Expert Systems with Applications 42 (2015) 1028–1038

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Consumerized and peer-tutored service composition

Klemo Vladimir *, Ivan Budiselić, Siniša Srbljić

University of Zagreb, Faculty of Electrical Engineering and Computing, Consumer Computing Lab, Unska 3, 10000 Zagreb, Croatia

ARTICLE INFO

Article history:
Available online 28 September 2014

Keywords:
Service composition
Peer tutoring
Knowledge sharing
Consumerization of service composition tools
Expertise location
Social matching

ABSTRACT

With continued development towards the Internet of Things, services are making their way from enterprise solutions to our offices and homes. This process is a major driving force in consumerization of IT, because sustainable application development at this scale will not be possible without direct involvement and innovation from consumers themselves. In this paper, we present our work on consumerization of service composition tools. First, we describe how consumer-facing services can be presented in a usable and intuitive way. Then, combining social computing with machine intelligence, we define a recommender system that supports consumers in sharing their knowledge and creativity in peer-tutored service composition, thus empowering consumers to create their own applications. This system recommends consumers with the required service composition knowledge based on mining procedural knowledge stored in previously defined compositions. Once such a group of consumers is identified, social computing tools are used to allow them to share this knowledge with their peers. To demonstrate the effectiveness of this peer-tutored service composition model, we performed consumer satisfaction studies on our consumerized service composition tool Geppeto, which we extended with the described recommender system. Results show significant improvements in service composition in terms of performance and quality of experience.

1. Introduction

Over the last couple of years, the Web underwent a rapid process of transition from a primarily passive consumer medium to an active collaborative environment. The success of Web 2.0, an umbrella term for a series of interventions and developments in digital networking technologies we all use on a daily basis, clearly shows the way for future development. One of the most important dimensions of this new Web is the notion of commons-based peer production, as Yochai Benkler calls this paradigm shift (Benkler & Nissenbaum, 2006). In this kind of communal production, work is jointly owned and accessed by its participants who operate as peers without need for a hierarchical organization of collaboration. Since the number of Web consumers is growing rapidly and with proliferation of ubiquitous mobile devices, it is reasonable to expect that this trend of online collaboration will evolve to not only production of data and information, such as in Wikipedia, but also to production of more advanced forms of creative work like software.

The creative force of all Web consumers, called cognitive surplus by Shirky (2010), can be utilized for creation, customization and automation of software artifacts. Today, programming environments for Web consumers are mostly focused on building situational, on-the-fly applications by combining existing services. For example, web mashups are applications generated by combining content, presentation, or application functionality from disparate Web sources (Yu, Benatallah, Casati, & Daniel, 2008). This process of content combination, or remixing, is usually carried out using some kind of GUI-oriented methodology, thus circumventing a traditional textual programming interface. For instance, Yahoo!Pipes,[1] which is one of the most popular mashup editors, allows consumers to process and remix data by visually connecting various modules.

While Yahoo!Pipes focuses solely on data flow, the consumer programming tool Geppeto[2] developed in our research group extends this paradigm beyond data flow, enabling consumers to define service compositions with control flow, event flow, temporal dependencies, location awareness, communication and synchronization. In the near future, tools like Geppeto will allow consumers to create large sets of service compositions and expand community composition knowledge.

Based on current experiences in software engineering, it can be concluded that building communicating, synchronized, and

* Corresponding author.
E-mail addresses: klemo.vladimir@fer.hr (K. Vladimir), ivan.budiselic@fer.hr (I. Budiselić), sinisa.srbljic@fer.hr (S. Srbljić).

[1] http://pipes.yahoo.com
[2] http://shadowfax.zemris.fer.hr:8080/geppeto/index.html

http://dx.doi.org/10.1016/j.eswa.2014.09.033
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

distributed service compositions that are event-, time-, and location-driven will never be easy. To address this inherent difficulty, *assistants* that help consumers in service composition, from finding and understanding component services to wiring them together, are a cornerstone of consumer computing. In this paper, we focus on such an assistant for *peer-tutored service composition.*

A prerequisite for effective peer tutoring, which aims to enable ad hoc, dynamic, and problem oriented gathering of qualified consumers, is tutor identification. A good candidate tutor should possess tacit knowledge about the services related to a problem and how they can be composed. However, people are often not fully aware of their tacit knowledge, or how this knowledge can be valuable to other consumers and how it correlates to collective knowledge of the larger community. Our research presented in this paper is based on the following hypothesis: Since procedural knowledge encoded in a service composition created by a consumer is derived from that consumer's tacit knowledge, consumers with the required tacit knowledge can be identified through analysis of the compositions they had created.

The process of consumerized service composition using a peer-tutor recommender is shown in Fig. 1. A key object used in peer-tutor identification is the *partial service composition* which a consumer created before encountering a challenge and asking for a tutor. While incomplete, this partial composition encodes parts of the consumer's tacit knowledge about the problem that is being solved. Furthermore, for many problem domains, it is likely that other consumers have already created compositions in that domain. These compositions are stored and analyzed, and are the basis for tutor identification. Specifically, the partial service composition created by the consumer asking for a tutor is compared to other compositions stored in the composition database using machine intelligence techniques. Authors of similar compositions are identified as potential tutors as they are likely to have the required tacit knowledge to help solve the problem. Once potential tutors are identified, social computing tools are used for peer tutoring with the goal of sharing service composition knowledge in the consumer community.

The rest of the paper is organized as follows. Section 2 gives an overview of related work in peer tutoring and tutor recommendation. In Section 3, we review our research in consumerization of

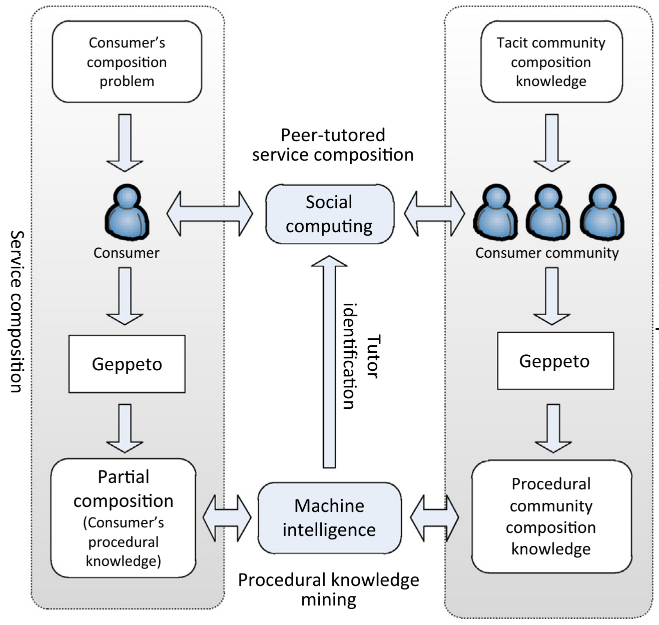

Consumer's composition problem
Tacit community composition knowledge
Peer-tutored service composition
Social computing
Consumer
Consumer community
Service composition
Tutor identification
Service composition
Geppeto
Geppeto
Partial composition (Consumer's procedural knowledge)
Machine intelligence
Procedural community composition knowledge
Procedural knowledge mining

Fig. 1. Peer-tutored service composition and expertise location based on procedural composition knowledge mining.

service computing technology. We describe our consumerized service composition tool Geppeto, discuss benefits of this approach to service composition and challenges that arise in it. In Section 4, we show how peer tutoring can be integrated into a consumerized service composition environment. In Section 5, we describe how service composition procedural knowledge is encoded in the recommender, and describe the peer-tutor recommender algorithm itself. In Section 6 we briefly describe how Geppeto was augmented with the tutor recommender system. To evaluate the effectiveness of the proposed recommender, we carried out a consumer study with our students. The study setup and results are given in Section 7. Section 8 concludes the paper and proposes future research directions.

2. Related work

Our approach to peer-tutored service composition employs recent technology advancements in multiple domains: correlating different aspects of knowledge; knowledge authorship attribution; expertise location and sharing; recommendation in the context of social matching; dynamic, ad hoc, and problem-oriented social networking; and knowledge automation assistance.

2.1. Correlating different aspects of knowledge

Contemporary socially-intelligent computing, based on a synergy of computation and human intelligence, gave rise to research in knowledge management technologies that enable correlation of different aspects of knowledge: tacit and procedural, individual and collective, artificial and human, knowledge and meta-knowledge. Most of these aspects of knowledge correlation are critical for peer-tutored service composition: (I) tutors with required *tacit knowledge* about service composition are identified by *procedural knowledge* embedded in service compositions stored in developed applications, (II) *individual* tutors with required expertise are identified based on mining of *collective* procedural knowledge embedded in applications that are developed by a community of peers, (III) collaborative peer tutoring requires correlating *individual* tacit knowledge of a peer to *collective* knowledge of a community of tutors, and (IV) the *artificial meta-knowledge* derived from procedural knowledge mining is correlated to *peers' (human) tacit knowledge* to enable correct tutor identification. Although knowledge representation, correlation, management and transfer have been in research focus for a long period of time (Liao, 2003), recent technology development enables pragmatic use of this research in various areas such as in search engines (Bobick & Wimmer, 2012), the semantic web (Berners-Lee, Hendler, & Lassila, 2001), and human computation (Quinn & Bederson, 2011).

2.2. Knowledge authorship attribution

Source code authorship attribution is an example of author identification based on procedural knowledge. Authorship attribution is defined as the process of assigning authorship of an unattributed or contentious sample of work to its correct author amongst a finite pool of authors (Burrows, 2010). In his thesis, Burrows introduces authorship attribution as a subset of fields such as software forensics and plagiarism detection. While knowledge authorship attribution is usually used to differentiate between authors' coding styles, we research these methods for the purpose of service composition matching.

2.3. Expertise location and sharing

Several knowledge-based recommender systems are described in the literature that depend on the explicit domain-specific knowledge models using similarities between customer requirements and items or using explicit recommendation rules. In the context of expertise location, identification and selection are considered general search problems (McDonald & Ackerman, 1998). The expertise identification process outputs a list of users in the community who have expertise to share. Once a list of relevant experts is available, the process of expertise selection is used to choose between one or more appropriate expertise sources.

The expertise recommender model is often cited as a general architectural model for expert recommender systems (McDonald & Ackerman, 2000). Another expertise recommender system described in Ehrlich (2003) puts emphasis on the whole process of expertise location as a way to initiate a conversation with the right person rather than merely designing the system as a search tool. Additionally, the importance of expertise locators in the process of linking people who might never have an opportunity to meet face to face is highlighted. The requirements for the design of an expertise locator in this kind of setting are user experience, system performance, and scalability. According to the authors, these requirements can be satisfied by careful design of user profiles that must contain information on what a user knows as well as context and history of a person.

Ackerman and Halverson in (Ackerman & Halverson, 2004) analyze different possibilities for expertise sharing by means of technical augmentation of repositories and social networks. They give examples of technically created community places where people share expertise with one another. Those places are usually implemented using popular communication technologies on the Internet, such as instant chat or Web-based platforms. Another overview of tools for virtual assistance and expert finding is given in Bakker (2002). Implementation of expertise locators is usually carried out using implicit recommender systems, which are based on expertise identification, or social network based recommender systems, which extend this approach with social network connections of the user (Shami, Ehrlich, Gay, & Hancock, 2009).

2.4. Recommendation in the context of social matching

Historically, recommender systems emerged from recommendation problems that explicitly relied on item ratings (Adomavicius & Tuzhilin, 2005). The problem that recommender systems solve is how to estimate ratings for new items that have not been rated yet. Estimated ratings provide sufficient information for the process of recommendation, since we can recommend the items with the highest estimated ratings to the user. Based on the actual process of how recommender systems make their recommendations, they are classified into content-based, collaborative filtering, and knowledge-based recommender systems (Jannach, Zanker, Felfernig, & Friedrich, 2010). Since our goal is tutor recommendation for the purpose of peer tutoring, our research is focused on social matching systems. Introduction of social aspects in context of recommender systems can direct research in two areas: (I) information derived from the social graph can be used to improve predictions of traditional recommendation systems (Xin, King, Deng, & Lyu, 2009) and (II) research on social matching systems as recommender systems that recommend people to each other instead of recommending items to people (Terveen & McDonald, 2005). Some other recommender systems can also be classified as social matching systems. A large portion of social matching systems are based on social relationship analysis by utilizing connections between members of the community (Kautz, Selman, & Shah, 1997). PHOAKS (Terveen, Hill, Amento, McDonald, & Creter, 1997) recommends Web resources within a newsgroup by listing resources for topics users are interested in. Social connection is possible because users are able to see who posted or recommended particular resources. Designer Assistant helps disseminate knowledge by structuring organizational knowledge into a series of questions and answers tagged with owners (Terveen, Selfridge, & Long, 1995). Social matching is also interesting in context of academic publishing where published work and citation patterns can be analyzed for connecting experts in specific research areas (Moreira & Wichert, 2013).

2.5. Dynamic, ad hoc, and problem-oriented social networking

While we focus on dynamic creation of ad hoc communities that emerge around specific problems in service composition, there is significant research on communities of practice that emerge around common domains or interests and are static in nature. A community of practice is a collection of people who engage on an ongoing basis in some common endeavor (Eckert, 2006). Different authors stress different aspects, such as Wenger (1998) who defines communities of practice as groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly.

Results from Chiu, Hsu, and Wang (2006) suggest that group interaction and the strength of the relationships among members are significant predictors of individual knowledge sharing, which nicely fits the model of communities of practice. Another important aspect of knowledge sharing is reciprocity. Since knowledge exchange involves giving and receiving knowledge, community members seek balance between their contribution to the community and what they take from the community. Additionally, the study also shows that the health of a virtual community depends on fostering and maintaining a set of core and experienced individuals whom other consumers can trust. As articulated in Wasko and Faraj (2005), these core consumer communities create a critical mass that sustains the network and maintains the network's usefulness by contributing knowledge to others. The same study describes means of promotion of individual participation in the critical mass, with individual reputation and social capital being most important.

A significant question emerges regarding the motivation of community members. Authors in McLure Wasko and Faraj (2000) suggest that successful communities have members that act out of community interest rather than self-interest. An interesting finding in the same study is that community members do not use the technology to socialize. Successful communities share a common feature in that they have members who enjoy helping others and feel that they have a moral obligation to share knowledge. These findings are helpful guiding principles for design and implementation of a strong community with the common goal of sharing knowledge.

2.6. Service composition assistance

In the work presented in this paper, we use machine learning to identify peers who have developed similar applications in the past and might help other peers in service composition. On the other hand, systems presented in related research mostly use machine learning to provide information that is used directly in application development. Fundamental research related to selection and discovery of suitable Web services focuses on the analysis of quality of service properties in order to make the best service selection (Parejo, Segura, Fernandez, & Ruiz-Cortés, 2014; Zheng, Ma, Lyu, & King, 2011). In addition to QoS, service recommendation can also utilize social connections. For instance, RelevantTrustWalker, a service recommendation method is based on user feedback in combination with trust relationships in social networks (Deng, Huang, & Xu, 2014).

Service composition assistance is particularly emphasized in context of consumerized tools for assembling services into working applications. Mashup Advisor (Elmeleegy, Ivan, Akkiraju, & Goodwin, 2008) estimates the popularity of specific mashup outputs in an IBM Lotus Mashup Maker repository and makes output suggestions using the conditional probability that an output will be included. Additionally, Mashup Advisor modifies the final mashup in order to produce the suggested output. Greenshpan, Milo, and Polyzotis (2009) describe a data model and ranking algorithm for auto-completion in their Mashup development environment. In Tapia, Torres, Astudillo, and Ortega (2011), authors describe an approach for helping mashup builders in choosing relevant API functions by mining information on co-usage of APIs in previous mashups. In Roy Chowdhury, Rodríguez, Daniel, and Casati (2012), authors describe an assisted mashup development environment that suggests reusable mashup patterns by mining community composition knowledge from an existing repository. Other research follows similar reasoning when recommending assistance for mashup or service workflow development (Angeli et al., 2011; Jie, Bo, Junliang, & Xiangtao, 2009; Radeck, Lorz, Blichmann, & Meißner, 2012).

3. Service composition consumerization

In this section, we overview our work on consumerization of services and service composition, which is the basis for the peer-tutor recommender system presented in this paper, and discuss benefits of this approach to service composition and the challenges it brings.

Through our experience with several iterations of simplifying service composition, first with Coopetition Language (CL) (Srbljić, Škvorc, & Skrobo, 2011), which is an extension of WS–BPEL, then with PIEthon (Srbljić et al., 2011), a Python-based DSL for service composition, and with the service composition spreadsheet-like language HUSKY (Srbljic, Skvorc, & Popovic, 2012), we found that service composition could be approachable to consumers with no software engineering training or experience, given appropriate tools and an intuitive representation of services. This would enable consumers to customize and personalize existing consumer-facing services, and to innovate completely new solutions through service composition.

As part of our research on such tools and methodologies for consumerized service composition, we developed a research prototype tool Geppeto.[3] At the most basic level, Geppeto provides a container for hosting Web widgets. Web widgets are graphical user interfaces for consumer-facing services, developed using standard Web technologies like HTML and JavaScript. Geppeto enables consumers to develop their own applications by composing these widgets through a language of intuitive GUI actions and programming by demonstration. By composing widgets in this way, consumers create a composition of the services underlying the widgets on the GUI level. Additionally, consumers can expose the resulting composition as a new widget, which can then be further composed. As widgets are the core component of this environment and the basic building blocks in compositions, we call this approach to service composition *widget-oriented architecture* (Srbljić, Škvorc, & Skrobo, 2009).

There are three types of widgets used in Geppeto: application-specific widgets, generic widgets and consumer assistants. Application-specific widgets provide application domain specific services, for example, driving directions displayed on a map. Generic widgets allow consumers to define data and control flow, event flow, timing, communication, and synchronization of the widget composition through programming by demonstration. Finally, consumer assistant widgets use crowdsourcing and collective intelligence methods to help consumers during widget composition (Silic, Delac, Krka, & Srbljic, 2013; Vladimir, 2013).

The Geppeto user interface and an example composition of widgets is presented in Fig. 2. The composition consists of three widgets: the *GeoLocation* widget, the *CurrentAddress* widget, and the *DrivingDirections* widget. The *GeoLocation* widget outputs the user's latitude and longitude based on their current IP address. The *CurrentAddress* widget returns a human readable address that is estimated from the given latitude and longitude. The *DrivingDirections* widget renders driving directions on a map for the given source and destination addresses. All three of these widgets perform their functionality through communicating with a service over the network.

These widgets can be used separately and their functionality combined through copying data between them and clicking buttons; for instance, the output of the *GeoLocation* widget can be explicitly copied and pasted as the input to the *CurrentAddress* widget. The address produced by the *CurrentAddress* widget can then be copied to the starting address of the *DrivingDirections* widget, while the destination address can be entered manually. However, this process of copying and pasting content between widgets is menial and it would probably be easier to simply type in the source address as well.

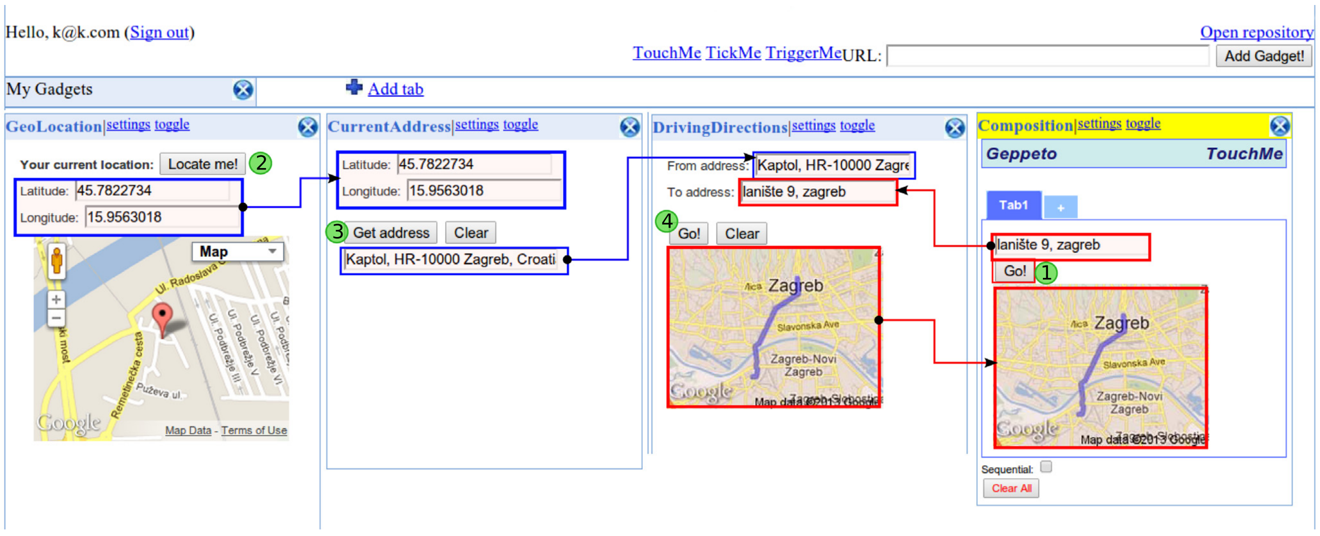
The Geppeto *TouchMe* widget enables consumers to automate this process using the programming by demonstration methodology (Srbljić et al., 2009). First, a consumer can add the Geppeto *TouchMe* widget that is a blank widget without an initial interface. In the second step, the consumer creates the GUI of the *TouchMe* widget by taking GUI elements from existing widgets (marked red in Fig. 2). Then, as the third step, the consumer defines the composition logic for this application by explicitly demonstrating actions on the GUI elements of the widgets. In Fig. 2, the actions shown in blue define the data flow of the composition, such as copying data from one widget to another, and the green marked actions define control flow, for example button mouse clicks. These actions are saved by the Geppeto *TouchMe* widget.

The logic that defines control and data flow of this example widget composition is listed in Fig. 3. The textual representation of the Geppeto composition is a series of GUI actions codified using simple commands. For instance, the command on line 3 (*click LocateMe!@GeoLocation*) defines a mouse click on the *LocateMe!* button of the *GeoLocation* widget. With the definition shown in Fig. 3, the composition is executed by pressing the *Go* button on the Geppeto *TouchMe* widget. Therefore, to get driving directions with this composition, the consumer only needs to enter the desired destination address and press the *Go* button on the *TouchMe* widget, and all the other operations are performed automatically by Geppeto.

A key benefit of a consumerized service composition tool like Geppeto is that it melds application creation with application consumption–consumers do not need to learn any new end-user language, but instead reuse their existing skills attained from using the Web to create applications. To preserve this property, one of the design decisions in Geppeto is not to provide support for general computation – i.e. all functionality must be explicitly exposed in form of Web widgets. However, being mostly a research prototype, the user base of Geppeto is currently limited to the academic and research community. Therefore, the widget repository is still relatively small and covers only basic functionality and popular services.

Moreover, Geppeto applications are created by composing graphical Web widgets, a large number of widgets in the same composition render the composition environment difficult to work with. In our experience, compositions of up to nine widgets are common as nine widgets typically fit on one screen in three rows. Geppeto addresses this limitation by providing support for

[3] Originally an acronym for Gadget End-user Parallel Programming Tool.


Hello, k@k.com (Sign out)

Your current location: Latitude: 45.7822734 Longitude: 15.9563018

CurrentAddress: Latitude: 45.7822734 Longitude: 15.9563018 Kaptol, HR-10000 Zagreb, Croati

DrivingDirections: From address: Kaptol, HR-10000 Zagre To address: lanište 9, zagreb

Composition: lanište 9, zagreb

Open repository
TouchMe TickMe TriggerMe URL:
Add Gadget!
My Gadgets
Add tab
GeoLocation|settings toggle
Locate me!
2
Map
CurrentAddress|settings toggle
3
Get address
Clear
DrivingDirections|settings toggle
4
Go!
Clear
Zagreb
Composition|settings toggle
Geppeto
TouchMe
Tab1
Go!
1
Sequential:
Clear All

Fig. 2. Simple Geppeto composition.

```
1. wait for click Go!@Composition
2. copy Fromaddress@Composition to
        Toaddress@DrivingDirections
3. click LocateMe!@GeoLocation
4. copy Latitude@GeoLocation to
        Latitude@CurrentAddress
5. copy Longitude@GeoLocation to
        Longitude@CurrentAddress
6. click Get address@CurrentAddress
7. copy Address@CurrentAddress to
        Fromaddress@DrivingDirections
8. click Go!@DrivingDirections
9. copy Map@DrivingDirections to
        Map@Composition
```

Fig. 3. Textual representation of the Geppeto composition from the Fig. 2.

hierarchical compositions where complex compositions are exposed in form of single widget that is then used in other compositions as a basic building block.

Several other challenges of service composition remain or are even enhanced in this environment. First, discovering and choosing the right services for a particular problem is becoming increasingly difficult with the fast growth in this area. Second, having found a service, the consumer needs to understand what exactly it does and learn how to use it.

Analogous challenges have been identified in an analysis of the Yahoo!Pipes community (Stolee, Elbaum, & Sarma, 2013). Yahoo!-Pipes is a feed mashup system that shares many characteristics with Geppeto, most of all in that it is directed towards consumers and not exclusively software developers. The authors concluded that many mashup builders need help understanding the behavior of software artifacts. Furthermore, the authors suggest that better development support is required to alleviate this challenge.

In addition to service discovery and selection, which can be partially managed by automatic composition assistance tools, e.g. next service recommendation, critical problem with service composition is the usage of services. In the physical world, people solve problems through communication and collaboration. We apply this approach to address the mentioned challenges in consumerized service composition. To facilitate communication and knowledge sharing between consumers, we utilize recommender systems technology to match consumers with peer-tutors in real time, during service composition. Specifically, based on the partial composition the consumer is creating, the peer-tutor recommender system finds other consumers who have relevant domain knowledge and are willing to share it. This matching is done by comparing the partial composition with previously created compositions and identifying authors of similar compositions who have chosen to be peer tutors. Having identified good peer-tutor candidates, the consumer asking for help is encouraged to engage a peer-tutor through one of the various forms of communication, from instant messaging to e-mail. This symbiosis of a technical system and a social system facilitates knowledge transfer in the consumer community, as well as encouraging collaboration and ad hoc problem-oriented communities.

Existing research supports the viability of peer-tutoring with two key results. First, peer-tutors are able to provide high level tutoring, at the level of professional tutors, and exceeding the results of professional tutoring in some situations (Hinds, Patterson, & Pfeffer, 2001). Second, consumers are motivated to be peer-tutors to other consumers. Ambitious and knowledgeable consumers are willing to help each other, especially if they have a learning environment with peer-tutoring support on their disposal. For example, a study of the Yahoo!Pipes community showed that there is a strong core community of Pipes users that are highly active in problem solving and collective debugging (Jones & Churchill, 2009).

It is important to note that peer tutoring does not need real-time physical contact (e.g. face-to-face interaction) in order to transfer knowledge. Advances in digital communication tools, such as social networks, instant messaging and video chat can be used to facilitate peer tutoring when consumers need remote access to tutors. Additionally, since number of knowledgeable and motivated tutors is considerably smaller than number of consumers and their availability is unpredictable, we are aware of the hypothetical situation when there are no available tutors and the peer tutoring process will be effectively blocked. However, tutor availability is a separate problem that is not focus of the research presented in this paper.

4. Peer-tutoring process

To facilitate knowledge transfer and foster expertise flow from consumers who have certain expertise to consumers who are interested in the given expertise, the tutor recommender analyzes archived procedural knowledge stored in previously created service compositions to recommend tutors and assist consumers who request help. Fig. 4 shows a more detailed description of the peer-tutored service composition process described in Fig. 1. Service compositions created by the consumer community are collected and archived (1). When a consumer asks for help (2), the partial composition is analyzed and compared to the existing compositions to identify potential tutors to recommend. Finally, the list of recommended tutors is returned to the consumer who asked for help (3) in order to bootstrap peer tutoring process (4).

The set of tools and technologies of *social computing*, which serve as intermediaries for social interaction and provide the fundamental collaboration platform underlying the peer-tutoring process, are extended with a new set of tools. The *service composition assistant* helps consumers in service composition. One possible implementation of the service composition assistant is the Geppeto *TouchMe* widget described in Section 3. The *tutor recommender wizard* provides a dynamic list of peer-tutor candidates who might provide help during service composition. One possible implementation of the tutor recommender wizard is the Geppeto *MentorMe* widget described in Section 5.

The service composition assistant and the tutor recommender wizard bridge *social computing* and *machine intelligence* environments. The role of the service composition assistant is twofold. On one hand, it encodes and stores completed service compositions into the composition archive. This process is called *procedural knowledge archiving* in the Fig. 4. On the other hand, it monitors and encodes the consumer's service composition and submits the partially finished compositions to the peer-tutor recommender in the machine intelligence environment. The tutor recommender wizard performs *procedural knowledge mining* of archived procedural knowledge and communicates the results of knowledge mining in the form of a list of recommended tutors to the service composition assistant in the social computing environment.

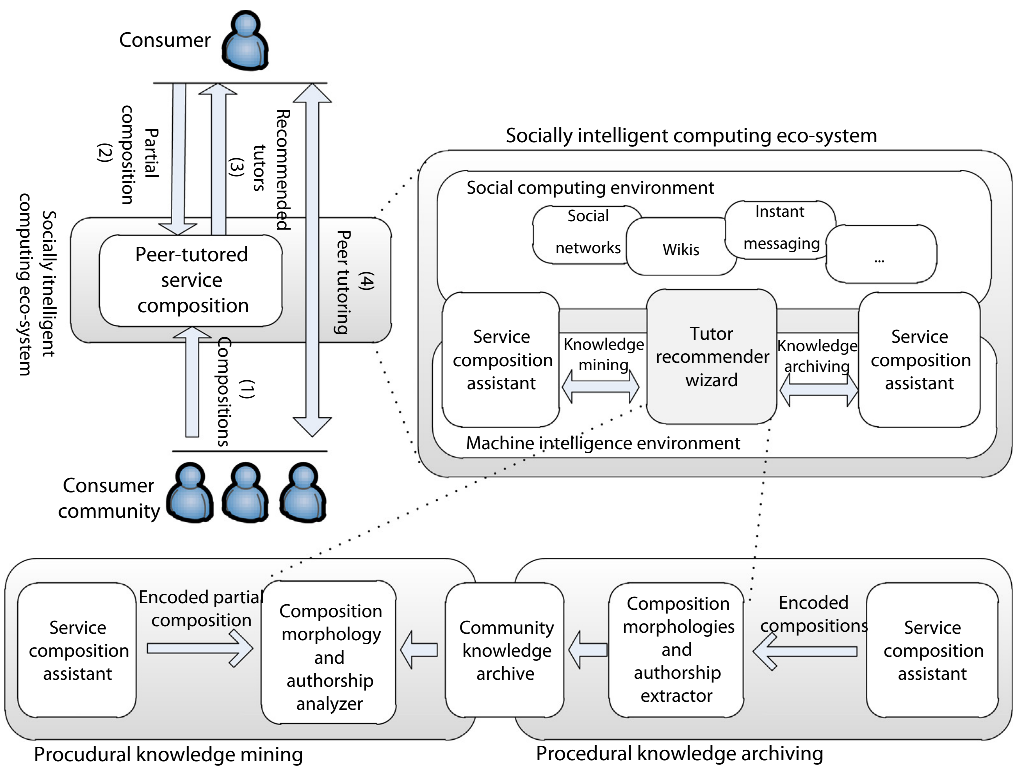
5. Tutor recommendation

In this section, we present possible procedural knowledge encodings and describe the tutor recommender algorithm. Tutor recommendation is performed in two steps. In the first step, the partially finished service composition created by the consumer asking for a tutor is compared with service compositions stored in the procedural knowledge archive. In the second step, similar service compositions are analyzed with respect to authorship. Based on the authors of service compositions that are similar to the partially finished service composition, a list of recommended tutors is generated.

In the research presented in this paper, we assume that each archived service composition has one author and that this information is available. We are aware that this information could not be easily accessible because of consumers' privacy settings and that many service compositions will have more than one author. We leave this issue for future work.

5.1. Procedural knowledge encoding features

Consumers build new compositions using Geppeto by composing existing services defining control, data, and event flow as well as time, communication, and synchronization dependencies. While a precise description of a composition including semantic annotation of the used components and the connections between them might provide more information for precise comparison of a partial composition and an archived composition, this kind of representation presents four significant challenges. First, a formal model that would represents all these features would inherently be complex. Second, matching these representations might be more computationally expensive. Third, an over-specific representation might cause the recommender to miss many useful matchings, which is akin to the problem of overfitting in machine learning. Fourth, a



Consumer
Partial composition (2)
Recommended tutors (3)
Socially itnelligent computing eco-system
Peer-tutored service composition
(4) Peer tutoring
(1) Compositions
Consumer community
Socially intelligent computing eco-system
Social computing environment
Social networks
Wikis
Instant messaging
...
Service composition assistant
Knowledge mining
Tutor recommender wizard
Knowledge archiving
Service composition assistant
Machine intelligence environment
Service composition assistant
Encoded partial composition
Composition morphology and authorship analyzer
Community knowledge archive
Composition morphologies and authorship extractor
Encoded compositions
Service composition assistant
Procudural knowledge mining
Procedural knowledge archiving

Fig. 4. Peer-tutored service composition process.

specific representation might not be applicable in various service composition domains, and would have to be changed to address the particularities of each domain.

Therefore, we chose to use a significantly simpler composition representation based on a binary feature vector that encodes the used component services and the connections between them. This representation can be easily extracted from the textual encoding of a composition, such as that in Fig. 3.

The vector representation of the service composition presented in Fig. 3 is shown in Fig. 5. The composition is illustrated for compactness with an undirected graph where components are represented with vertices and connections are represented with edges.

The feature vector holds binary values, where the value 1 marks the presence of a component or connection in the composition, while the value 0 marks the absence of the corresponding component or connection. The feature vector from Fig. 3 has the value 1 for CA, GL, and DD elements, because the application is composed out of the *CurrentAddress* widget (CA), the *GeoLocation* widget (GL), and the *DrivingDirections* widget (DD). Additionally, CA–GL and CA–DD elements have the value of 1 to denote the existence of connections between these components in the composition.

5.2. Tutor recommendation algorithm

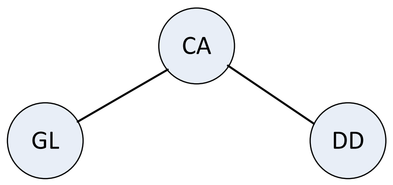
Let the set of all available components be T, and its cardinality be $n_T = |T|$. Since we need to codify the component list and every possible pair of these components, every composition $c_i \in C$ is represented as a vector in n-dimensional space where

$$n = n_T + \frac{n_T(n_T - 1)}{2}. \tag{1}$$

Let the set of all consumers be U. Every consumer $u_i \in U$ has an associated set of widget compositions C_i, and these sets form a partition of the set of all widget compositions C. Let us assume that a consumer u_x created a partially finished widget composition c_p and asked the system for expert recommendation. The tutor recommender system takes the partial widget composition c_p as input and outputs a list of k consumers U^* who have the required expertise.

The first step in the tutor recommendation process is to find service compositions that are similar to the partially finished service composition. One of the most popular methods for computing the similarity between vectors in multidimensional space is Pearson's correlation coefficient (Adomavicius & Tuzhilin, 2005). Pearson's correlation $p_{i,j}$ measures the extent to which two application vectors c_i and c_j linearly relate to each other. We use Pearson correlation to generate a similarity vector. A similarity vector s_p for partially finished service composition c_p is generated by calculating Pearson's correlation between c_p and service compositions in C:

$$s_p = [(i, p_{p,i}) | c_i \in C, a(c_i) \neq u_x], \tag{2}$$



...	CA	GL	DD	CA-GL	CA-DD	GL-DD	...
...	1	1	1	1	1	0	...

Fig. 5. Undirected graph of the consumer service composition defined on Fig. 3 and the corresponding feature vector (CA – CurrentAddress widget, GL – GeoLocation widget, and DD – DrivingDirections widget).

where $a(c)$ denotes the author of the service composition c and i is the unique identifier of c_i. Service compositions written by author u_x are not included in the similarity vector.

In order to obtain the most similar compositions to c_p, we sort the tuples in s_p and generate a sorted similarity vector ss_p. Since our goal is to recommend authors of similar service compositions, we extract authors from the sorted similarity vector ss_p:

$$U^{**} = [a(c_i) | (i, p_{p,i}) \in ss_p]. \tag{3}$$

The final list U^* is then generated by removing duplicates from U^{**}. U^* is the list of consumers who have built service compositions most similar to the observed partially finished service composition c_p.

For example, assume that the consumer aims to develop an application similar to the service composition in Fig. 3. We will present how Pearson's correlation coefficient for the partially finished service composition and the completed service composition changes as the partial composition approaches the final goal.

After adding *CurrentAddress* and *GeoLocation* widgets, the feature vector for the partially finished composition is $[1,1,0,0,0,0]$ and the Pearson correlation coefficient is 0.31. By adding the data flow action of *copy Latitude@GeoLocation to Latitude@CurrentAddress*, the feature vector for this partially finished composition is $[1,1,0,1,0,0]$ and the Pearson correlation coefficient is 0.44. By adding the *DrivingDirections* widget and the data flow action *copy Address@CurrentAddress to Fromaddress@DrivingDirections*, the feature vector for this partially finished composition is $[1,1,1,1,1,0]$ and the Pearson correlation coefficient changes to 1.0.

The recommended lists of tutors from our experimental dataset for the given three situations are: A.R., S.R., P.D. for the first stage, A.R., S.R., P.D for the second stage, and S.R., A.R., P.D for third stage of service composition. Since the actual author of this composition was S.R., we can see that this author is identified as a good candidate peer-tutor in all three stages because the partial composition is similar to the completed composition.

We conclude this section with an analysis of the computational complexity of the described recommender algorithm. With n_T distinct components available in the composition system, the dimensionality of the feature vector space is $O(n_T^2)$, as both components and undirected connections are represented with one dimension each. However, the feature vectors can be encoded much more efficiently with a sorted list of nonzero coordinates. This is because feature vectors in this space are very sparse as most compositions use a much smaller number of component services. Given a composition with n components and m connections, the feature vector encoding size is therefore $O(n + m)$.

Computing the Pearson correlation coefficient of two vectors requires time linear in the size of the vectors. The algorithm computes the Pearson correlation coefficient between the vectors of the input partial composition and every composition in the archive. This is a limiting factor in the scalability of the algorithm. With this simple approach, archives of several thousand compositions can easily be processed in a responsive manner, which is sufficient for many environments. As the main objective of this paper is to test the effectiveness of peer tutoring with respect to service composition knowledge identification, we do not discuss this issue of computational complexity further.

6. Tutor recommender tool

To support peer tutoring, we have augmented Geppeto with the tutor recommender system presented in Section 5. The implementation of the recommender in the Geppeto environment is presented in Fig. 6. The Geppeto environment consists of a frontend that runs within the browser and a backend that runs on one of the Geppeto cloud servers. The frontend consists of a *widget container* and a *widget list* tool. The widget container provides a runtime environment for widget execution and composition, and provides tools for widget administration. The widget list tool is used by consumers to access existing widgets that are used as components. The Geppeto *TouchMe* widget records consumer created compositions, as described earlier. As part of the tutor recommender system, we implemented the Geppeto *MentorMe* widget that allows consumers to ask for a tutor and then lists tutor candidates retrieved from the recommender.

The *system backend* runs services that support the execution of the widget container's tools, consumer widgets, the *TouchMe* and *MentorMe* widgets, and runs the widget database service, the recommender service and other services that support things like authentication and content caching.

A more detailed view of the recommender service as part of the tutor recommender subsystem is shown in Fig. 7. The *MentorMe* widget monitors the process of service composition and forwards partially finished compositions to the recommender service (1). The *composition pre-processor* component is responsible for parsing consumer service compositions and extracting feature vectors. Additionally, the composition pre-processor periodically updates the *feature vector database* with vector representation of finished consumer service compositions stored in the service composition database. The feature vector that represents the partial composition is then forwarded to the *service similarity calculator* (2). The service similarity calculator then produces a sorted list with the most similar compositions fetched from the feature vector database (3) and forwards that list to the *authorship identifier* module (4). The authorship identifier assigns author names to the selected service compositions. Finally, this list of authors is returned to the consumer through the *MentorMe* widget (5).

6.1. MentorMe widget user interface

The user interface of the *MentorMe* widget is presented in Fig. 8. The *MentorMe* widget displays a list of widgets currently in use by the consumer, i.e. widgets that are being used to create a new composition. Consumers have an option to remove some of these widgets directly from the *MentorMe* widget, which enables them to direct the tutor recommender system to the particular problem area by excluding the removed widgets from the query feature vector. The recommended peer-tutors are displayed with their username, social contacts, availability status, similarity score, and a timestamp of their last activity.

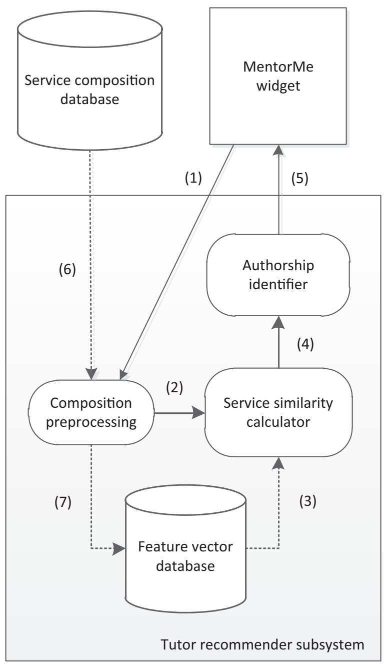

Service composition database
MentorMe widget
(1)
(5)
(6)
Authorship identifier
(4)
(2)
Composition preprocessing
Service similarity calculator
(7)
(3)
Feature vector database
Tutor recommender subsystem

Fig. 7. Architecture of the Tutor recommender subsystem.

7. Consumer study

We evaluated our peer tutoring system on a group of 56 students through assignments they completed in a computer lab. The assignments were structured as sets of tasks, where each task was a relatively simple service composition that students were directed to create using Geppeto. All the service compositions were defined to be similar in complexity and in expected completion time. The average task required composing three to six simple widgets into a new application. For instance, one task was defined as follows: *Compose a new widget that renders driving directions from your current address to a given destination address on a map.*

Each student had a separate cubicle with a computer running only a Web browser with two tabs opened: Geppeto running in one and a stopwatch application in the other. We explicitly requested and enforced complete silence and no cooperation between neighboring students. Before handing out assignments,

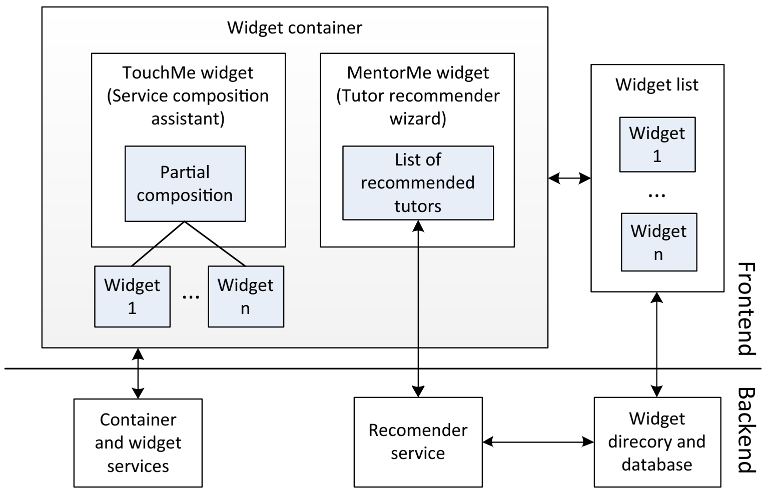
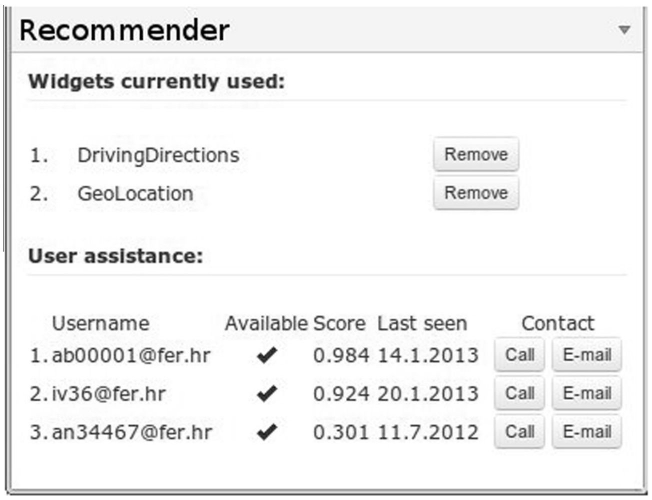

Widget container
TouchMe widget (Service composition assistant)
Partial composition
MentorMe widget (Tutor recommender wizard)
List of recommended tutors
Widget 1
...
Widget n
Widget list
Widget 1
...
Widget n
Frontend
Backend
Container and widget services
Recomender service
Widget direcory and database

Fig. 6. The Geppeto environment enhanced with the tutor recommender system.


Recommender

Widgets currently used:

1. DrivingDirections Remove
2. GeoLocation Remove

User assistance:

	Username	Available	Score	Last seen	Contact	
1.	ab00001@fer.hr	✓	0.984	14.1.2013	Call	E-mail
2.	iv36@fer.hr	✓	0.924	20.1.2013	Call	E-mail
3.	an34467@fer.hr	✓	0.301	11.7.2012	Call	E-mail

Fig. 8. Geppeto MentoMe widget user interface.

we gave a 5-min introductory course on Geppeto to cover basic service composition methodology. Each assignment was defined with a list of tasks printed on a piece of paper. Assignments were handed out to students after the short introduction. There were a total of fifteen different service composition tasks randomly distributed among the students, and randomly ordered within each assignment to avoid any potential systematic impact of the sequence in which students perform individual tasks. Each student had three to four widget compositions to complete during one sixty minute session.

Upon completing a composition, students were instructed to call the supervising teaching assistants to review their work. Students verbally articulated their solutions and demonstrated their service composition on an example. If the composition was indeed correct, students were instructed to save the solution and mark the time needed to finish the given task. On the other hand, if the service composition did not conform to the task requirements, the student was advised to try again.

7.1. Setup of the study

After the whole group of students listened to the short introductory course on Geppeto, students were divided into three separate groups. Each group had two supervising teaching assistants that monitored students' progress. Once the first group of students finished their projects without any help, they were asked to be peer tutors for the other two groups. We call this first group of students as the *No Assistance group*.

Students from the second group were given the same set of assignments as students from the first group. However, unlike the first group, students from the second group were encouraged to ask for help from the *No Assistance group* students using the *MentorMe* widget. Students from the No Assistance group would physically approach the student who asked for help assuming the role of a *student tutor*. Student tutors were advised to exclusively answer questions they were explicitly asked. We limited interaction time to two minutes. We denote this second group of students as the *Recommender Assistance group*.

The third group of students followed the same rules as the *Recommender Assistance group*, except that the tutor recommender algorithm was changed to choose candidate tutors randomly from the *No assistance group* when students would ask for help. We call this final group of students the *Random Assistance group*.

7.2. Results and discussion

To be as objective as possible in peer tutoring evaluation, we chose different groups of metrics. These metrics include both parties in peer tutoring, one receiving help and one providing help. Also, we include objective metrics based on performance measurement and subjective metrics based on consumer experience. Therefore, we use two performance metrics and two *quality of experience* (QoE) metrics.

The performance metrics measure consumers' behavior. We chose two performance metrics: time-on-task and task success. The time-on-task metric measures time consumers need for completion of the service composition. The task success metric measures percentages of consumers that successfully complete service composition. The QoE metrics are a self-reported metrics based on consumer's experiences. Two user experience metrics are used that represent the subjective satisfaction of either those who received or provided help. Consumers rated their satisfaction with the help provided from tutors, while tutors rated their own satisfaction with the help they provided.

In this section we present and discuss the results of our student studies for each metric separately. Before discussing the experimental results, Table 1 shows how different groups perceived the complexity of the given widget composition tasks. Upon completing tasks, students were asked to assign a complexity score on a scale from 1 to 5, where 1 corresponds to simple tasks and 5 to complex tasks. Since there is no significant difference between groups, we conclude that we can safely compare results from different groups and that results of these comparisons are meaningful.

7.2.1. Time-on-task metric

We start measuring task time at the moment when students start reading the text of the task and stop measuring at the moment when the supervising TA confirms that the task service composition is successfully finished. The time-on-task results for a total of 188 successfully completed tasks are shown in Table 2.

Our results show that the *Recommender Assistance group* performed significantly better having time-on-task almost 3 min lower than the *No Assistance group*, i.e. an improvement in task completion time is 28.2% (t-test's p-value < 0.05), while the *Random Assistance group* performed slightly better than the *No Assistance group*.

While all students in the *No Assistance group* that act as tutors have equal experience in using Geppeto, only some of them have domain expert knowledge on any specific composition task. This includes knowledge about simple widgets that need to be composed and how they need to be interconnected to create the desired functionality. Therefore, randomly chosen tutors were able to provide only general help on using Geppeto as a composition tool, which slightly improved task completion time for the members of the *Random Assistance group*. However, our *MentorMe*

Table 1
Average perceived task complexity.

	Task complexity
No assistance group	2.62
Recommender assistance group	2.67
Random assistance group	2.52

Table 2
Completion time for successfully completed tasks and task success rates per group.

	Time-on-task (min)	Task success (%)
No assistance group	10.51	95.30
Recommender assistance group	7.53	98.50
Random assistance group	9.41	91.04

Table 3
User satisfaction results.

	Help received	Help given
Recommender assistance group	4.35	4.19
Random assistance group	3.80	3.75

widget correctly identified tutors with needed domain expert knowledge, which significantly improved task completion time for the *Recommender Assistance group* and proved our hypothesis that domain experts can be identified based on procedural knowledge inferred from their previous work.

7.2.2. Task success metric

The results for task success in Table 2 also prove our hypothesis. While the *Recommender Assistance group* had a nearly perfect task success rate, the *Random Assistance group* had poorer results than the *No Assistance group*. These results show that false positive recommendations have a negative impact on task success rate. For example, during the experiment, we observed a sequence of three randomly recommended tutors who were unable to assist a student in finding a solution.

We conclude that false positive recommendations negatively impacted the *Random Assistance group*, but did not impact the *Recommender Assistance group*, showing that the proposed tutor recommender system reduces the number of false positive recommendations.

7.2.3. Consumer satisfaction metrics

Our fourth measure of the effectiveness of the proposed peer-tutor recommender system is the subjective satisfaction of those who received help and those who provided help. We asked students from the *Recommender Assistance* and *Random Assistance* groups to rate the tutoring received from students in the *No Assistance group*. Also, students from the *No Assistance group*, who act as tutors, rate their own satisfaction with help they provided. The satisfaction score was a number on a scale from 1 to 5, where 1 means that the student was dissatisfied with help received/provided and 5 means that student was satisfied with received/provided help. Results for both groups are displayed in Table 3.

Results show significant difference in user satisfaction (t-test's p-value < 0.05) with received help between the *Recommender Assistance* and *Random Assistance* groups. This result shows that, on average, students from the *Recommender Assistance group* are more satisfied with received help than students from the *Random Assistance group*. These results show that the recommender finds tutors with the adequate expertise and that these students are more qualified for providing help than randomly selected students from the *Random Assistance group*.

Much more difficult was for students to rate themselves as tutors, because students act as tutors in both groups tried very hard to help. Although randomly selected students who provided help to the *Random Assistance group* did not themselves work with similar service compositions, they still provided help based on their experience with Geppeto. However, students that acted as tutors in the *Recommender Assistance group* showed higher satisfaction with the help they provided than students that tutored in the *Random Assistance group*.

8. Conclusion and future work

In this paper, we build on our previous work in consumerization of services, and define a peer-tutor recommender system for service composition. The presented recommender system is based on socially-intelligent computing through two core processes. First, the recommender uses machine intelligence to identify experts' service composition knowledge based on procedural knowledge stored in previously created compositions. Second, social computing is used to match these experts with their peers who need their assistance in creating a new service composition.

The proposed tutor recommendation algorithm represents service compositions with two sets of features-the list of component services used in the composition, and the list of component connections. These sets of features are derived from the morphology of a service composition and merged into a multidimensional binary feature vector. Given a partial service composition as input, the recommender finds peers with relevant expert knowledge by searching for similar service compositions that have been defined previously. These expert peers are then recommended to the consumer, and the consumer is encouraged to approach them for help, through various means of communication.

The described peer-tutor recommender was implemented as an extension of our consumerized service composition tool Geppeto. Geppeto allows consumers to compose services via service GUI extensions in the form of widgets through programming by demonstration and representing service composition in semi graphical tabular spreadsheet like form.

We evaluated the peer tutoring system through a consumer study. The consumer study was conducted with students in a lab environment. The students were instructed to create several service compositions using Geppeto. All students had previous experience with general purpose programming languages like C and Java, but no experience in service composition or with Geppeto. The students were briefly instructed on the basics of Geppeto at the start of the experiment. They were then divided into three groups-the *No Assistance group* which worked with no outside help, the *Recommender Assistance group* which used the proposed peer-tutor recommender system, and the *Random Assistance group* for which the recommender suggested random tutors. Two performance metrics and two *quality of experience* (QoE) metrics were considered to objectively evaluate the proposed tutor recommender.

The experimental results gathered in the consumer study confirm the effectiveness of the proposed peer-tutor recommender system. We observed a reduction in project completion time ($\approx 30\%$), an increase of overall project success ($\approx 10\%$), an increase of consumers' satisfaction with received assistance ($\approx 15\%$), and an increase of consumers' satisfaction with provided assistance ($\approx 12\%$). The obtained results support our hypothesis that it is possible to identify peer-tutors with required service composition knowledge based on their procedural knowledge stored in previously defined compositions.

These results provide a strong basis for further research in consumerization of service composition tools and peer-tutored knowledge sharing. However, the presented research still has much room for improvement. Besides the already discussed issue with computational complexity of the proposed recommendation algorithm, the problem of tutor availability requires further research. For instance, the recommendation algorithm could make use of data about past tutor activity, scope, score and location. Using this additional information about tutor's past performance, the algorithm could inform consumers whom to call for a specific problem and when to call them. In the extreme case when tutors are not present at all for a certain period of time, some type of fail-back mechanism in form of automatic assistance could be provided, for instance, by suggesting services that could be included into the composition or even suggesting whole compositions to consumers. Besides basic face-to-face interaction, our future research will include social computing tools, primarily video call for remote assistance. Moreover, video call could be enhanced with direct connection of tutor and tutee working environments, much like customer support using remote desktop. Another line of future research could enable tutors to record development progress of their service compositions. Recorded compositions could then be exposed in form of interactive references for the larger consumer community. Furthermore, research on tutor involvement could include some form of external motivation, for example, community badges or "karma" points.

In addition to academia, we believe that the presented research, with its theoretical framework and practical implementation, is also interesting to the industry. Many programming environments would benefit with the introduction of a peer-tutoring platform as the advanced form of community support. The presented recommendation algorithm is based only on components used in the composition and connections between those components, and thus is applicable for any programming environment where basic building blocks could be abstracted into components, such as domain-specific programming languages like Scratch,[4] visual programming languages like PureData[5] or even general purpose programming languages like Python.[6] Additionally, traditional (offline) peer tutoring could scale considerably with the introduction of tutor recommendation system that enables fast and seamless location of tutors with the required expertise. For example, in a university course context where service composition is taught, the presented approach could allow students to identify other students or course staff who could efficiently help them with a problem they might encounter.

Acknowledgments

The authors acknowledge the support of the Ministry of Science, Education, and Sports of the Republic of Croatia through the Computing Environments for Ubiquitous Distributed Systems (036-0362980-1921) research project.

References

Ackerman, M. S., & Halverson, C. (2004). Sharing expertise: The next step for knowledge management. *Social Capital and Information Technology*, 273–299.

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*(6), 734–749.

De Angeli, A., Battocchi, A., Roy Chowdhury, S., Rodriguez, C., Daniel, F., & Casati, F. (2011). Conceptual design and evaluation of wire: a wisdom-aware eud tool. Tech. rep., University of Trento.

Bakker, T. (2002). Virtual reference services: Connecting users with experts and supporting the development of skills. *Liber Quarterly, 12*(2/3), 124–137.

Benkler, Y., & Nissenbaum, H. (2006). Commons-based peer production and virtue*. *Journal of Political Philosophy, 14*(4), 394–419.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 28–37.

Bobick, M., & Wimmer, C. (2012). Knowledge correlation search engine. US Patent App. 13/400,829.

Burrows, S. (2010). Source code authorship attribution (Ph.D. thesis), School of Computer Science and Information Technology, Melbourne, Australia: RMIT University.

Chiu, C.-M., Hsu, M.-H., & Wang, E. T. (2006). Understanding knowledge sharing in virtual communities: An integration of social capital and social cognitive theories. *Decision Support Systems, 42*(3), 1872–1888.

Deng, S., Huang, L., & Xu, G. (2014). Social network-based service recommendation with trust enhancement. *Expert Systems with Applications, 41*(18), 8075–8084.

Eckert, P. (2006). Communities of practice. *Encyclopedia of Language and Linguistics, 2*, 683–685.

Ehrlich, K. (2003). Locating expertise: Design issues for an expertise locator system. In *Sharing Expertise-Beyond Knowledge Management* (pp. 137–158). Cambridge: MIT Press.

Elmeleegy, H., Ivan, A., Akkiraju, R., & Goodwin, R. (2008). Mashup advisor: A recommendation tool for mashup development. In *IEEE international conference on web services, 2008. ICWS'08* (pp. 337–344). IEEE.

Greenshpan, O., Milo, T., & Polyzotis, N. (2009). Autocompletion for mashups. *Proceedings of the VLDB Endowment, 2*(1), 538–549.

Hinds, P. J., Patterson, M., & Pfeffer, J. (2001). Bothered by abstraction: The effect of expertise on knowledge transfer and subsequent novice performance. *Journal of Applied Psychology, 86*(6), 1232.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: An introduction*. Cambridge University Press.

Jie, G., Bo, C., Junliang, C., & Xiangtao, L. (2009). Applying recommender system based mashup to web-telecom hybrid service creation. In *IEEE Global Telecommunications Conference, 2009. GLOBECOM 2009* (pp. 1–5). IEEE.

Jones, M. C., & Churchill, E. F. (2009). Conversations in developer communities: a preliminary analysis of the yahoo! pipes community. In *Proceedings of the fourth international conference on Communities and technologies* (pp. 195–204). ACM.

Kautz, H., Selman, B., & Shah, M. (1997). Referral web: Combining social networks and collaborative filtering. *Communications of the ACM, 40*(3), 63–65.

Liao, S.-h. (2003). Knowledge management technologies and applications-literature review from 1995 to 2002. *Expert Systems with Applications, 25*(2), 155–164.

McDonald, D. W., & Ackerman, M. S. (1998). Just talk to me: A field study of expertise location. In *Proceedings of the 1998 ACM conference on computer supported cooperative work* (pp. 315–324). ACM.

McDonald, D. W., & Ackerman, M. S. (2000). Expertise recommender: A flexible recommendation system and architecture. In *Proceedings of the 2000 ACM conference on computer supported cooperative work* (pp. 231–240). ACM.

McLure Wasko, M., & Faraj, S. (2000). it is what one does: Why people participate and help others in electronic communities of practice. *The Journal of Strategic Information Systems, 9*(2), 155–173.

Moreira, C., & Wichert, A. (2013). Finding academic experts on a multisensor approach using shannon's entropy. *Expert Systems with Applications, 40*(14), 5740–5754.

Parejo, J. A., Segura, S., Fernandez, P., & Ruiz-Cortés, A. (2014). Qos-aware web services composition using grasp with path relinking. *Expert Systems with Applications, 41*(9), 4211–4223.

Quinn, A. J., & Bederson, B. B. (2011). Human computation: A survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 1403–1412). ACM.

Radeck, C., Lorz, A., Blichmann, G., & Meißner, K. (2012). Hybrid recommendation of composition knowledge for end user development of mashups. In *ICIW 2012, The seventh international conference on internet and web applications and services* (pp. 30–33).

Roy Chowdhury, S., Rodríguez, C., Daniel, F., & Casati, F. (2012). Baya: Assisted mashup development as a service. In *Proceedings of the 21st international conference companion on world wide web* (pp. 409–412). ACM.

Shami, N. S., Ehrlich, K., Gay, G., & Hancock, J. T. (2009). Making sense of strangers' expertise from signals in digital artifacts. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 69–78). ACM.

Shirky, C. (2010). *Cognitive surplus: How technology makes consumers into collaborators*. Penguin.

Silic, M., Delac, G., Krka, I., & Srbljic, S. (2013). Scalable and accurate prediction of availability of atomic web services. *IEEE Transactions on Services Computing, 1*.

Srbljic, S., Skvorc, D., & Popovic, M. (2012). Programming languages for end-user personalization of cyber-physical systems. *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications, 53*(3).

Srbljić, S., Škvorc, D., & Skrobo, D. (2009). Widget-oriented consumer programming. *Automatika, 50*(3-4), 252–264.

Srbljić, S., Škvorc, D., & Skrobo, D. (2011). Programming language design for event-driven service composition. *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications, 51*(4).

Stolee, K. T., Elbaum, S., & Sarma, A. (2013). Discovering how end-user programmers and their communities use public repositories: A study on yahoo! pipes. *Information and Software Technology, 55*(7), 1289–1303.

Tapia, B., Torres, R., Astudillo, H., & Ortega, P. (2011). Recommending apis for mashup completion using association rules mined from real usage data. In *SCCC* (pp. 83–89).

Terveen, L., Hill, W., Amento, B., McDonald, D., & Creter, J. (1997). Phoaks: A system for sharing recommendations. *Communications of the ACM, 40*(3), 59–62.

Terveen, L., & McDonald, D. W. (2005). Social matching: A framework and research agenda. *ACM Transactions on Computer–Human Interaction (TOCHI), 12*(3), 401–434.

Terveen, L. G., Selfridge, P. G., & Long, M. D. (1995). Living design memory: Framework, implementation, lessons learned. *Human–Computer Interaction, 10*(1), 1–37.

Vladimir, K. (2013). Peer tutoring in consumer computing. (Ph.D. thesis). Faculty of Electrical Engineering and Computing, Croatia: University of Zagreb.

Wasko, M. M., & Faraj, S. (2005). Why should i share? Examining social capital and knowledge contribution in electronic networks of practice. *MIS Quarterly*, 35–57.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge University Press.

Xin, X., King, I., Deng, H., & Lyu, M. R. (2009). A social recommendation framework based on multi-scale continuous conditional random fields. In *Proceedings of the 18th ACM conference on information and knowledge management* (pp. 1247–1256). ACM.

Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding mashup development. *Internet Computing, IEEE, 12*(5), 44–52.

Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2011). Qos-aware web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing, 4*(2), 140–152.

[4] http://scratch.mit.edu/
[5] http://puredata.info
[6] https://www.python.org/