Service Subscription and Consumption for Personal Web Applications

Chunyang Ye, Young Yoon, and Hans-Arno Jacobsen

Middleware Systems Research Group
University of Toronto

Abstract. Web services have played a vital role in our daily life for some time now. A wide spectrum of online applications have been developed in diverse domains such as banking, shopping, gaming, and video streaming. However, the end-user does often not have the means to tune the applications to her personal needs and interests, especially not across services from different providers. Moreover, the end-user can not take full advantage of the myriad of useful resources and services available on the Web, as interoperation among different services is often not given. Hence, the new Web application paradigm called *Personal Web* has emerged. The key idea behind the Personal Web is to have Web services exploit Web data that is collected and organized automatically according to the end-users' context and preferences. This paper introduces a new concept that enables Personal Web applications, namely, *service subscription and consumption*. This new concept is driven by events exposed from Semantic Web resources and Web services through PADRES, a distributed content-based publish/subscribe messaging substrate, and POLARIS an approach for event exposure at service interfaces. We explain service subscription and consumption based on a comprehensive scenario and design a framework and architecture that realizes the approach.

1 Introduction

In 1976, Niklaus Wirth presented the insight that in computer programming, algorithms and data structures are inherently related [1]. Since then programming has evolved, embracing distribution and the Web. Complex applications are formed with Web services [2] following the service-oriented architecture (SOA) paradigm. Also, data structures are represented as linked data catering towards the vision of a Semantic Web [3].

In this chapter, we build on Wirth's philosophy, proposing a new interpretation of Web applications as combinations of SOA principles and Semantic Web ideas. This novel perspective motivates us to devise a holistic framework for automatically composing personal Web applications based on individual end-users' interests and needs.

The basic idea behind our framework is as follows: We make Web services aware of personal data published as linked data to the Semantic Web. Here, personal data such as comments on a movie and product wish lists are produced through personal Web applications that are shared with others across social networking platforms such as Facebook. Today, it is not uncommon to subscribe to the updates of personal linked data and to get notified about others' activities of interest. We take this user experience to the next level. That is personal Web application users can also *subscribe* to Web services

M. Chignell et al. (Eds.): The Personal Web, LNCS 7855, pp. 49–64, 2013.

that are triggered upon the update of others' personal linked data. For example, a user may first subscribe to movie recommendations made by her Facebook friends. Then, this user subscribes to an online box office service that is automatically triggered to reserve movie tickets based the user's calendar data and her friends' recommendations about current movies.

To make this vision of a *Personal Web* a reality, there are several technical challenges to overcome. First, SOA and Semantic Web adopt different standards and protocols to share and exchange data. Thus, interoperability between both technologies poses a problem. Second, data and services usually belong to different organizations that are beyond the end-users' control. Hence, coordinating data and services across organizational boundaries is a further challenge. In the rest of this paper, we lay out how we address these challenges with a novel approach we refer to as *service subscription and consumption*.

2 Service Subscription and Consumption Overview

In this section, we present an overview of our methodology, namely *service subscription and consumption*, that overcomes the obstacles in jointly harnessing SOA and Semantic Web concepts.

In order to enable the interoperability between SOA and Semantic Web, we have to first enable Web services to access linked data from Semantic Web resources. The difficulties in achieving this are as follows: (1) The data from the Semantic Web resources and Web services may not belong to the same organization; (2) the data may have different formats and semantics. Therefore, existing Web services cannot access and manipulate the linked data directly.

Our solution is to bridge the gap between the Semantic Web and Web services based on *"events"*. An event is defined as a state change [4]. We view the linked data in Semantic Web as states. Any modification to the linked data is regarded as a state change. For example, the update of a user's online calendar is an event indicating that the state of the calendar has changed. Such an event can be propagated to Web services that are interested and authorized to receive such updates, even if they are administratively separated from one another. Upon the notification via events, Web services can determine individual end-users' personal contexts. For instance, a box office service could become aware of the changes in availability of end-users.

In order to enable this event-driven solution, exposing contents of events is imperative. We do this through a novel concept called *event interfaces* [5,6]. An event interface declares what events should be exposed at runtime and what events Web services want to subscribe to. For example, Figure 1 shows an event interface for an end-user's calendar. This interface declares the events that are raised upon the update of particular linked data elements. Similarly, a Web service can also declare in its own event interface what kinds of events it is interested in. For instance, a booking service subscribes to events from the user's calendar and box office through the respective event interfaces.

The interaction between the Semantic Web and Web services through event interfaces requires a communication mechanism. Here, we use the publish/subscribe (in short pub/sub) communication paradigm [7]. In pub/sub, loosely-coupled clients communicate in an asynchronous fashion. Figure 1 gives an example of how pub/sub is

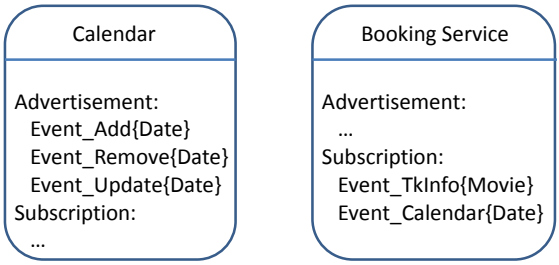

Fig. 1. An example event interface

applied in this example. The online calendar *advertises* what it intends to *publish* in the future. The booking service *subscribes* to the events the online calendar may publish. Whenever the calendar is updated, an event is exposed and propagated to the booking service. Upon the receipt of events, the booking service keeps a copy of the updated linked data. Later, the booking service can also update the copy of the linked data. Upon this update, the booking Web service can generate events and expose them in its own event interface. For example, the booking service reserves a time slot for watching a movie, based on the event received from the online calendar of a given user. Subsequently, the online calendar can be updated to block the time slot reserved for the movie, if the online calendar also subscribed to the events published by the booking service.

The pub/sub and event interface approaches solve the interoperability problem between Web services and Semantic Web. However, we are still left with the issue that linked data may not be under the control of Personal Web users. To address this issue, we devise an interface for Personal Web users to subscribe to Web services. This is to invoke a subscribed service automatically whenever it acquires the required linked data. For example, an end-user may issue its subscription to a box office service. The box office service is invoked automatically to reserve tickets for the user when a movie recommended by the user's friends is currently playing. More specifically, a Personal Web user subscribes to a Web service based on Event-Condition-Action (ECA) rules. In the ECA rules, the *Event* component describes what the subscribed service should listen to. The *Condition* component describes the conditions for invoking the subscribed services. The *Action* component describes what the services do when triggered. For example, the following ECA rule is defined for the example in Figure 1:

Event = Recommended movie is currently in theatre.
Condition = There is an available time slot in the end-user's calendar.
Action = Box office service automatically reserves the ticket for the end-user.

The event interface can be implemented through the POLARIS framework [8]. For exchanging the events through event interfaces, we employ the brokered PADRES content-based pub/sub system [7]. The details of the implementation using the the above described building blocks are presented in Section 4.

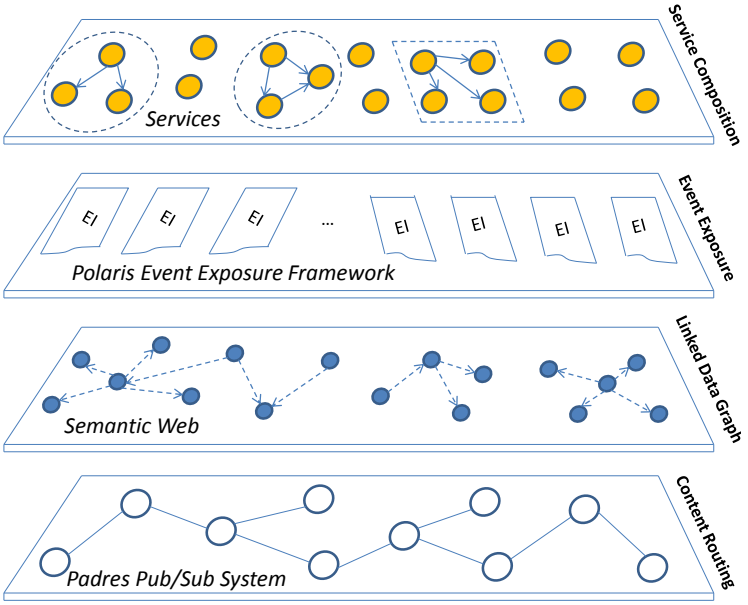

Fig. 2. Methodology overview

3 A Sample Use Case

This section illustrates our solution based on an online Personal Web movie recommendation and booking scenario.

John is a movie fan who closely follows the review of his friends Ebert and Roeper in some social networking application. As shown in Figure. 3, John would like to use an application that can automatically reserve a ticket whenever both Ebert and Roeper recommend a newly released movie as long as John's personal schedule permits. We describe how our service subscription and consumption concept applies to satisfy John's need.

First, John has to subscribe to recommendations by Ebert and Roeper. Since the recommendations are conveyed via a social networking application, the data is usually represented as linked data inside the application. In order to subscribe to updates of recommendations, events are defined to represent the change of recommendations (*e.g.*, recommendations added or updated). As shown in Figure 4, federated content-based publish/subscribe brokers route the recommendation events of Ebert and Roeper towards John.

Recommendation events are abstracted and represented as linked resources in a standardized RDF format and are added to Semantic Web repositories. A Semantic Web repository implements the event exposure interface [5] that can convert an update to the repository (*e.g.*, a recommendation by Ebert or Roeper) to a publication message which is forwarded to interested subscribers. The consumption of the event

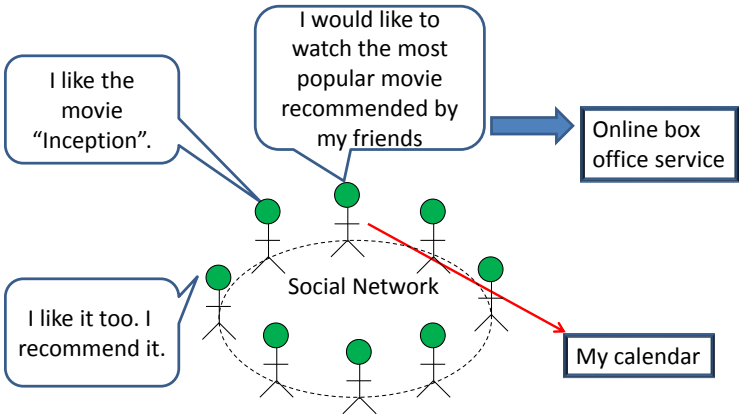

Fig. 3. Automatic ticket booking system in the Personal Web

(contained in a publication message) is represented as linked data and added into a Semantic Web repository, as shown in Figure 5.

We now introduce the novel concept of subscribing and consuming a service. So far, what John can only do upon receipt of the recommendation event is to reserve tickets manually, *i.e.*, examine his personal calendar and search for a ticketing service. Instead, with service consumption and subscription, a composition of services executes desired tasks upon receipt of events on behalf of John. As shown in Figure. 6, triggering and executing of a task is based on Event-Condition-Action (ECA) rules which are declared by end-users.

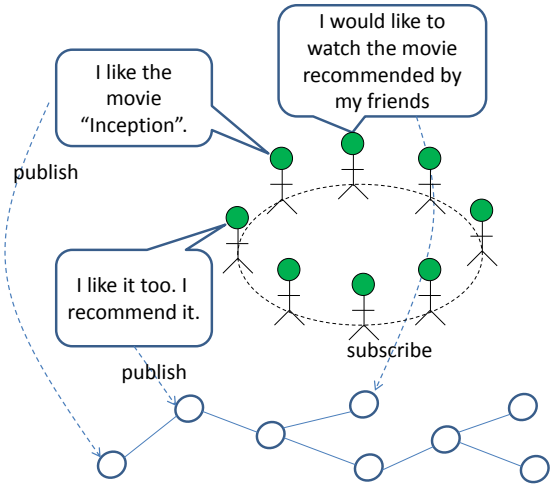

Fig. 4. Event-based subscriptions to linked data updates

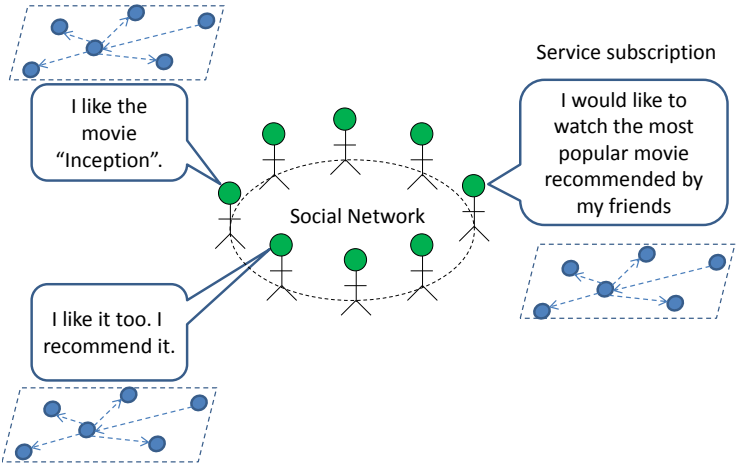

Fig. 5. Semantic Web repository update as an event

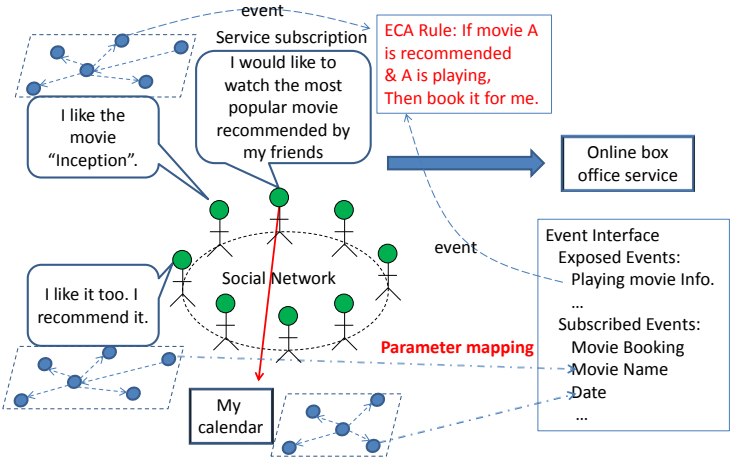

Fig. 6. Event routing through federated content-based publish/subscribe brokers

The *condition* of the ECA rule represents the current state. For example, a service may check the condition of whether the recommended movie John wants to watch is available in the nearest theater or not. This condition is advertised as an event in the event interface of a service, e.g., the booking service. The *event* (e.g., recommendation by Ebert and Roeper) triggers an action (e.g., a reservation process). It is highly likely that an action can be triggered by multiple events that are in conjunction or disjunction. This is handled through *composite subscriptions*. Also, as shown in Figure 7, an action is actually a process which can be composed with other services in a distributed fashion given additional end-user specific constraints, for example, John's preferred payment option and delivery date preference.

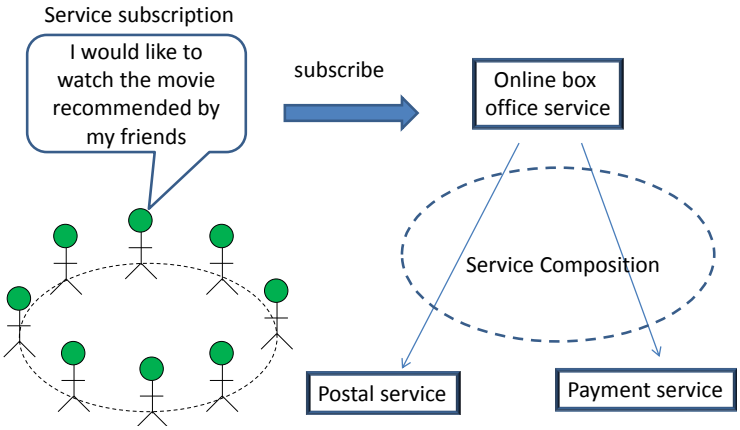

Fig. 7. A service composition for the execution of a process to satisfy the end-user constraints

The critical components to support service subscription and consumption are a content-based publish/subscribe messaging system, event-exposure interfaces, composite subscriptions, and a process planning framework, which we discuss in the following section.

4 Prototype Implementation

In this section, we present the key components that enable our service subscription and consumption concept, presented in the previous section.

4.1 The PADRES Event Dissemination Substrate

In the previous section, we described how the Semantic Web and Web services can interact by publishing and subscribing to events. Since Semantic Web and Web service data sources are distributed, we need a event dissemination substrate that can route events in a distributed and scalable manner. PADRES [7] developed by the Middleware Systems Research Group at the University of Toronto satisfies these requirements. PADRES is a content-based publish/subscribe system that routes publications to interested subscribers through an overlay network of brokers.

The architecture of the PADRES broker is presented in Figure 8. The broker is mainly responsible for matching events (i.e., publications) against subscriptions and relaying the publications to the next destination according to the outcome of the matching process. The broker is equipped with an efficient matching engine for filtering historic and future events based on subscriptions and correlating events from multiple data sources [9,10,11].

A publisher issues an *advertisement* before it can publish. Advertisements are disseminated to all brokers in the overlay network. Subscriptions are routed based on

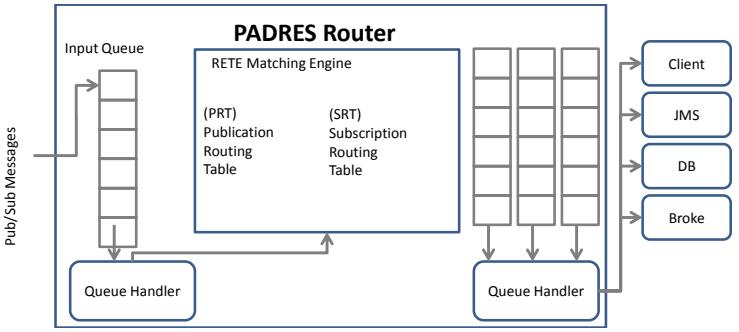

Fig. 8. Architecture of a PADRES broker

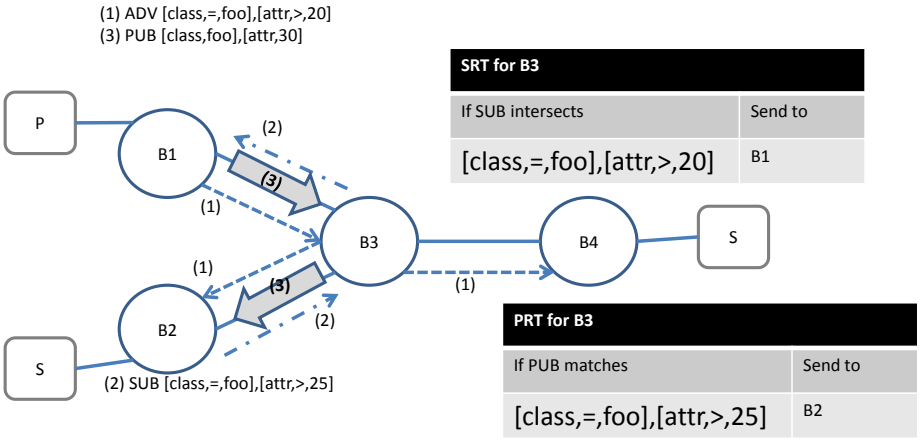

Fig. 9. PADRES Broker Network

the Subscription Routing Table (SRT). The SRT is essentially a list of [advertisement,last hop] tuples. If a subscription matches an advertisement in the SRT, it is forwarded to the last hop broker the advertisement came from. In this way subscriptions are routed towards the publisher along the reverse-path of the advertisement the publisher issued. Subscriptions are used to construct the Publication Routing Table (PRT). Like the SRT, the PRT is a list of [subscription,last hop] tuples, which is used to route publications. If a publication matches a subscription in the PRT, it is forwarded to the last hop broker the subscription came from. This process continues until the publication reaches the subscriber. Figure 9 illustrate an example of content-based routing. In Step *1)*, in the figure, an advertisement is published at B_1. In Step *2)* a matching subscription enters from B_2. Since the subscription matches the advertisement at broker B_3, it is sent to B_1. In Step *3)* a publication is routed along the path established by the subscription to B_2.

4.2 The POLARIS Event Exposure Framework

POLARIS is an event exposure framework for Semantic Web and Web services. It is built on top of PADRES, which we described in the previous section. The architecture of the POLARIS framework is shown in Figure 10. The framework consists of the following three components: Event listeners, pub/sub adapters and a rule engine.

Event Listener. The event listener module is designed to monitor state changes over the Web. Different data sources may have different event listeners. For example, one can design an event listener to monitor the data change inside an online calendar application. Similarly, we can define another event listener to monitor the update of linked data about movie information. Note that the update of data generates many raw events, some of which may not need to be exposed. Also, some events may need to be transformed before being exposed. For example, an event indicating the update of a Lunar calendar needs to be transformed into the data format of a Gregorian calendar. Therefore, mapping rules can be specified to filter and transform events. In other words, events that do not conform to the mapping rules are not exposed.

Pub/Sub Adapter. The pub/sub adapter serves to publish and subscribe to events over the Web. If an event listener wants to expose an event, the pub/sub adapter is invoked to publish the event to the underlying pub/sub system. The event is then propagated to interested subscribers. The pub/sub adapter can be used to subscribe to particular

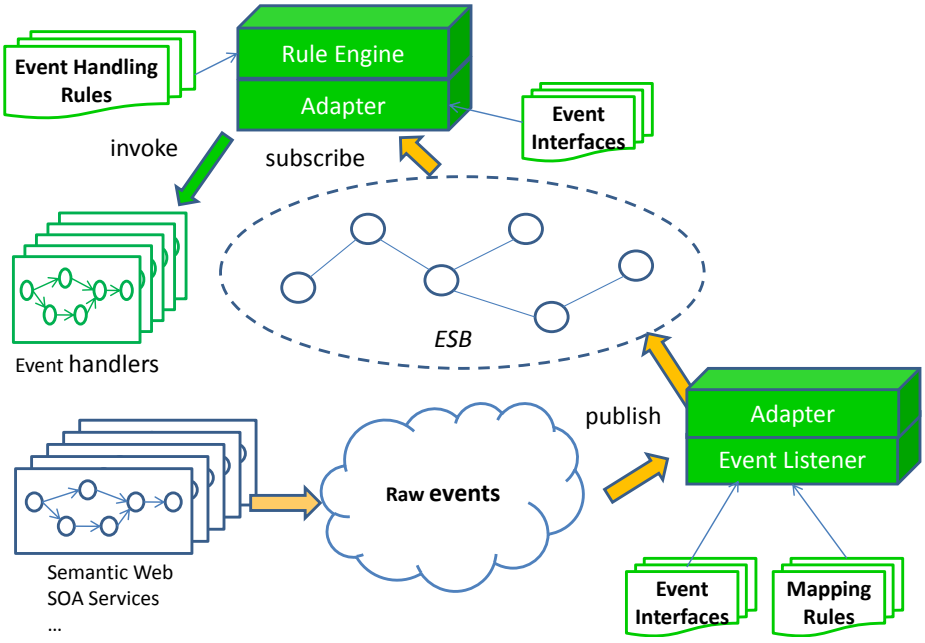

Fig. 10. POLARIS architecture

events. Upon the notification about an event, the pub/sub adapter invokes the rule engine to handle the event. PADRES provides APIs to implement pub/sub adapters.

Rule Engine. The rule engine is responsible for managing and maintaining ECA rules. On being notified about an event, the rule engine checks and evaluates the registered ECA rules. If the conditions for an ECA rule is satisfied, then the action in the rule (e.g., the invocation of a subscribed service) is executed. To describe the service subscription, a description language about the subscribed service is needed. The description language can include functional and non-functional descriptions of the subscribed services. Many existing languages have been devoted to describing and searching services [12,13,14]. In our framework, we focus on functional descriptions of a service based on its behavior. Other aspects of the service subscription language can be extended based on the application requirements.

4.3 Service Composition

The final component for implementing the service subscription and consumption concept is the service composition framework. Revisiting the movie ticket booking example from Section 3, the booking service may involve Web services for ticketing, delivery and payment as shown in Figure 11. Note that the customized booking service for a particular person may not be readily available since the user-specific constraints can be transient and even implicit. For example, John's personal schedule can change over time and John may not be able to watch the recommended movie on certain dates and times. The technical challenge is to infer the implicit constraints and successfully find compatible services that can satisfy those constraints. Also, since there can be thousands of users like John on the Web with various constraints, our framework must be able to compose customized services in a highly scalable manner.

In order to address the technical challenges above, a distributed service composition technique can be used [15]. Suppose John's constraints are as follows: (1) John wants a ticketing service that guarantees to provide online ticketing for AMC Theaters; (2) John wants overnight delivery; (3) John wants to pay with his Visa credit card and (4) John wants to watch the movie at a date and time Mary plans to watch it as well. Interestingly, the last constraint does not explicitly specify the exact date, since John does not necessarily know whether Mary has booked the movie or not. Given these personal constraints, Figure 11 describes how the distributed service composition realizes a personalized workflow just for John. Figure 11 shows a directed acyclic graphic (DAG) that represents the transition between different services to fulfill John's request. We briefly explain how this DAG is constructed in a distributed way.

John's request consists of an input and output. The input constitutes the user-specified constraints such as ticketing options and the output constitutes the results of the service request execution such as delivery status. In our framework, all participating services subscribe to these service requests published by personal agents. For example, the `MovieTicket.com` Web service subscribes to constraints pertaining to payment, ticketing, delivery options, date preferences, and delivery status specified by a personal agent. The `MovieTicket.com` service does not completely fulfill John's request, because this service does not yield the delivery status [15]. Also the UPS delivery

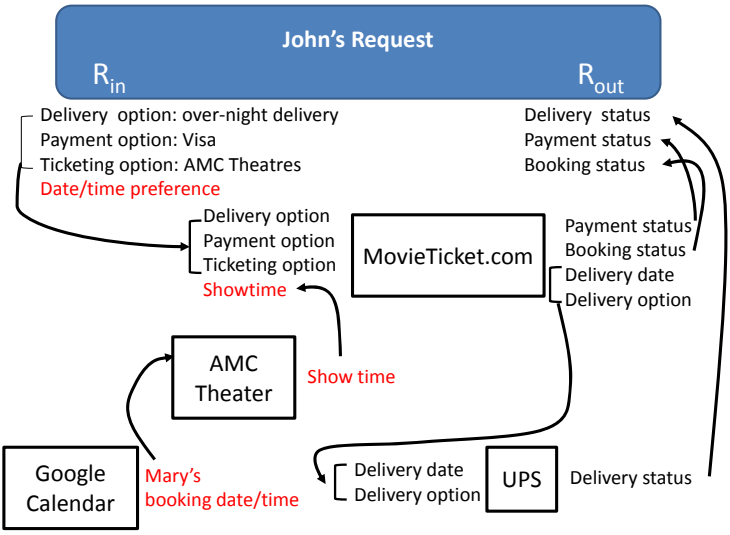

Fig. 11. Distributed service composition

service does not fulfill John's request either, as it does not accept a ticketing option as an input. Hence, the participating services have to concurrently and incrementally search for services that can satisfy John's request on behalf of John's personal agent. For example, the `MovieTicket.com` service has to find a service that can accept a delivery date as input. The `MovieTicket.com` service internally publishes another service request with the delivery date as an input which the UPS delivery service subscribes to. When the delivery service receives this internal publication, it adds the `MovieTicket.com` service as a service that executes a preceding task, so that it knows it can later output a delivery status upon processing the delivery date event issued by the `MovieTicket.com` service. Every service follows this procedure concurrently until services that satisfy John's request are found. Recall that ECA rules govern the processing of events at the POLARIS event-exposure interface as explained in the previous section. For example, an ECA rule can be specified to express that the update of Mary's calendar regarding the movie booking triggers a Google Calendar service to notify the update to the AMC Theater service, which in turn yields the remaining seat availability information for the particular show time Mary booked.

Note that John's preference on date and time is implicitly given as alluded to earlier (John wants to watch the movie at the same date his friend Mary plans to.) Thus, the `MovieTicket.com` service has to infer the exact show time, which is the required input. Our framework extracts John's preferences involving Mary's booking information which is available through the Google Calendar service shared with John.

With the concurrent and distributed service matching process we briefly described above, a transition is added between the Google Calendar service and the `MovieTicket.com` service in the DAG. Also, available seats can be obtained when the Google Calendar service's output (e.g., the show time booked by Mary) is connected to the AMC Theater service that yields the remaining seats for the particular show time.

The AMC Theater service's output also matches one of the `MovieTicket.com` service's inputs, *i.e.,* the availability of a particular show time. Hence, a transition is added between the AMC Theatre service and the `MovieTicket.com` service.

In short, this service composition method realizes a personalized workflow in a concurrent, distributed and incremental manner. It involves personal agents and distributed services, given that there are constraints and an objective declared by end-users. Moreover, there is no centralized orchestrator to direct the execution of the transitions specified in the DAG. This distributed processing of a workflow resembles prior approaches [16,17]. It has been experimentally proven, that under many experimental conditions, the distributed composition approach outperforms approaches for determining and composing candidate services at a central location [15].

5 Related Work

The key idea of the Personal Web is to exploit and integrate Web data and Web services that are collected and organized automatically according to end-users' context and preferences [18]. Many recent research efforts have been devoted to related issues, such as information integration frameworks [19], data sharing [20], privacy concerns [21], and information filtering [22], to just name a few. In our work, we introduce a new paradigm to develop Personal Web applications by integrating linked data concepts with Web services. The newly proposed concept, service subscription and consumption, not only complements the spectrum of Personal Web applications, but also enables end-users to customize the Web applications in an easy to manage manner.

In our solution, one of the key components of POLARIS are the ECA rules that are dynamically specified by end-users. A system that can infer an appropriate action upon receipt of events has already been seen in production systems such as OPS5 [23]. Recently, more flexible declarative approaches for specifying constraints in business processes have been proposed [24,25]. Also, it has been shown that a Dynamic Call Graph (DCR) [26], which formalizes ECA-like rules as Event Structures [27], can be distributed to multiple participants for efficient collaboration. These prior works have been the basis for POLARIS that allows users to specify constraints dynamically and to impose constraints to services participating in the collaboration.

Recently, a number of works about exploiting data for SOA technologies have also been introduced. For example, a *Business Artifact* [28] is a declarative process definition framework that is based on the notion of guard-stage-milestones which are similar to ECA rules. Events are triggered to achieve a milestone. The events can be shared across different tasks, thus the life cycle of a given process can be governed transparently and consistently. However, a Business Artifact cannot be adopted for Personal Web applications for a couple of reasons. The process defined with a Business Artifact is not flexible enough to incorporate end-users' dynamic constraints. Most importantly, event notification is done in a centralized fashion, limiting scalability to a large user base, a critical requirement for Personal Web applications, since they have the potential to involve many services and users from all around the world.

Another similar concept, "Event-driven SOA", also known as SOA 2.0, has been proposed from the perspective of event-based business process execution and

collaboration [29]. The purpose is to define and trigger business applications based on event-driven rules instead of describing the business logic in a procedural manner. The advantage is that business applications can be executed dynamically and can react quickly to changing requirements. In our work, an event is not only a medium to trigger subscribed services asynchronously, but also a way to bridge the gap between the linked data of the Semantic Web and Web services.

Event-driven business process management has been widely adopted in enterprise applications due to the needs for increased flexibility and adaptability of business processes [30,15,16,4,31,32,33,34,35,17]. This requires effectively integrating business logic with the generation, exposure, propagation, detection and handling of events in business applications. Frei *et al.* [36] proposed to use aspect-oriented program (AOP for short) techniques to extract and expose events from legacy enterprise applications. Developers can make use of these events for refactoring legacy applications. In addition, industry standards like BPEL [12], also support two kinds of events, namely a timeout alarm and the receiving of a message, which are local to a BPEL process and are not propagated to other partners. The notification mechanism in BPEL is similar to event notification, but it is based on messages. BPEL processes interact through messages only. In our work, we do not make any assumption about how events are generated. Instead, different web services and Semantic Web linked data can define their own event listeners using the aforementioned solutions and deploy them into our POLARIS framework.

6 Conclusions

In this paper, we introduced a new paradigm to develop Internet applications for Personal Web leveraging Web services and Semantic Web concepts. We integrated PADRES and POLARIS, our content-based publish/subscribe middleware and event exposure framework, to provide a comprehensive solution for service subscription and consumption in the context of the Personal Web. Moreover, we employed composite subscriptions and distributed service composition techniques to support complex and dynamic user-specific rules and constraints. These techniques extend the spectrum of Personal Web applications, allowing end-users to subscribe to Web services and consume services asynchronously and automatically without the need of any designated centralized coordinator.

The current prototype implementation provides some basic functionality to prove the concept of service subscription and consumption for Personal Web applications. The following features complement the work described in this chapter.

– **Selective Service Discovery and Matching.** The given service subscription and consumption framework can be developed further by extending the matching algorithm of PADRES to filter various preferences over services. For example, a user may want to subscribe to services used and recommended by friends. These preferences can impose additional overhead to the service composition we discussed in Section 4.3. This motivates the development of heuristics for determining the best match satisfying a user's needs and interests.

- **Service Wrapper.** In order to make use of linked data, services need to provide an event interface to specify how the data internal to the service and the linked data outside the service is exchanged. This task however is tedious and error-prone. This motivates the development of a tool for services to generate wrappers automatically to map between the data exposed by Web services in event interfaces and the linked data from the Semantic Web.
- **Linked Data Advertisement and Subscription.** In Semantic Web, data from different locations is linked. The linking relationship between data forms linked data graphs. In some applications, if one piece of data is changed, some other data, that is linked to the changed data, may also have to be changed. For example, if a user updates the linked data about his/her favorite movies, the user's friend may want to know the details of the movie that may be available in another location. Therefore, it is interesting to explore the linked data graph to generate the related advertisements and subscriptions automatically. In this way, users can get notification of related data transparently.

Acknowledgments. This research was in part supported by IBM's Center for Advanced Studies, an IBM Faculty Award, a Discovery Accelerator Supplement and a Discovery grant from the Natural Sciences and Engineering Research Council of Canada, an Ontario Early Researcher Award, and a grant from the Mathematics of Information Technology and Complex Systems research network.

References

1. Wirth, N.: Algorithms + Data Structures = Programs. Prentice Hall PTR, Upper Saddle River (1978)
2. Papazoglou, M.P., Traverso, P., Ricerca, I., Tecnologica, S.: Service-oriented computing: State of the art and research challenges. IEEE Computer 40 (2007)
3. W3C: Semantic web, `http://www.w3.org/RDF/FAQ`
4. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2001)
5. Ye, C., Jacobsen, H.A.: The smart internet, pp. 197–215. Springer, Heidelberg (2010)
6. Ye, C., Jacobsen, H.A.: Whitening soa testing via event exposure. IEEE Trans. Softw. Eng., 1–25 (April 2013) (preprint)
7. PADRESweb site, `http://padres.msrg.org`
8. Ye, C., Jacobsen, A.: Polaris: a framework to compose and evolve smart web services via event exposure. In: IBM CASCON Exhibits (2010)
9. Li, G., Jacobsen, H.A.: Composite subscriptions in content-based publish/subscribe systems. In: ACM/IFIP/USENIX International Middleware Conference, pp. 249–269 (2005)
10. Li, G., Muthusamy, V., Jacobsen, H.A.: Adpative content-based routing in general overlay topologies. In: ACM/IFIP/USENIX International Middleware Conference, pp. 1–21 (2008)
11. Li, G., Muthusamy, V., Jacobsen, H.A.: Subscribing to the past in content-based publish/subscribe. Technical Report CSRG-585, Middleware Systems Research Group, University of Toronto (January 2008)
12. OASIS: BPEL 2.0, `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`

13. W3C: Web Service Description Language, http://www.w3.org/TR/wsdl
14. W3C: Web Services Choreography Description Language, http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/
15. Hu, S., Muthusamy, V., Li, G., Jacobsen, H.A.: Distributed automatic service composition in large-scale systems. In: DEBS, pp. 233–244 (2008)
16. Li, G., Muthusamy, V., Jacobsen, H.A.: A distributed service-oriented architecture for business process execution. ACM Trans. Web 4(1), 1–33 (2010)
17. Yoon, Y., Ye, C., Jacobsen, H.A.: A distributed framework for reliable and efficient service choreographies. In: Proceedings of the 20th International Conference on World Wide Web, WWW 2011, pp. 785–794. ACM, New York (2011)
18. Abrams, D., Baecker, R., Chignell, M.: Information archiving with bookmarks: personal web space construction and organization. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 1998, pp. 41–48. ACM Press/Addison-Wesley Publishing Co., New York (1998)
19. Ng, J.: The personal web: smart internet for me. In: Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, CASCON 2010, pp. 330–344. IBM Corp., Riverton (2010)
20. Geambasu, R., Cheung, C., Moshchuk, A., Gribble, S.D., Levy, H.M.: Organizing and sharing distributed personal web-service data. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 755–764. ACM, New York (2008)
21. Mannan, M., van Oorschot, P.C.: Privacy-enhanced sharing of personal content on the web. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 487–496. ACM, New York (2008)
22. Somlo, G.L., Howe, A.E.: Filtering for personal web information agents. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2004, pp. 588–589. ACM, New York (2004)
23. PCAI: OPS5, http://www.pcai.com/web/aiinfo/pcaiops.html
24. Pesic, M., van der Aalst, W.: A Declarative Approach for Flexible Business Processes Management, pp. 169–180 (2006)
25. Pesic, M., Schonenberg, M.H., Sidorova, N., Van Der Aalst, W.M.P.: Constraint-based workflow models: change made easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
26. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 237–252. Springer, Heidelberg (2011)
27. Winskel, G.: Event structures. In: Advances in Petri Nets, pp. 325–392 (1986)
28. Hull, R., Damaggio, E., Masellis, R.D., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS, pp. 51–62 (2011)
29. Wiki: SOA 2.0, http://en.wikipedia.org/wiki/Event-driven_SOA
30. Chau, T., Muthusamy, V., Jacobsen, H.A., Litani, E., Chan, A., Coulthard, P.: Automating sla modeling. In: CASCON 2008, pp. 126–143. ACM, New York (2008)
31. Muthusamy, V., Jacobsen, H.A.: BPM in cloud architectures: Business process management with SLAs and events. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 5–10. Springer, Heidelberg (2010)

32. Muthusamy, V., Jacobsen, H.A., Coulthard, P., Chan, A., Waterhouse, J., Litani, E.: Sla-driven business process management in soa. In: CASCON 2007, pp. 264–267. ACM, New York (2007)
33. OSOA: SCA event processing, http://www.osoa.org/
34. Papazoglou, M.P., Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16(3), 389–415 (2007)
35. Yan, W., Hu, S., Muthusamy, V., Jacobsen, H.A., Zha, L.: Efficient event-based resource discovery. In: DEBS 2009, pp. 1–12. ACM, New York (2009)
36. Frei, A., Popovici, A., Alonso, G.: Eventizing applications in an adaptive middleware platform. IEEE DSO 6(4), 1 (2005)