Assisted Service Composition for End Users

Nikolay Mehandjiev, Freddy Lécué, Usman Wajid and Abdallah Namoun
Centre for Service Research, University of Manchester
Booth Street East, Manchester, UK
{(firstname.lastname)@manchester.ac.uk}

Abstract—Involving people who do not have programming background in assembling and tailoring service-based applications promises to open up access to the creativity of millions of users. An increasing number of development environments aim to do this by offering drag-and-drop visual representations connecting different service components into an assembly. In contrast to the majority of these, we did not start with the technology but with the users - service producers and consumers, and studied the core issues which should be resolved before people who are not programmers can start to assemble services into meaningful applications, over and above the presentation-level integration offered by current mash-up environments. The result is an assisted approach to service composition for end users, which uses semantic technologies to shield users from the irrelevant complexity of service technology and from the need to manually resolve dependencies between services. The approach was evaluated by a focus group of non-technical users, who ranked it highly and provided valuable suggestions for further improvements and supporting features.

I. INTRODUCTION

Allowing general users to assemble and tailor their own service-based applications is seen as a key step to wide uptake of service-oriented computing. As we have seen with the appearance of wikis and blogs on the WWW, unlocking the creativity of millions of users can bring about innovation and open up new application niches for service-oriented computing, increasing its impact to a global scale.

Many development environments have appeared aiming to facilitate the consumption and assembly of software services, ranging from the simple side-by-side mashups of iGoogle and myYahoo through the RSS-based Yahoo!Pipes to the sophisticated service environments reported in Section II. The majority of the latter use diagrammatic languages designed to represent technical dependency details. Published evaluations of these environments focus on demonstrating the feasibility of building service-based applications, and on validating technology-led user interface design decisions.

In contrast, we followed a user-centric approach by starting with our target service producers and consumers. We studied the core issues which should be resolved before people without programming background can begin to assemble services into meaningful applications, over and above the presentation-level integration offered by current mash-up environments. Our results, reported in Section III of this paper, suggest we need to reduce learning costs by making the composition as transparent as possible, hide any technical details which are not relevant to the task of the user, and provide immediate feedback in respect to any design decisions by end users. We also gained insight into our users’ mental models regarding services and service composition.

These findings motivate a transparent, task-oriented approach to service composition, where background processing takes over mundane technical details such as aligning service inputs and outputs. Such an approach should allow sharing “best-practice” templates of successful service compositions within a community of users, and make different degrees of customisation available for users with different skills and technology aptitudes. In this paper we propose such an approach, which we call assisted service composition for end users. The approach, described in detail in Section IV, comprises the following two contributions:

1) A template-based process for user composed services, which involves three stages: assisted composition, template adaptation and learning (generalising user-adapted innovative applications into templates). In this paper we focus on the stage of assisted composition since it targets the widest audience of general users.

2) A semantic technique of service alignment, alleviating the need for data integration between constituent services, and shielding users from inter-service dependencies and the technical complexity of service technology.

In Section V, we report on the evaluation of our approach, for which we used a focus group with 13 non-technical users. We presented an early interactive prototype of the approach and gathered detailed formative feedback which will allow us to build a more effective environment for user composition of services.

II. RELATED RESEARCH

A. Service composition environments

A number of service composition environments exist, ranging from the simple side-by-side presentation of independent widgets such as iGoogle, through increasingly sophisticated “mashup” environments allowing users to mix information produced by different services (e.g. Yahoo!Pipes\(^1\) and Mash-Maker [1]) to fully-fledged environments for composing stateful services using visual wiring between the services [2], [3], [4], [5]. Indeed, bpmn.org reports 62 tools for wiring services together using the Business Process Modelling Notation (BPMN). Most service mashup systems provide a single tool, except [6], which uses two different tools for the different stages of the mashup. Only a few of these systems have been

\(^1\)http://pipes.yahoo.com/pipes/
evaluated in terms of usability and cognitive effectiveness. We focus on these criteria in the next section.

**B. User-centric Approaches to Service Composition**

The academic field of End User Development (EUD) [7], [8] takes a user-centric approach to creating tools which can enable non-technical users to develop sophisticated applications. Main EUD results include theoretical models such as the tradeoff-based “Attention Investment Model” [9] and the lifecycle model of Meta-Design [10]. There are also a number of well-known practical successes such as spreadsheets and database form painters.

Service-focused work in this field, however, is surprisingly rare, and focused on professional programmers [11], or on web mashups rather than fully fledged service composition [12]. An exemplary user-driven design process is reported in [13], yet it is focused on conventional web applications rather than web services.

Interesting conclusions in this field include the need for supporting end users by hiding irrelevant technical details and complexity from them, providing them instead with task-oriented languages [14]; and the view of end user environments as a medium of continuous collaboration between end users and developers, resulting in the evolution of the environment itself to reflect evolving user skills and requirements [15]. The concept of “Power Users” (technology-savvy end users) as a third side to this collaboration is also important since they are often leaders of user-driven application innovation. Our studies, reported in Section III, confirmed the validity of some of these conclusions for the domain of user-driven service composition.

**C. Automating service composition**

Taken to its extreme, the idea of supporting end users in service composition would translate into the aim of fully automating the composition. Indeed, many AI-inspired approaches [16], [17], [18], [19] address the issue of automated web service composition. Full automation, however, even if it were feasible, would miss the chance for user-led innovation and fine-tuning services to user needs.

Only a handful of approaches have the users in the driving seat and support them by resolving technical details such as data integration and other service dependencies. For example, Carlson et al. [20] introduce an approach where users can drag a service onto a canvas, and this narrows down all discovered services to only those which have compatible inputs and outputs with the service thus selected by the user. A more structured support of the composition process is provided by the composition tool reported in [21], where the process is step-by-step, guided by the tool. Both these approaches use semantic tagging of services, and limited semantic reasoning with the data thus available. However, neither of them uses templates and thus cannot support reuse of composition knowledge.

Semantic reasoning underpins such selection of compatible services. This can use basic semantic matching types [22], [23], the difference operator [24], [25] or Concept Abduction [26]. Different approaches have different performance and scalability, and we need to consider the correct approach based on the expected scale of compositions and number of candidate services.

**III. ISSUES IMPEDING END USER COMPOSITION OF SERVICES**

Developing a successful end user composition tool required insight into the mental models of services and service composition held by our target user groups, and understanding of the main issues which may impede their uptake of service composition. To this end, we held five focus groups with a total of 64 users of mixed background - technical and non-technical. This section reports the main issues raised in our focus groups regarding service composition by users from non-programming background.

**A. Solution Complexity and Reuse of Example Templates**

The examples used were extrapolated by participants to cover larger-scale compositions, with “2000 services for each task”. This brings to the fore the need for recommender systems to rank the most suitable services for your circumstances.

More complex compositions bring the limitation that not every user will know the overall process to be followed because of its complexity, and dependency on other processes. Some users also felt it was not fully natural for them to “think in sequence”. This motivates the need for the system to suggest which order should services be arranged in, and to facilitate sharing of process knowledge between users, focused on the tasks to be achieved. They liked the idea of customising standard applications to tune them to personal needs and circumstances, suggesting the need for example templates.

Another dimension is the evolution of both the task and the available service solutions: any composition tool will need to accommodate new tasks and new services appearing every day.

**B. Technology Complexity and Data Alignment**

A number of participants pointed out that explicit representation of the flow of data between services makes it “difficult to understand” and hard to figure out “what is going on”, because of the multiplicity of links between services. Explicitly representing the data was felt to create “spaghetti” diagrams, where it is “not easy to put a new service”, and the data dependency was not felt to be “natural” for some of the participants.

At the same time, several participants felt that control-flow-based representation “lacks the level of detail that is required to make it work”, and that this creates potential for errors in terms of data mismatch between services thus connected. Indeed, a number of users pointed out the different standards and formats of data (XML versus text for example), and the potential for error this would create. Interestingly, it was the technical users who highlighted the dangers of missing information about the data being exchanged in the control flow representation. Most non-technical users ranked highly
control flow because of its apparent simplicity and disliked the complexity which stems from explicitly representing data flow between services.

Other relevant discussion points covered the need to specify the semantics of the services using standard semantic notations. This, however, was expected to bring complexity to service descriptions, so we also need to have different ways of representing the composition to people with different skill levels. We need the tool to be “flexible enough to allow composition without worrying about low-level details”, whilst we need some “expert mode” for people with technical skills. The tool should support the users by validating the services chosen, ensuring there are no mistakes.

IV. ASSISTED COMPOSITION APPROACH

Given an abstract description of any service composition in a template, such as in the Student registration process above, we propose an approach which can support end users in creating actual service compositions, even when these end users are not technically savvy. Those who are more happy to engage with software, often called “power users”, will be able to further customise such compositions by changing the abstract templates, creating innovative variations of standard service composition tasks. These innovations would then be generalised into reusable templates by software developers, thus ensuring the growth of the overall system, learning from generalised into reusable templates by software developers, service composition tasks. These innovations would then be generalised into reusable templates by software developers, thus ensuring the growth of the overall system, learning from these innovative compositions and offering them for reuse by people with technical skills. The tool should support the users by validating the services chosen, ensuring there are no mistakes.

End users should be shielded from the technical details of such service assembly, so we hide dependencies (control and data) between tasks within the template processes. These aspects are instead considered behind the scene using semantic reasoning.

After a short motivating scenario, this section describes in further detail the template-based aspects of our approach; then defines the formal model of semantic connections between services, and finally presents the process of assisted composition.

A. Motivating Scenario

The following scenario is one of the several we used within SOA4All. It targets the arrival of an overseas student to the UK. Students search for suitable universities and register for a course upon arrival. They use the acceptance letter to open a bank account and submit tax exemption letters. The bank account is then used to set up payment for University fees. This process comprises different steps (i.e., tasks in our work) some of which are inter-connected. For instance, here is a list of tasks required to achieve student registration:

- **SearchForAUniversity** that returns some **Universities** and their descriptions (e.g., **UniversityID**, **Name**, **Postcode**, **Course** and **Fees**) given a **PostCode** as a location and some subjects of courses **Subject**;
- **RegisterForACourseInUK** that returns an **AcceptanceLetter** and a **StudentID** given a **Person** and a **UKUniversity**;
- **OpenBankAccount** that returns a **BankAccount** given a **Person** and an **AcceptanceLetter**;

All input and output tasks parameters above refer to concepts of a domain ontology, and example portion of which is shown in Fig. 2 in ALE.

---

**Fig. 1. Lifecycle of user-developed service applications**

**Fig. 2. Part of an ALE TBox.**

**B. Template-Based Service Composition**

An intuitive view to service composition would see it as aiming to satisfy the need for a (non-existing) service by bringing together existing ones. This integration can be done manually, yet this would involve the alignment of numerous inputs and outputs, considering a number of pre- and post-conditions, and dealing with other technical issues of grounding, etc, which are clearly outside of the skills and interests of our end users. Since we allow end users to tune a composition to their own needs, we do not need to have complete automation “from scratch” using program synthesis and AI planning techniques [19].

Instead we opt to reuse composition knowledge and provide a starting template for the users. This composition knowledge
can be based on past successful compositions, and can be seeded by formalising domain-specific knowledge about how the problem addressed by the sought service would decompose into sub-problems [27], and task-specific knowledge about the core types of information processing activities [28].

Example 1: (Template-Based Service Composition)

Fig. 3 presents the template-based composition view of the motivating scenario in Section IV-A. Tasks (or “service slots”) are designed along rectangles while simple arrows are used to model a partial order on these services (i.e., its control flow). Dashed arrows refers to data-flow description as possible interdependencies between tasks.

Here we focus on the stage of template instantiation [29], [30], where we need to allocate a specific service for each of the generic “service slots” in the template, using knowledge about the data connections and pre- and post-conditions of services. This is the ‘Assisted Composition’ arrow in Fig. 1, the overall approach also includes the stages of template adaptation where power users can create innovative solutions, and generalisation, or converting the useful innovative adaptations into new templates, helping the library of templates grow.

The focus of supporting end users means that, contrary to existing work in the area (e.g. [31], [32]), we aim to automate only three aspects of the process: selecting suitable services for each task, ranking them according to user profile and working out compatibilities between services in terms of data flow, pre- and post-conditions. The users are involved in selecting one or more services from the shortlist for each task, according to their preferences.

In this paper we focus only on the aspects of selecting a set of appropriate service candidates for each task, and on working out compatibilities between services in terms of data flow in order to show to the end user the consequences of them selecting a given instance. We use semantic reasoning for both aspects, for example once a user selects a service \( s_i \) for task \( T_i \), we use semantic reasoning to tag as eligible for further selection only those service candidates for the other tasks in the template which are compatible with \( s_i \). Before providing further details in Section IV-D, we need to describe the semantic reasoning taking place behind the scene.

C. Semantic Connections of Services

Using tasks specifications of inputs, outputs, pre- and post-conditions of templates, we should be able to infer additional dependencies between tasks, for example we can infer data flow dependencies between tasks using their input and output specifications. In the following we present such dependencies as semantic links [32] between services. Then we describe our semantic-link-based composition model.

1) Semantic Links:: Since input and output parameters of semantic web services are specified using concepts from a common ontology \(^2\) or Terminology \( T \) (an example of such is given in Fig. 2), retrieving links between output parameters \( \text{Out}_j \) of services \( s_i \) and input parameters \( \text{In}_j \) of other services \( s_j \) could be achieved by using a DL reasoner such as Fact++ \(^3\) [33]. Such a link, also known as semantic link [32] \( s_{i,j} \) (Fig. 4) between two functional parameters of \( s_i \) and \( s_j \) is formalized as

\[
\langle s_i, \text{Sim}_T(\text{Out}_s_i, \text{In}_s_j), s_j \rangle
\]

Thereby \( s_i \) and \( s_j \) are partially linked according to a matching function \( \text{Sim}_T \). This function expresses which matching type is employed to chain services. The range of \( \text{Sim}_T \) is reduced to the four well known matching type introduced by [22] and the extra type Intersection [23]:

- **Exact** If the output parameter \( \text{Out}_s_i \) of \( s_i \) and the input parameter \( \text{In}_s_j \) of \( s_j \) are equivalent; formally, \( T \models \text{Out}_s_i \equiv \text{In}_s_j \).
- **Plugin** If \( \text{Out}_s_i \) is sub-concept of \( \text{In}_s_j \); formally, \( T \models \text{Out}_s_i \sqsubseteq \text{In}_s_j \).
- **Subsume** If \( \text{Out}_s_i \) is super-concept of \( \text{In}_s_j \); formally, \( T \models \text{In}_s_j \sqsubseteq \text{Out}_s_i \).
- **Intersection** If the intersection of \( \text{Out}_s_i \) and \( \text{In}_s_j \) is satisfiable; formally, \( T \models \text{Out}_s_i \sqcap \text{In}_s_j \models \bot \).
- **Disjoint** If \( \text{Out}_s_i \) and \( \text{In}_s_j \) are incompatible i.e., \( T \models \text{Out}_s_i \sqcap \text{In}_s_j \models \bot \).

In the same way as semantic links \( s_{i,j} \) between web services \( s_i \) and \( s_j \), we define abstract semantic links \( s_{i,j}^A \) between tasks \( T_1 \) and \( T_2 \).

Example 2: (Semantic Link & Subsume Matching Type)

Suppose \( T_1 \) and \( T_2 \) are respectively tasks related to SearchForUniversity and RegisterForACourseInUK in Fig. 4 (right part) of the motivating scenario in Section IV-A. In such a case, the output parameter University of \( T_1 \) is semantically linked to the input parameter UniversityUK of \( T_2 \). According to the example ontology in Fig. 2, this abstract semantic link \( s_{1,2}^A \) is valued by a Subsume matching type since University \( \equiv \) UniversityUK.

\[
\text{Semantic Link } s_{1,2}^A
\]

Fig. 4. A Semantic Link \( s_{i,j} \) and its Illustration on the Motivating Scenario.

\(^2\)Distributed ontologies are not considered here but are largely independent of the problem addressed in this work.

\(^3\)http://owl.man.ac.uk/factplusplus/
2) Semantic Link Composition Model:: Here, we aggregate the concept of web service composition and semantic link in a same model. Therefore the process model of web service composition and its semantic links is specified by a directed graph which has the web service specifications $s_i$ as its nodes, and the semantic links $s_{i,j}$ (data dependencies) as its edges. In the same way a template-based composition, pre-computed for instance by template-based and parametric-design-based approaches [29], [30], has the tasks specifications $T_i$ as its nodes, and abstract semantic links $s_{i,j}^A$ as its edges.

D. Helping Users Choose Services through Semantic Reasoning

Given a template-based composition, semantic descriptions of their tasks and candidate services, we help users to instantiate templates. The semantic link composition models which provide the data flow in the composed process are automatically inferred from the DL descriptions of services parameters, and, although remaining hidden from the end users, used to provide feedback to them about their selection decisions. Once a user selects a service, the tool will gray out all service candidates which are incompatible, and highlight the compatible ones. As illustrated in Fig. 5, our abstract visualisation hides all details related to control and data flow in the composition, and deals with them in the background.

Example 3: (Abstract Visualisation of Composition)

Fig. 5 illustrates a template-based composition where the user has selected a goal from the taxonomy on the left panel. All related details to data and control flow are abstracted away, and end-users could simply interpret compositions as a list of tasks (first row in Fig. 5) wherein each task could be instantiated by services (columns in Fig. 5).

The user will proceed to select any service in any column. This selection step assigns the selected service(s) for the considered task. The system reduces the list of candidate services for each task to those which are compatible with the selection and gets back to the users with (only) services that could be assigned to other tasks. This reduction is based on both the previous selection and how the selected services can be semantically linked to candidate services of other tasks. This is repeated until each task is assigned to a service.

Example 4: (Assisted Composition from the End-User Perspective)

Fig. 5 illustrates the instantiation procedure of template after the selection of service $s_1$ for task SearchForAUniversity. Services CollegeApps and $s_2$ of task RegisterForACourseInUK are highlighted (in blue) because of (semantic) compatible data flow (Section IV-C) between them and $s_1$, while GradeSavers (in grey) is not because of its incompatibility with $s_1$.

During each step of the template instantiation, the end-user can backtrack and even manually remove some services from any candidates’ list.

2) From the Back-end Perspective: Once the template is selected by the end-user, our system aims to discover candidate services that could be assigned to tasks of the template. Note that all services and templates are annotated with goals, pre- and post-conditions, and input- and output-parameter types. The addition of goals as a separate tagging element allows us to estimate their semantic proximity and differentiate between tasks and services having same inputs and outputs but achieving different things.

A task $T$ of a template can be instantiated by a service $s$ if and only if:

1) The service $s$ achieves the same goal as $T$, assuming an ontology of goals [34].
2) The pre-conditions of $s$ are implied by the pre-conditions of $T$.
3) The post-conditions of $s$ imply the post-conditions of $T$.
4) The matching type between the input specification $In_T$ of $T$ and the input specification $In_s$ of $s$ i.e., $Sim_T(In_T; In_s)$ is $	ext{PlugIn}$. $s$
5) The matching type between the output specification $Out_s$ of $s$ and the output specification $Out_T$ of $T$ i.e., $Sim_T(Out_s, Out_T)$ is $	ext{PlugIn}$. $s$

Conditions (1) to (3) above ensure the candidate service $s$ has the desired effect of the target task $T$, whilst conditions (4) and (5) ensure the semantic (functional) fit between the candidate service and the target task. Condition (4) ensures that all the data which can be passed onto $T$ can be processed by $s$. Condition (5) ensures that the output of $s$ fits within the output specifications of $T$.

Once a service is selected by the end-user, our system retrieves its semantic descriptions and computes all its potential incoming and outgoing semantic links with services of other tasks. The computation is based on the abstract semantic links pre-defined in the template and the actual description of semantic web services. Services can be then linked with
many services depending on the data flow description of the composition. As previously mentioned, only services linked with a semantic link of value Exact or Plugin are consired for robustness reasons. Therefore, these services are highlighted, others are greyed out in the abstract visualisation of the composition.

Example 5: (Assisted Composition from the Back-end Perspective)
According to Example 4 and Fig. 5, $s_1$ has been selected to achieved task Search-ForAUUniversity. Our system dynamically reduces the candidates’ list of other tasks such as RegisterForACourseInUK or OpenBankAccount depending on quality of semantic links between services. For instance, the service GradeSavers (for RegisterForACourseInUK task) is discarded because $T \not\sqsubseteq Out_Sear-chForAUuni-versity \sqcap In_RegisterForACourseInUK \sqsubseteq \bot$ while the service $s_2$ is highlighted because $T \models Out_Sear-chForAUuni-versity \sqcap In_RegisterForACour-seInUK$.

In our approach, the user can assign more than one service to a task, implying parallel execution of services from the back-end perspective. Therefore, the control flow of the template can be even modified on the fly, by adding new parallel branches. Such a modification is transparent to the end user, who are not interested in interacting with real control flow model of composition. Once the user has assigned services to every task of the template-based composition, the instantiation procedure is complete. Then, the composition is ready to be deployed and executed according to the control and data flow information automatically elaborated respectively by the template description and the semantic links. Fig. 6 depicts the final composition we obtain in our motivating scenario i.e., services in black are used to achieve the composition in Fig. 3.

![Fig. 6. Final Composition.](image)

V. EVALUATION
To test the proposed approach, we have conducted a focus group study at the Centre for Service Research with 13 students who are not programmers, aiming to explore the degree to which they accept the approach. We also aimed to discover our user’s view about the merits and potential problems with the approach, and gather suggestions for further improvements. The participants were shown a 20-minute presentation introducing the concept of Service Composition and the vision of SOA4All project. They were then split into three groups, each attending a 15 minute demonstration of an early interactive prototype of the assisted composition platform, and encouraged to ask clarifying questions about the operation of the tool. After they felt familiar with the tool, they were asked to rate the perceived usability of the assisted composition approach and answer three open-ended questions (the merits of assisted service composition, problems arising from assisted composition, and possible ways to overcome these issues). The session concluded with a group discussion of assisted service composition to enrich the collected data and elicit further comments and ideas.

Overall, our users’ interest in the “composition of service-based applications by end users” is rooted in the hope that it will empower them to perform tasks more effectively, save time, and tailor applications according to their own needs. In regards to assisted service composition, Table I shows that users were highly satisfied with the approach (an average rating 4.15 on a 5-point Likert scale where 1=disagree and 5=agree). Usability measures such as easy to use, easy to learn, easy to navigate, and convenient to use all received high ratings. On the other hand, questions “reliable, makes no mistakes, and efficient to use” received lower rating of 3.69 demonstrating low levels of user trust of the system in performing the appropriate actions. Users seem to fear that the system might perform wrong actions and decisions; therefore enabling the right level of user control is a necessity in such system-driven platforms. No statistical differences were evident apart from the question “makes no mistakes” which was rated significantly lower than all other measures except “reliable” (Paired Samples test, $P \leq .01$) emphasising the earlier conclusion.

The analysis of the comments and opinions generated by our users individually and in groups revealed high user satisfaction and appraisal for the assisted service composition approach. According to users the main advantages of this approach are its ease-of-use, uncomplicated interface, time saving, suitability for users with no computing background and expertise, ability to solve problems, and compatibility of services. Users appreciated the feature of “showing compatible
services by single mouse clicks” since this will resolve dependency problems between services and enable users to select desired services without running into incompatibility issues and software crashes.

In regard to the associated problems and negative features of assisted service composition users complained about reliability of the produced applications and were worried about the privacy and security of their personal information especially when using services which might be compromised by hackers and thus leak sensitive data such as their bank details. Another concern was related to whether the assisted composition would be able to cope with the increasing number of services and the changes affecting these services in the future within a fast-evolving world of Internet. A further drawback highlighted by users was that the selection of one service usually precludes the selection of other services which might be the services one wants to use for their desired applications, thus ‘highlighting compatible services’ is a double-edged sword. Users also feared that the presented services might be only those which are sponsored and supported by big companies whereas other less famous services get excluded or shown down the list; thus biasing users towards utilising specific services. Understanding the meaning and purpose of each service could also be problematic when users want to find their target services; the current information provided for each service is insufficient to enable users to make sound selection decisions. Users were also worried that the system could make mistakes which they would not be able to understand or locate in order to solve. It is worth noting that one user reported that this approach is too complex and requires training before one can use it, as well error-prone and restrictive.

When asked to recommend ways to overcome the above issues users emphasised the need to take into consideration the following recommendations:

R1 Introduce a special mechanism to enable the selection of incompatible services in the same composition. Indeed, in our previous work [31] we demonstrate how service providers can be involved in automatic negotiation of compatibility constraints between services when the links between them are not robust [35]. In our future work we will merge these two strands of work together.

R2 Provide more details about the available services and explanation to what each service does and how it works. In addition the users confirmed the need to include the recommendation system functionality in our tool, by requesting suggestions as to which service to use for each task, saying this could be based on users’ rating and comments about their experience in using specific software services.

R3 The assisted composition environment should be able to automate repetitive tasks and reuse information whenever possible.

R4 Allow users to modify the templates by removing optional tasks from the composition, or even by rearranging the execution flow of the services. It was felt these changes should use the current user-facing representation as much as possible (tick-boxes for each task, direct manipulation of task columns, etc.)

R5 Show interdependency between selected services and underlying requirements in order to enhance user understanding of the composition process and thus engage them more effectively into developing service-based applications.

R6 Improve the overall look and feel of the assisted service composition platform.

VI. CONCLUSION

Most of the recommendations R1 to R6 listed above will shape the final stage of our tool development, yet some are more far-reaching and provide valuable suggestions for further work. The work on the development of the supporting tool will be followed by a set of controlled observational studies of people with different levels of programming skills. The results from these experiments will also feed into a set of recommendations for future work in the area.

Together with the quantitative feedback obtained, the discussions and recommendations of our focus group evaluation demonstrate that the 13 non-technical end users understood the principles of the assisted composition approach, and were enthused about it. Following our earlier work [36], [37], we believe that this motivation is partially due to the benefits expected from service composition in terms of applications fine-tuned to the user needs, and partially due to the reduction of the perceived learning costs. The reduction in learning costs is attributed to our two main contributions presented in this paper: the approach of hiding technical complexity using semantic reasoning, and the reuse possible by the template-based development process.

REFERENCES


