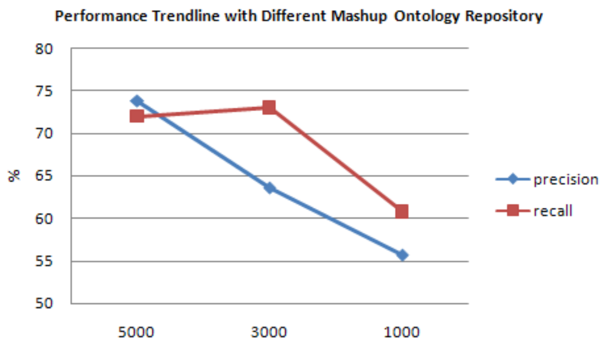


Fig. 7. Tendency of Precision and Recall with Different Mashup Goal Ontology Repository.

5 Conclusion and Future Work

It is a great challenge on how to map general user goals into low-level and concrete mashlets in automatic mashup development. In this paper, we explored a goal decomposition and refinement approach aiming at enhancing the automation of mashup development. We defined a Mashup Goal Ontology model based on which Mashup Goal Ontology instances can be derived. Further, we proposed an ontology merging algorithm to create a union Mashup Goal Ontology repository for automatic decomposition of user goals. Meanwhile, leveraging recommendation methodology, we involve user interactions in the whole decomposition process to adjust the results of auto-decomposition and ensure the decomposition goes in the way that users expect.

Mashup development is a new pattern of End-User Programming. Considering the capability of end users in development, we explore mashup applications from the user requirement perspective and in a coarse-grained manner, e.g. data item, UI and operational functionality that have something to do with user interactions. While for the low-level and non-user-interactive services, e.g. the data operations like filtering, merging, et al., it is outside the discussion of this paper. Composition of these fine-grained mashup resources can be achieved, for example, by intelligent planning approach proposed in [3,8,9].

This paper presents a history heuristic based approach to create Mashup Goal Ontology repository. The quality of the ontology is dependent on the quantity and quality of the existing mashups and mashlets which have been developed and registered in the network, for example, whether the mashup specification is well-defined or whether the mashlet is accessible. Matured mashup community, like ProgrammableWeb.com, plays a fundamental role in our approach.

In this paper, we mainly considered perceptible user goals. In the future, we will take implicit user goals into consideration, which may not be perceived by the user himself but could have a great effect on his current goals, e.g. the user's geographical location, preference, or knowledge background, etc. How to model and measure these kinds of goals is our next work.

Acknowledgments. This work is supported by the National Basic Research Program of China (973 Program) under Grand No. 2009CB320704, the National Natural Science Foundation of China under Grant No. 61170074 and the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grand No. 2010zx01045-001-006-1.

References

1. Wikipedia, http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid), accessed by 03/21/2012.
2. Bouguettaya, A., et al. End-to-End Service Support for Mashups. IEEE Transactions on Services Computing, 2010. 3(3): p. 250-263.
3. Bouillet, E., et al. A tag-based approach for the design and composition of information processing applications. in Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. 2008. Nashville, TN, USA : ACM.
4. Bouillet, E., et al. A Folksonomy-Based Model of Web Services for Discovery and Automatic Composition. in IEEE International Conference on Services Computing, 2008. SCC '08.
5. Bouillet, E., et al. A Faceted Requirements-Driven Approach to Service Design and Composition. in IEEE International Conference on Web Services, 2008. ICWS '08.
6. Ranganathan, A., A. Riabov and O. Udrea. Mashup-based information retrieval for domain experts. in Proceeding of the 18th ACM conference on Information and knowledge management. 2009. Hong Kong, China : ACM.
7. Riabov, A.V., et al. Wishful search: interactive composition of data mashups. in Proceeding of the 17th international conference on World Wide Web. 2008. Beijing, China : ACM.
8. Riabov, A. and Z. Liu. Scalable Planning for Distributed Stream Processing Systems. in Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006. 2006: AAAI.
9. Riabov, A. and Z. Liu. Planning for stream processing systems. in Proceedings of the 20th national conference on Artificial intelligence - Volume 3. 2005. Pittsburgh, Pennsylvania : AAAI Press.
10. Xiao, H., et al. A Framework for Automatically Supporting End-Users in Service Composition. Lecture Notes in Computer Science, 2010.
11. Xiao, H., et al. An Approach for Context-Aware Service Discovery and Recommendation. in Proceedings of the 2010 IEEE International Conference on Web Services. 2010: IEEE Computer Society
12. Hua, X., et al. An automatic approach for ontology-driven service composition. in Proceedings of the 2009 IEEE International Conference on Service-Oriented Computing and Applications. 2009.
13. Xiao, H., et al. , Personalized Service Discovery and Composition, in Pre-proceedings of SITCON: The CAS / NSERC Strategic Workshop in Smart Internet Technologies. 2009: Canada.
14. Advanced Knowledge Technologies, http://www.aktors.org/ontology/, accessed by 12/23/2010

15. Sirin, E., et al. HTN planning for Web Service composition using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web, 2004. 1(4): p. 377-396.
16. Wu, D., et al. Automating DAML-S Web Services Composition Using SHOP2. Lecture Notes in Computer Science, 2003. 2870: p. 195-210.
17. Wikipedia, http://en.wikipedia.org/wiki/Ontology_(information_science), accessed by 04/10/2012
18. Hoyer, V., et al. The FAST Platform: An Open and Semantically-Enriched Platform for Designing Multi-channel and Enterprise-Class Gadgets. in Proceedings of the 7th International Joint Conference on Service-Oriented Computing. 2009. Stockholm: Springer-Verlag.
19. Hoyer, V., et al. Enterprise Mashups: Design Principles towards the Long Tail of User Needs. in IEEE International Conference on Services Computing. 2008.
20. Rodriguez, M.A. and M.J. Egenhofer, Determining semantic similarity among entity classes from different ontologies. IEEE Transactions on Knowledge and Data Engineering, 2003. 15(2): p. 442- 456.
21. http://www.programmableweb.com/
22. Akbarinia, R., E. Pacitti and P. Valduriez. Best position algorithms for top-k queries. in Proceedings of the 33rd international conference on Very large data bases. 2007. Vienna, Austria: VLDB Endowment.
23. Wang, J., et al. TFP: an efficient algorithm for mining top-k frequent closed itemsets. IEEE Transactions on Knowledge and Data Engineering, 2005. 17(5): p. 652 – 663.
24. Wu, J., et al. Web Service Discovery Based on Ontology and Similarity of Words. CHINESE JOURNAL OF COMPUTERS. 2005, 28(4): p. 595-602.
25. Cao, J., et al.. A Goal Driven and Process Reuse Based Web Service Customization Model. CHINESE JOURNAL OF COMPUTERS. 2005, 28(4): p. 721-730.
26. WordNet, http://wordnet.princeton.edu/wordnet/