On the Fly Service Composition for Local Interaction Environments

Hossein Pourreza and Peter Graham
Department of Computer Science
University of Manitoba
Winnipeg, MB, Canada R3T 2N2
and TRLabs, Winnipeg
{pourreza, pgraham}@cs.umanitoba.ca

Abstract

Dynamically creating new, composite services "on the fly" using existing ones in a local interaction environment (e.g. a home, meeting room, airport lounge, etc.) presents challenging problems. In this paper we describe a middleware model for such service composition targeted, initially, to a home area network (HAN) scenario. Being able to automatically synthesize new and useful services in a HAN (or other environment) without user intervention makes the use of the network simpler and more attractive for non-expert users (e.g. home owners). We propose a service composition model which involves third-party "service providers" (SPs) in the composition process thereby allowing the discovery of services without direction from the users as to what type of service is desired. We also discuss our experiences with our initial prototype system where ontology based matching is done by the service providers and the resulting composite service is deployed as a workflow using available in-home protocols (e.g. UPnP, Jini, etc.).

1. Introduction

Current technology now enables a multitude of consumer equipment (e.g. personal entertainment devices, home appliances, etc.) in addition to computers and PDAs to be connected [1]. Further, the number of homes and other local interaction environments (e.g. meeting rooms, airport lounges, shopping malls, etc.) with broadband (often wireless) connection to the Internet is increasing [11]. This offers the possibility of involving more powerful, Internet based, third party service providers (SPs) in dynamically and automatically creating new services based on those already offered by the devices present in a given environment.

[1] Such connection may be either direct or using a proxy device and may use either wired or wireless infrastructure.

Although the work presented in this paper is applicable to a wide range of local interaction environments, since our prototype is focused on use in Home Area Networks (HANs), we now focus our discussion primarily on HANs.

A Home Area Network [18] is comprised of several networked devices and appliances connected to each other. A networked appliance "is a dedicated-function consumer device with an embedded processor and a network connection". The connection between an in-home network and the Internet is usually provided by a *home gateway* (a.k.a. *residential gateway*) device (HGD) which is roughly analogous to a wireless access point though it may offer storage and other services as well as Internet connectivity. There may also be several service providers which offer different types of services (e.g. phone, Internet, security monitoring, remote device management, etc.) to the home. Figure 1 shows a simple HAN and some of its components.

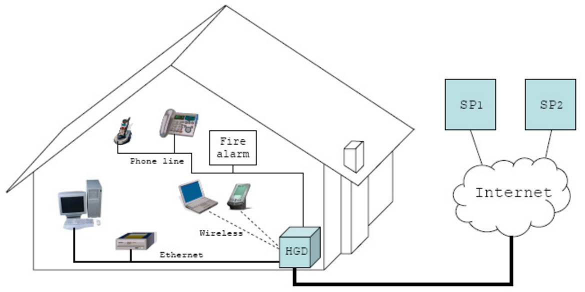

Figure 1. A simple HAN environment

Due to the variety of home devices, communication media, and manufacturers, a HAN can be a highly heterogeneous environment. This makes the integration of multiple devices challenging. Further, most home owners have low technical knowledge and are generally unwilling to be actively involved in maintaining and upgrading their HAN (characteristics also true of other local interaction environments). Further, with the increasing number of networked homes and devices in each home, distributing new services and upgrading existing ones has become a cumbersome and expensive task for service providers.

Service lookup and discovery protocols such as UPnP [5] and Jini [6] enable service consumers (client programs that need a service) to find their required services. However, different protocols are typically not inter-operable and this can cause a problem in integrating new devices into a HAN. Further, a device by itself is seldom as useful as when its functionalities can be combined with other devices. (e.g. A Personal Video Recorder could be implemented by recording a program that a user is known to regularly watch from a networked TV onto available disk space on a home computer for later viewing if the home owner is away during the broadcast).

In a HAN (or similar) environment, it must be possible to integrate new devices with existing devices/services[2] in a transparent fashion. This requires many issues to be addressed including service discovery and service composition. Consider, for example, a scenario where a home owner is in the basement watching TV. The security camera, installed at the front door, is activated when the doorbell rings and, using stored context about where the homeowner is, sends its images to the basement TV. The home owner can then choose to go to the door or not. Such a composite service will be valuable to the homeowner (and marketable by a service provider) if it can be offered using existing HAN devices and implemented without user intervention.

Service composition is a technique that creates new compound services using existing ones and can be used to assist non-expert users in integrating new devices into their homes. Service composition, generally, and the composition of Web services, in particular, has been studied extensively in the literature (e.g. [21, 14]) but relatively few papers (e.g. [16]) have considered this issue for a pervasive computing environment such as a home.

Most research on service composition assumes a known workflow for a desired composite service and tries to discover compatible service components and compose them in the required, pre-defined order. Some other research (e.g. [21, 19]) has tried to solve the service composition problem (in particular workflow generation) using logic-based languages. Their primary goal is to generate a semantically-correct workflow but, again, this is based on an existing service request from the user. Unfortunately, given the limited technical expertise of a typical homeowner, assuming a pre-defined "goal" service limits the applicability of these methods.

We are trying to address service composition in the local interaction environments (including homes) by involving third-party service providers in the composition process. Service providers have enough expertise to create generic workflow specifications for potentially useful composite services which can then be used to drive the service component matching process based on the available devices and services in the home. In our proposed model, the HGD provides information about the devices in the home to a service provider (e.g. when a new device is discovered). The service provider then uses ontological matching to see if any new composite services may be created using the new and existing devices and services in the home. If new composite services are found, the SP sends the required workflows to the HGD to deploy. The existence of the new services can then be advertised to the home owners in whatever way is best suited to their technical abilities.

The rest of this paper is organized as follows. Section 2 reviews related work. Our proposed middleware model is explained in Section 3. Section 4 describes our experience with a prototype implementation. Finally Section 5 concludes the paper and discusses directions for future work.

2. Related Work

There are two basic service composition methods [10]:

- semantic-based methods: which try to use semantic information about services to create a logical workflow corresponding to a user's request.
- architecture-based methods: which assume an existing composite workflow and try to execute it subject to some user-defined constraints (e.g. QoS constraints).

2.1. Semantics-based Service Composition

Systems in this category try to create a workflow for a composite service. They normally use a language such as OWL-S (Ontology Web Language - for Services) [7] to describe the semantics of each service. An OWL-S service description is composed of:

- a service profile: which explains what a service does in terms of its inputs, outputs, preconditions, and effects.
- a service process: which explains how a service works by showing the sequence, iterations, decisions, etc. The service process glues together the service profile and the service grounding (described next).
- a service grounding: which explains how to access the service. The service grounding describes where the implementation of the described service can be found and what protocol should be used to invoke it.

OWL-S provides a separation between a service's description and its implementation. This separation means

[2]In this paper we use the terms "service" and "device" interchangeably.

that device manufacturers as well as third party service creators can easily provide semantic descriptions for new services based on a variety of devices. Also, it means that a service description can have multiple implementations depending on where the service is going to be deployed (and what devices it will be implemented on).

Sirin et al [21] proposed a semi-automatic solution for Web service composition. Their web services are described in OWL-S and, in their system, a user is given a list of available services to start the service composition process. Once the user selects a service, the system determines its input(s) and shows the user a list of services whose output(s) can be fed to the input(s) of the selected service so a selection can be made. This process continues to construct a composite service. Finally, the generated composite service, expressed in OWL-S, can be executed using Web services technologies (e.g. Web Services Definition Language (WSDL) [4]). Unfortunately, Sirin et al's method requires unacceptably high user interaction with the system and is therefore inappropriate for our application environment.

Rao et al [20] developed a system which accepts a user's request in DAML-S [12] and translates it into Linear Logic (LL). An LL theorem prover is then used to find a sequence of available services which are input/output compatible and which, collectively, fulfill the user's request. The result of composition can be described in DAML-S but the system is not capable of running the composite service. This method again requires a user to define the needed service's inputs and outputs which is not practical with non-expert users.

Another system, proposed by Majithia et al [14], stores two types of workflows (*abstract* and *concrete*) to increase re-usability. An abstract workflow does not have any information about service instances and is high level. A concrete workflow, on the other hand, is a mapping from an abstract workflow onto available services. This system is similar to Rao et al's but a different method is used for matchmaking and reasoning purposes. This system also requires extensive user interaction. The idea of using separate abstract and concrete workflows, however, is applicable to our work.

2.2. Architecture-based Composition

Architecture-based service composition assumes an existing workflow (possibly created using semantics-based composition) and tries to execute it.

eFlow [8] is an e-service composition system developed at HP labs. The specification of a composite service is given by the user and eFlow models the composite service as a graph [9]. Service nodes in the graph contain "service selection rules" specifying the desired service's characteristics. An XML-based language is used to describe each service. A "service broker" is responsible for discovering services matching the given descriptions. Once all the services in the graph have been discovered, the composite service is executed by the e-Flow composition engine.

Ninja [13] from the University of California, Berkeley, provides a similar service composition platform. Ninja defines *paths* which chain different *operators* using *connectors*. Operators are services doing computation on passing data flows. Connectors abstract away the details of underlying communication links and connect operators residing on different nodes. To create a path, a user provides a specification of the endpoint nodes in the path, a partially-ordered list of operators that must be included in the path, and some cost-related criteria that the path should satisfy. Once the characteristics of a path are defined, Ninja tries to create a path that meets the user's criteria.

Task Computing [16, 17], which is the closest system to our proposed model, tries to fill the gap between a task (what a user wants) and a service (what a device provides). Task Computing combines Semantic Web and ubiquitous computing to try to help non-expert users exploit the capabilities of a device rich environment. Task Computing uses UPnP to discover available devices and OWL-S to describe services for composition. Task Computing requires each participating device to have the Semantic Task Execution EditoR (STEER) [17] installed. STEER provides a user interface showing the available services (using UPnP) and allows a user to select one of a number of possible compositions. STEER has an inference engine which is used to do the service composition. Unfortunately, requiring all participant devices to install STEER is undesirable and, again, users are explicitly involved in compostion.

3. Our Middleware Model

In this section we overview our proposed model for service composition. The goals in designing our system were to simplify the composition task for users, to remove its computational burden from resource-limited devices, to avoid changes to existing HAN standards (e.g. discovery protocols, service description languages, etc.) and to require at most minimal changes to a home's infrastructure. In our prototype, we used standard lower level middleware (e.g. UPnP and Jini) and languages (e.g. OWL-S) for service composition and deployment.

We assume a HAN connected to the Internet via an HGD. We also assume that the HGD runs the Open Services Gateway initiative (OSGi) [2] framework which helps support interoperability between various HAN protocols.

OSGi provides a Java-based framework in which different services, offered by different devices, can be integrated. For portability and scalability reasons, the implementation of a service is separated from its interfaces. Thus, depending on the environment in which a service is being deployed, different implementations may be chosen.

An OSGi *bundle* [15] is a package for service implementation that contains resources for implementing services as well as a *manifest* of headers that specify required parameters for installing the bundle. Our prototype is implemented as a set of OSGi bundles.

In-home devices can, use any available protocols (e.g. UPnP, Jini, HAVi, bluetooth, etc.) to share their services. OSGi, running on the HGD, is used to provide interoperability between the different protocols. For our prototype, we have extended our base OSGi distribution to support UPnP and Jini services calling one another directly (not as separate OSGi services). This provides additional flexibility in service composition.

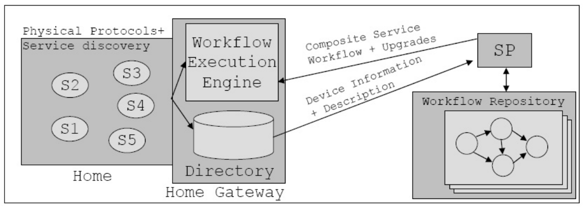
Physical Protocols+
Service discovery
S2
S3
S4
S1
S5
Home
Workflow Execution Engine
Directory
Home Gateway
Composite Service Workflow + Upgrades
Device Information + Description
SP
Workflow Repository

Figure 2. High level composition model

Figure 2 shows our high-level model for service composition. Each service (S_n in the figure) carries its semantic description encoded in OWL-S. This description has enough information to support input/output matching required for service composition. The OWL-S semantic descriptions for devices are assumed to be available online (e.g. from device manufacturers' web sites) based on information retrieved from devices when they are discovered. Service composition is performed by the SP using information sent by the HGD and the execution of composite services in the home is overseen by the Workflow Execution Engine. In our model, services have associated OWL-S service profiles and processes. We dynamically create the OWL-S service grounding based on the implementation environment (as described later).

Although our model is similar to Task Computing [16], there are major differences between the two:

- Our model moves service composition from the home to the service provider. This off-loads the composition overhead from the home and also eliminates the need for user involvement in service composition. Additionally, devices participating in service composition do not need to run any special software, such as STEER.
- Using OSGi-based groundings to access service components instead of a single protocol provides flexibility in the variety of protocols that can be used.
- Our prototype uses UPnP and well as Jini for advertising the newly created composite services.

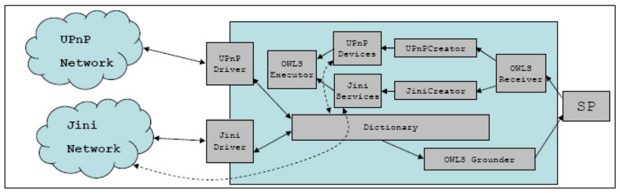

Figure 3. Components inside the HGD

In our middleware, we use an OSGi-based grounding to access a service. Using OSGi for service grounding allows us to support heterogeneous composite services. In other words, a component of a composite service may be provided by an existing Jini service while another component can be provided by a UPnP (or other) service.

The components running (as OSGi bundles) inside the HGD ("Workflow Execution Engine") are shown in Figure 3. In our prototype, a UPnP or a Jini service is registered in OSGi using the corresponding (UPnP or Jini) OSGi drivers. The grounding for each service is then created and is associated with the semantic description of the service (i.e. the OWL-S service profile) that is carried with it. Once the total service description (profile, process, and grounding) has been created, it is sent to the service provider (SP) (Figure 2) where a search is made to try to find a possible composition. The searching uses the received service information from a given home to match the available devices and services against those found in the "Workflow Repository". If a composite service is found, its description, encoded in OWL-S, is sent to the HGD for execution by the "Workflow Execution Engine".

The Service composition that must be done by the service providers could, in general, be done using either or both of the following approaches:

- A repository of workflows could be used to match permutations of the received service descriptions against pre-defined workflows. If all components of a workflow match the received service descriptions, that workflow, presented in OWL-S, would be sent to the HGD. This approach requires no human intervention except for creation of workflows in the repositories.
- The service provider could, alternatively, try to find a sequence of received service descriptions that are input/output compatible. This process, however, would require human involvement to determine which compositions are actually useful. (This person would be a technically adept SP employee, not a naive home owner.) Again, the result of the composition would be sent to the HGD.

In our prototype, both matching of existing workflows in the repository and dynamic creation of workflows based on I/O compatibility are done. Matching is based on the OWL-S information carried in each service description and is done generically (i.e. with "abstract" service descriptions) based on the OWL-S profiles. Once an "abstract" workflow has been matched, the corresponding OWL-S groundings are used by the Workflow Execution Engine to tie the workflow components to the existing, in-home component services. Using our earlier scenario as an example, the HGD will send OWL-S descriptions of the doorbell, camera and TV to the SP. The SP will match a sequence of "doorbell+camera+TV" using either I/O matching or searching in the workflow repository. Once found, this sequence will be sent to the HGD.

Once the HGD receives a composite service description from a service provider, (e.g camera+TV to show a security camera image on a TV) it creates corresponding wrappers for all protocols in the home and registers them with OSGi. A wrapper is an OSGi service created from a received OWL-S description of a composite service. Each wrapper is capable of calling sub-components of its composite service grounded to OSGi. Once a new service is registered, the appropriate driver (e.g. UPnP) will export it so other devices within the home are notified of the service's existence and can subsequently make use of it. The components of a composite service (doorbell, camera and TV) also have OSGi groundings and hence can interact with each other. In our example, the doorbell triggers the camera to capture images which are passed to the display service to show them on the TV. For services involving direct user interaction, a mechanism to advertise the existence of the services (e.g. user interface widgets) might also be desirable (even though UPnP services are accessible through their control points). This has not yet been implemented in our prototype.

To clarify this process, assume we want to install a new UPnP device in a home. The steps that are done by our system (refer to both Figures 2 and 3) are as follows:

1. The UPnP device announces itself (via UPnP) and OSGi's UPnP driver registers the device's services in the OSGi dictionary.

2. The "OWLGrounder" bundle creates an OSGi grounding for the new device, attaches the grounding to the OWL-S description of the device and sends a link to the complete OWL-S file (profile, process and grounding) to the service provider (SP).

3. The SP tries to match the received information against existing workflows in its repository (including previously composed services). If no match is found, the SP attempts to create a new workflow by matching the outputs of existing services to the inputs of others.

4. The SP sends any matched or created workflows to the OWL-S receiver bundle in the HGD.

5. The OWL-S receiver bundle uses the OWL-S descriptions to create UPnP wrappers for the new services.

6. The created UPnP wrappers are then registered inside OSGi so that the UPnP driver will detect them and announce them to the UPnP network.

7. The newly created composite services then become available to the UPnP network.

8. When a UPnP control point calls a method on one of the new UPnP services the method call is received by the UPnP driver and it passes it to the UPnP wrapper.

9. Finally, the UPnP wrapper uses the "OWLExecuter" to execute the composite service and returns the result to the UPnP driver and then to the UPnP control point

4. Experience with the Prototype System

Most of the model described in Section 3 has been implemented except for certain details of the composite workflow repository. We have implemented all the HGD components (shown in Figure 3) using the Oscar [1] implementation of OSGi. We found Oscar to be effective in managing the components required for service composition using our middleware. We have also added OSGi groundings to the OWL-S API [3] so each service defined by OWL-S can be accessed within OSGi. Further, we have tested the system using a number of emulated UPnP devices and several sample Jini services which are fully functional.

The implemented prototype can make effective use of both UPnP and Jini services registered within OSGi as components of composite services. We have experienced no difficulties in discovering and creating basic (linear) composite services using UPnP and Jini services but have yet to build the required system components to permit complex composite services (i.e. those with concurrent and conditional service components) to be executed. All created composite services are also registered as OSGi services and can therefore be used in subsequent service compositions . Additionally, the deployment of composite services as UPnP or Jini services (as well as OSGi services) is implemented. This allows other (non-OSGi) services in a HAN to seamlessly make use of the newly created composite services.

Although the prototype system is relatively small, initial results suggest that the workload placed on a moderately powerful embedded HGD should be small enough to permit effective implementation for homes containing a relatively large number of devices . Our initial evaluation of service matching has shown that the cost of OWL-S lookups is significant and could limit performance in applications where many devices are frequently added/removed. No detailed performance or scalability assessment of the matching process for service composition has yet been done.

5. Conclusions and Future Work

In this paper we presented a middleware model for service composition in HANs that is also applicable to other local interaction environments such as classrooms and conference halls. Unlike other systems for service composition, we involve a third party . This approach offers several advantages. First, the service provider performs the bulk of the computationally expensive service component matching thereby offloading this effort from resource-limited devices that will typically exist in a HAN. Providing new services "on the fly" as devices are added to a HAN also makes their use more attractive to technically unsophisticated home owners. (Our service composition model allows a home owner to exploit his/her newly added devices immediately and without being involved in the process of integrating the devices into the home.) Additionally, by treating service upgrades such as the downloading of new device software as a special case of service composition, our approach also mitigates maintenance problems for home owners and provides cost benefits to SPs who will need to make fewer service calls to homes. Further, using our approach, a service provider can take a workflow created for one home and add it to its repository thereby reusing compositions to reduce future composition overhead. Finally, deploying new services using all existing in-home protocols provides for the seamless integration of new composite services with existing ones.

Our future work includes extension of the SP side processing to support matching and generation of composite services including concurrency and conditional execution as well as the creation of GUI widgets to announce the availability of interactive composite services to users. Longer term work revolves around developing an efficient workflow repository to permit the creation of large scale SPs and a detailed performance analysis of SP side processing. We are also in the process of adding a push mode for service delivery where service providers can react to the provision of new code by third parties (e.g. device manufacturers) that results in new composite services not driven by the availability of new in-home devices. In this model, SPs periodically identify homes that will benefit from new services and "push" the required workflows to those homes.

References

[1] Oscar: An OSGi framework implementation. http://oscar.objectweb.org/.

[2] The OSGi homepage. http://www.osgi.org.

[3] The owl-s api. http://www.mindswap.org/2004/owl-s/api/.

[4] Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.

[5] Universal plug and play device architecture. http://www.upnp.org/download/UPnPDA10_20000613.htm, June 2000.

[6] Jini Architecture Specification v2.0, Sun Microsystems. http://wwws.sun.com/software/jini/specs/jini2_0.pdf, 2003.

[7] http://www.daml.org/services/owl-s/, 2004.

[8] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eFlow. Technical Report HPL-2000-39, HP Labs, 2000.

[9] D. Chakraborty and A. Joshi. Dynamic service composition: State-of-the-art and research directions. Technical Report TR-CS-01-19, University of Maryland, 2001.

[10] D. Chakraborty, Y. Yesha, and A. Joshi. A distributed service composition protocol for pervasive environments. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2575–2580, Mar. 2004.

[11] P. Dobrev, D. Famolari, C. Kurzke, and B. A. Miller. Device and service discovery in home networks with OSGi. *IEEE Communications Magazine*, 40(8):86–92, Aug. 2002.

[12] A. A. et al. Daml-s: Web service description for the semantic Web. In *International Semantic Web Conference*, pages 348–363, 2002.

[13] S. D. G. et al. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.

[14] S. Majithia, D. W. Walker, and W.A.Gray. Automated Web service composition using semantic Web technologies. In *International Conference on Autonomic Computing (ICAC-04)*, pages 306–307, May 2004.

[15] D. Marples and P. Kriens. The open services gateway initiative: An introductory overview. *IEEE Communications Magazine*, 39(12):110–114, Dec. 2001.

[16] R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *IEEE intelligent Systems*, 18(5):68–72, 2003.

[17] R. Masuoka, B. Parsia, and Y. Labrou. Task Computing - the semantic Web meets pervasive computing. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, pages 866–881, 2003.

[18] G. O'driscoll. *The essential guide to home networking technologies*. Prentice Hall, Upper Saddle River, 2001.

[19] J. Rao, P. Kungas, and M. Matskin. Logic-based web service composition: from service description to process model. In *Proceedings of the IEEE International Conference on Web Services*, pages 446–453, July 2004.

[20] J. Rao and X. Su. A survey of automated web service composition methods. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[21] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic Web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.