Dynamic Service Discovery and Management in Task Computing

Zhexuan Song   Yannis Labrou   Ryusuke Masuoka
Fujitsu Laboratories of America
8400 Baltimore Avenue, Suite 302
College Park, Maryland 20740-2496, USA
{zsong, yannis, rmasuoka}@fla.fujitsu.com

Abstract

Task Computing ([4, 5]) enables a user to compose and execute complex tasks in application-, device- and service-rich environments. Task Computing is possible through the availability of semantically described services that, thanks to their semantics, can be composed on-the-fly by end-users into executable tasks. Through the use of a Task Computing Client, users can not only compose tasks from available semantically described services, but discover, create, manage and manipulate services as well.

The focus of this paper is the technologies used for dynamic service discovery, and creation, management and manipulation of semantically described services.

1. Introduction

Task Computing [4, 5] is a user-oriented framework that lets non-expert users accomplish complex tasks in application-, device-, and service-rich environments. Task Computing provides myriads of ways for the users to interact with and through ubiquitous environments.

Some examples of the complex tasks that can be defined and executed (both in real time) in our Task Computing demonstrations are:

- Forwarding of a security video (or of an Internet TV video stream) to any of a number of display devices (TV, computer monitor, etc., or a friend's TV) without manually connecting cables.

- Dialing the work number of a contact from your Personal Information Management (PIM) using the phone in a conference room you are visiting for the first time.

- Displaying on your browser the current weather information at the location of a contact from your PIM, or showing and printing the driving directions from your current position to that location using a kiosk at a shopping mall.

- Giving a presentation on a projector in the conference room from your own computer or PDA (without connecting the VGA cable) and then depositing your presentation file in the room and let other people, who later come into the room, to view or copy the presentation.

Some of these scenarios are, of course, possible with error-prone and time-consuming processes involving endless cutting and pasting and invocation of multiple applications by computer-savvy users. Yet, other scenarios would be impossible without additional custom software development. Task Computing makes all of the above scenarios possible with a few point-and-click operations, and most importantly through a simplified and easily manipulable view of the universe of possible actions, at any given point (temporally and spatially). Furthermore, Task Computing makes all of these (and many more) tasks possible without explicitly programming for them [4, 5].

Figure 1 illustrates the Task Computing perspective. In architectural terms, Task Computing is comprised of four distinct layers (reminiscent of a typical 3-tier architecture).

- [REALIZATION LAYER] The bottom most layer encompasses the universe of devices, applications, e-services and content, where all functionality available to the user originates.

- [SERVICE LAYER] These various sources of functionality are made computationally available as services, in the sense that service interfaces are employed to access (execute) this functionality. Each service is associated with at least one semantic description, which sometimes may be created on-the-fly as services might be created dynamically. Services are the abstraction of functionality in the Task Computing universe, and semantic descriptions of these services are meant to shield the user from the complexity of the underlying sources of functionality and make it easy (hopefully trivial) for the user to employ these sources in accomplishing interesting and complex tasks.
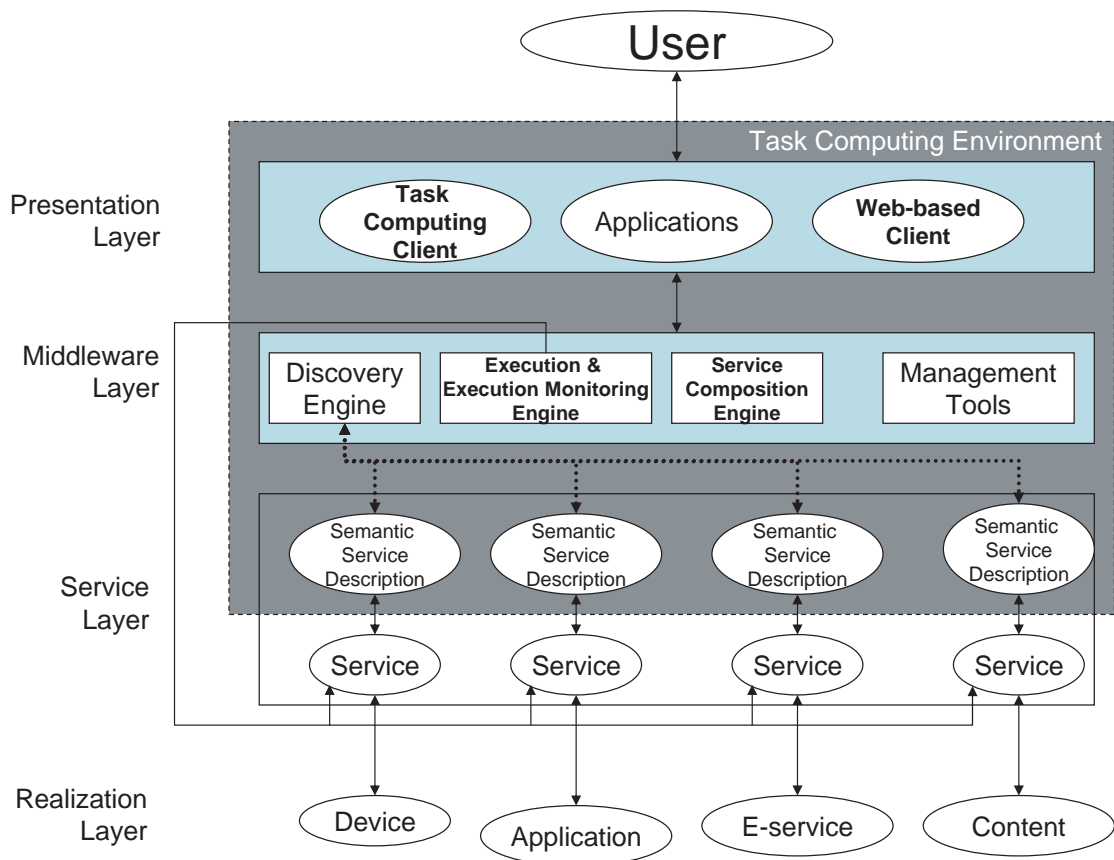
Figure 1. General architecture of Task Computing Environment

- [MIDDLEWARE LAYER] These lofty goals are enabled by the middleware layer components that are in charge of discovering services, deciding how services can be composed, executing services and monitoring service execution, and finally enabling and facilitating a variety of management tasks, including the creation and publishing of semantically described services.

- [PRESENTATION LAYER] The most important aspect of Task Computing is the presentation layer, which uses the capabilities of the layers below in order to provide the user with a "Task" abstraction of the complexity of whatever lays underneath. We have developed a variety of clients for that purpose, such as voice, textual and graphical interfaces, (referred to as Task Computing Clients (TCC's)), and a web-based interface (utilizing a web browser). The presentation layer presents to the user an environment where functionality that can be transient and dynamically created can be assembled (in real time) to perform users' tasks. Since the middleware layer components expose well-defined service API's, it is possible to create custom applica-

tions in the presentation layer in any development environments that can invoke Web Services.

The separation of these layers is both logical and implementational. We have found it useful for building an environment where the user can perform complex tasks that have not been (neither implicitly nor explicitly) designed into the system, thus multiplying the uses of the sources of functionality (devices, applications, content and e-services). Technically speaking, Task Computing is possible because of the availability of *dynamic service discovery, service publishing and management, task creation, and execution of tasks on-the-fly* [9]. We have discussed the role of the semantics and the details of task creation and execution in [5]. In this paper, we will focus on the first two issues, namely dynamic service discovery, and service publishing and management.

The rest of this paper is organized as follows: In Section 2, we will discuss the challenges that we have to address; in Section 3, we present in detail our approach and technical solutions to the identified challenges and in Section 4 we evaluate our approach with respect to related work and dis-

cuss our future plans and work. Finally, Section 5 concludes the paper.

2. Insights and Concepts

Task Computing depends on and uses Semantic Web, web services, and ubiquitous computing technologies. But, in order to deliver a real, functioning system in a truly dynamic and ad-hoc ubiquitous computing environment, we needed to establish and implement the following:

- Complete separation of semantic service descriptions (SSD's) and service implementations

- Separation between discovery mechanisms and discovery ranges and manipulation capability of services within and between those ranges

- Ability for users (and services) to dynamically create and manipulate services that can be made available and shared with others (or made unavailable when necessary)

- A variety of services that enable interesting and truly useful tasks

Services are abstractions of functionality that is of interest to the user. Such functionality generally emanates from at least three different types of sources: devices, applications and over-the-web e-services. These three sources are loosely defined categories, as the boundaries between these categories are highly malleable. Broadly speaking, device-originating services are associated with the core functionality that the device is designed to deliver. For example, a phone's (device) main functionality is making phone calls (service). Similarly, application-originating functionalities are associated with a computing application that is executing on a computing device. For example a PIM application's functionalities, includes storing and retrieving contact information of persons. Finally e-services functionality is associated with web services that are executing on some remote server and deliver the functionality through access to the web, beyond the boundaries of a user's local network. Occasionally a fourth source of functionality is very useful, namely content that is made available as a service; this type of service has proven practically very useful as an information-sharing mechanism between users. This broad categorization is only offered as rough way to distinguish between the nature of services, as service may belong in multiple categories. It is worth noting that in Task Computing the origins of such functionality is transparent to the user as the user is mainly interested in combining such functionality for the purposes of defining a task.

Service Discovery essentially refers to discovery of the Semantic Service Description (SSD) of a service. From a TCC's perspective, only the SSD's of services matter since the SSD includes enough information for the users to manipulate the services. This separation of the actual implementation of a service and the metadata of the service (in the form of semantics) means that any party can provide SSD's, not just the creator of the service. On the other hand, since services in a typically dynamic ubiquitous environment are highly transient, it is important that the availability of the SSD is consistent with the availability of the service.

The possible tasks that a user can accomplish with their TCC depend on the variety of functionality that is made available to the user. The different types of services we mentioned require different discovery ranges. Services running on a user's device should be made available to at least the user of the device (even though the user may choose to temporarily disable them). In ubiquitous computing, though, the subnet that a user's device is connected to, offers a logical localization mechanism, as such functionality may be made available to all users in that subnet. On the other hand, communities of users (identified by shared interests, organizational membership, etc.) should be able to share access to a set of services intended for that community. Finally, there are services that are intended for general use and any user on the network should have access to them (being able to discover them and execute them). The distinction between service types (devices, applications, over-the-web e-services, content) is orthogonal to the discovery ranges just discussed. The latter can be thought of as layers of permissions in a file system (user, group, world)[1] and "permissions" refers to the ability to discover and execute a service.

The notion of ownership applies to managing (modifying) various features of the service, including its availability. We have recognized that it is important that users can create new services on-the-fly and use them as means to share information with other users, to enable users to access functionality that might have been previously unavailable to them, or in order to make functionality available to themselves and other users. In some cases a user will create a new web service with associated semantic descriptions, while in others, the user is creating, sharing, or just making available a semantic service description, which is exactly what TCC's require in order to effectively utilize a service. In each of these cases, it is important that the user has control over the created services (or the availability of a semantic service description[2]), such as making them visible or invisible, determining who can discover (and subsequently execute) them, determine their temporal availability, and so on.

---

1 where "group" bears two "meanings" (subnet and community)
2 The distinction between an actual service and a semantic service description becomes unimportant to the user in a Task Computing Environment and is not reflected in the user's UI.

3. Service Discovery and Management in Detail

In this section, we present in detail our approach towards addressing the challenges discussed in Section 2, and in particular, the technical details for service discovery and management in Task Computing.

3.1. Separation of Service Implementation and Semantic Service Description

We have implemented an environment with complete separation of service implementation and corresponding semantic description. As a result, *users can discover services, construct, and save tasks no matter where the services (or tasks, i.e., service compositions) reside (locally on the user's device, ubiquitously on the subnet, or remotely on the Internet) or which technology the services are implemented in.* The whereabouts and the implementation details of services are of no concern to users while they create tasks and manipulate services in the semantic layer.

Two kinds of service technologies are used for implementations of services in the Task Computing Environment (TCE), namely, web services and UPnP [12]. Both web services and UPnP services use SOAP [10] (Simple Object Access Protocol) for message encoding and usually HTTP (Hyper-Text Transfer Protocol) for message transport. Both of them define XML-based interface descriptions, namely WSDL [13] for web service and Device Description for UPnP. However, those interfaces do not contain sufficient semantic information for end-users to manipulate them on-the-fly.

Our solution is to use SSD's to describe services at the semantic level. The SSD's in Task Computing are encoded in OWL-S language [8], which is a description language for semantic services based on OWL [7]. We choose OWL-S for SSD's in TCE over extending WSDL because a standard-based service description with built-in metadata supports is essential for realization of separation of service description and implementation. The SSD consists of three parts: profile, process, and grounding.

- The **profile** part provides the high-level descriptive information of a service, including the name, the text description, semantic input and output of the service. We also have additional parameters such as links to the multimedia descriptions (in video, image, icon, audio, etc.) and a user interface web page for users to interact with the service when necessary.

- The **process** part describes the semantic level description of the process of the service and glues the profile part and the grounding part.

- The **grounding** part is the most important part for the realization of the separation. It describes where the service implementation (in web services or UPnP) can be found and how the semantic objects in the process part and the parameters in the service implementation are mapped to each other.

Note that the profile part allows users to manipulate services in the semantic layer and the grounding part allows users to actually invoke services. The relationship between service implementations and SSD's is one-to-many, namely, one service implementation can have multiple SSD's. This design allows the developers to reuse the same Web Services in many different contexts, simply by providing additional SSD's.

Since SSD's are just text files in OWL-S, they can be moved, copied, emailed, or downloaded from a Web page. In the rest of the paper, we use "discovery of a service" and "discovery of the SSD of a service" interchangeably.

3.2. Service Discovery

We define service discovery as the process of *finding services related to a user's context*. Technically, given the separation of service implementation and SSD, discovery is reduced to the acquisition of the SSD's of services by TCC. The implementation of service discovery relies on one or more *discovery mechanisms*; a TCC can exploit multiple service discovery mechanisms and a particular service might be discoverable through multiple discovery mechanisms.

Users, or the services (or their providers) may set the discovery mechanism employed for the discovery of a particular service. Changing the discovery mechanism for a service, may affect who can and where one can discover a service. Although service discovery mechanisms are orthogonal to discovery ranges, we have found that some discovery mechanisms are more suited for a specific discovery range than others (see Table 1). We next elaborate on each discovery range.

[Empty] Services in *empty* discovery range are those that can not be discovered by anyone. *Empty* is not an entirely conceptual range; any service that is made unavailable (even for its owner) may assume this range. For example, a user does not want the service providing her contact information discovered by others due to privacy considerations, or even by herself because it is annoying to always discover a service that she does not intend to use, and thus may choose the *empty* range for this service. When, later, she wants to use her contact for displaying on a kiosk the route from the airport she is at to her home, she may move the service into the *private* discovery range.

[Private] Services in the *private* discovery range are discoverable only by their owner and typically reside on the

Discovery Range	Example Discovery Mechanism
Empty	N/A
Private	File system based discovery
Group by Subnet	Multi-cast based discovery
Group by Interest	Community directory, publish /subscribe (company, community)
Public	Open semantic service directory

Table 1. Service discovery ranges and corresponding discovery mechanisms

user's own computing device which runs the TCC. For example, the local resource handling services such as *My File*, which lets the user select and expose a file on her device, assumes (by default) this discovery range. We rely on a file system-based discovery mechanism combined with notifications using sockets to implement this discovery range.

[Group by Subnet] The *group by subnet* discovery range is most closely related to ubiquitous environments because of its ad-hoc and spontaneous nature of grouping. Services that happen to be on the same subnet as the user, such as a part of a company Intranet or a home network, will be discovered, enabling a very localized discovery mechanism.

We use UPnP as the discovery mechanism to implement this range. Specifically, UPnP's discovery mechanism is used to find the UPnP devices on the subnet (not all of which are Task Computing-enabled services) and for each UPnP device, the TCC invokes one specific UPnP action (*getDescriptionURL*) to determine if the UPnP device represents a Task Computing-enabled service and if so, the TCC proceeds to download the SSD from the UPnP device. Other discovery mechanisms such as JINI [2] can also be used in the same way as UPnP to implement this discovery range.

[Group by Interest] This *group by interest* discovery range refers to services discovered by any arbitrary group of people, perhaps bound by similar interests or group membership, such as the group of employees of a company or the members of a golf club. Currently we do not have a particular discovery mechanism for this discovery range, but we plan to implement it by combining web services with callbacks and polling mechanisms.

[Public] Services in this discovery range can be discovered by anyone. A good discovery mechanism for this range is an open semantic service directory; examples include Web pages with links to SSD's of publicly available services (a variant of the early days of Yahoo!), or a search engine for semantic web services like UDDI [11]. Alternatively, users can share the SSD's by emailing them to each other, or by sharing them over a peer-to-peer network.

3.3. Service Publishing and Sharing

We have created two tools, namely White Hole and PIPE (Pervasive Instance Provision Environment), to support the dynamic creation of services (service publishing) and their dissemination (sharing). Technically speaking, these two tools are used to semantic-ize, service-ize and publish (information) objects and services (see also Figure 2).

One of the scenarios enabled by the White Hole and PIPE is as follows. A user finds a very good restaurant and creates a contact entry of the restaurant in her PIM. She drags and drops the contact entry into the White Hole to create a contact providing service in PIPE. Later at a shopping mall, she uses the service to show the route from the mall to the restaurant and prints the map. Or at her friend's house, the friend invokes the service along with "Add into PIM" service to save the restaurant contact in his PIM.

White Hole The White Hole component has a convenient drag-and-drop interface for operating system or application objects (such as files from the local machine's operating system, contacts of PIM application, etc.), semantic objects in OWL format (or URL to the OWL file), and semantic service descriptions in OWL-S format (or URL to the OWL-S file). When something is dropped into the White Hole, the tool first decides its type: (a) if it is an OWL or OWL-S object, the White Hole just passes it to PIPE (discuss next); (b) if it is a URL to a OWL or OWL-S file, the White Hole downloads the content of the URL and passes it to PIPE; (c) if it is a known (semantically speaking) OS/application object the White Hole semantic-izes the object (see Table 2)

Semantic-ization is the process of creating a semantic object from an OS/application object. We support approximately ten types of OS/application objects (such as file and URL from OS, contact and schedule from PIM application, etc.). White Hole determines the semantic type of objects by their name, extension, and content. Once the type is determined, an OWL template for the type is retrieved and filled with the values extracted from the original object. Then the OWL description of the object is generated and passed on to PIPE. For example, if a user drops a contact item from a PIM application, the White Hole first loads the OWL template for the contact type, then, retrieves the name, company, email, phone, etc., from the contact item, and fills them into the template. Finally the complete OWL object is passed on to PIPE.

PIPE PIPE is a tool to service-ize semantic objects and to publish them (see Figure 2); the possible outputs of the White Hole (semantic object in OWL or semantic service description in OWL-S) need to be service-ized prior to publishing. So a service (with associated semantic description)
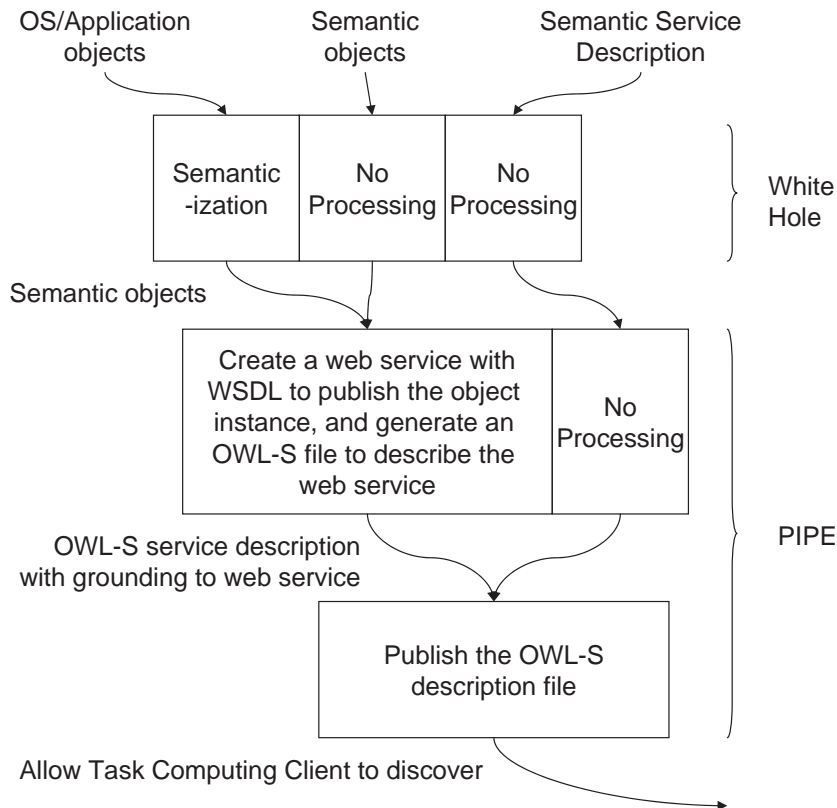
Figure 2. Procedure of semantic-izing, service-izing and publishing of objects and services

	OS/Application Object	Semantic object in OWL or URL to it	SSD in OWL-S or URL to it
Input	OS/Application Object	Semantic object in OWL or URL to it	SSD in OWL-S or URL to it
Function	Identify the object type, create a semantic object	Obtain or download the OWL file	Obtain or download the OWL-S file
Output	Semantic Object in OWL		SSD in OWL-S

Table 2. Function of the White Hole application

is created, which when invoked, will return the semantic object itself. Specifically, PIPE first dynamically creates a web service, which returns the semantic object as its output when invoked; next, a semantic service description for the newly created service is generated (see Table 3). During this process, the name, description, output type, and grounding details of the service are determined and described in a high level (in OWL-S). Our current implementation supports not only the objects we defined, but any OWL object as well.

The outcome of the service-ization is a Semantic Service Description, or SSD, which is either the original one that the user dropped into the White Hole, or the one PIPE cre-

ated to describe the newly created web service. PIPE can be used to publish the SSD depending on the discovery range that the user chooses. For example, if the user wants to publish it as a *group by subnet* service, PIPE will create a UPnP device with a *getDescriptionURL* action that points to the OWL-S file.

Even though we described PIPE in relation to the White Hole, PIPE is a completely independent tool with a web service interface so that it can be called by any other component in a TCE, and used to publish objects or services. One important usage of PIPE is to realize a so-called "bank service", which is a persistent repository of semantic objects. A bank service can be used by users in an environment to

Input	Semantic object in OWL	SSD in OWL-S
Function	Create a web service, which provides the semantic object (in OWL); generate an OWL-S file for the service; make it available through a discovery range	Make the OWL-S file available within a discovery range of choice
Output	Published semantic service	

Table 3. Function of the PIPE application

deposit such things as files, contacts, schedule, etc. as semantic object providing services in the current environment so that people (maybe later) can use those services to accomplish their own tasks.

PIPE also includes a management user interface which helps users to organize the semantic objects or services that the user has published through PIPE. The functions we have implemented include:

- **Switch discovery range** The user can switch the discovery range for the services published through PIPE, for example, in order to temporarily "hold" services (*empty* discovery range).

- **Expiration time** The user can set the expiration time for the services, so that the service becomes undiscoverable after the expiration time.

- **Invocation limit** The user can set a limit for the number of possible invocations, so that the service becomes undiscoverable after that number of invocations.

- **Name/Description** The user can set or change the name and the text description of a service.

3.4. Implemented Services

We have implemented more than fifty semantic services[3]. Each service has a service implementation and a SSD created, except for publicly available services, for which we only created SSD's.

Some services are created from functionalities of the OS and personal applications. For example, "My Video" lets the user pick and expose a video file from her own device and "Add Contact into PIM" lets the user insert a contact into her PIM application. Other services are related to delivering a functionality of a device; for example, "View on Projector" displays on a projector web pages, presentation files, etc. that are provided as input and lets the user control the display (moving to the next slide, scrolling, etc). "View on Projector" is a typical example of how we exposed the functionality of a device as a service. The device (in this case a

projector) is physically linked to a small computer (typically a laptop) that runs a web server (necessary for the invocation of a service using WSDL) and makes available a web service interface to a custom application that implements (in a software layer) the functionality of a device. The computer, which has a network connection (typically wireless) publishes its services within the subnet and handles all incoming requests. However, we expect that in the near future, smarter devices will embed the hardware and software necessary for running a web server and processing web server and service calls. Yet, other services are running remotely on the Internet, such as the "Weather Info of" service that returns a Web page with the weather information for the address given as its input.

None of these services is particularly interesting by itself, but with the help of the TCC, a user can create and execute powerful and useful tasks from combination of basic services, such as the ones mentioned in the introduction.

4. Evaluation and Future Work

We have designed and implemented the capabilities described in Section 3 except for the discovery mechanisms for the *group by interest* discovery range; we have implemented four types of Task Computing Clients, more than fifty services, PIPE, White Hole, and all the modules of the middleware layer of Figure 1. We have created a single installer for all the components so that a user can install a TCC or services in just a couple of minutes. Even though we presented a part of the approaches and solutions, we want to emphasize that all the scenarios described in this paper can be defined and executed in realtime by a user through her TCC. Figure 3 shows a user's desktop screenshot with some TCE components (two TCC's, White Hole and PIPE)[4].

The separation of service implementation and its SSD, and the distinction between discovery mechanisms and discovery ranges have been implemented consistently through out system and work smoothly to deliver the overall user experience and helped us localize system failures while building and debugging the system. In addition, by consistently

---

3　We do not include the object providing services that can be dynamically created through PIPE. Since PIPE can create a semantic object providing services out of any OWL object, it would not be meaningful to include them in the count.

4　Still, a single screenshot conveys little of the dynamism of the implemented Task Computing experience, so we plan to demo our complete system at the conference.

observing these concepts we have implemented a modular system, in which components can be easily modified, replaced and enhanced.

Task Computing seamlessly weaves the rich worlds of personal applications and Internet-based e-services with the devices and services in ubiquitous computing environment. Pieces of the puzzle have been worked on by other researchers and industry. For example, technologies like UPnP, HAVi [1], JINI, etc., address a part of the problem. In general, those technologies are usable by the programmers who know the interface after reading many pages of the standard. Each individual kind of device has a different interface from others even within the same standard and it is difficult to make them interoperate (for programmers and much more so for end-users.) The missing element is the machine-understandable semantic for devices and services with annotations for humans. Semantics plays an essential role in ad-hoc environments, where the spontaneous nature of relationships between devices, services, clients, etc., means that little a priori knowledge is available about the other systems. The Semantic Web technologies (OWL/OWL-S) give a foundation for such semantics so that devices and services can have a common ground of interoperation. In fact, we have tried to use standard technologies wherever available and possible, such as Semantic Web technologies (RDF, OWL, OWL-S), web services (SOAP, WSDL), and ubiquitous computing technologies (UPnP).

Another research project with a similar scope as Task Computing is the Obje Software Architecture [6] at PARC (Palo Alto Research Center Inc.), described as "an interconnection technology that enables digital devices and services to easily interoperate over both wired and wireless networks". Although a detailed comparison with that work would be beyond the scope of this paper, we can summarize as follows: conceptually, the two projects have a different way of abstracting the problem and its corresponding layers. Task Computing being heavily invested in the Semantic Web technologies, and technically, Task Computing relies on semantically described services as the universal abstraction of functionality. Obje relies on mobile code to deliver device and service functionality to devices discovered in the ad hoc environment.

Our work is not anywhere near completion. We believe that at this point we have a robust enough platform, which incorporates the proper abstractions, in order to pursue a variety of different research directions. Increasing the degree of automation (or assistance to the user) during task creation is one such direction and we expect to utilize some planning capabilities in the inference engine of our system. Saving and re-using past tasks is another problem we are pursuing and we follow closely the evolution of OWL-S in that regards. Addressing various aspects of security is another avenue of future work; we plan to incorporate web services

security-related standards and semantics-based policy work [3].

5. Conclusions

We envision Task Computing[5] as a new metaphor for accomplishing tasks in the application-rich, device-rich and service-rich computing environments that permeate daily living. We have designed and built a complete functioning system that utilizes standard technologies in a novel integrated manner, from sources of functionality to end-user. In the process, we strived to create a clear separation of functional layers in order to facilitate a modularized engineering of the overall system and to enable other researchers to utilize components of our system, or to add and enhance the overall functionality delivered to the end user. In this spirit, we plan to release binaries of the modules of our implementation (of the clients, of the middleware layer modules and of the services) to the broader community.

In this paper, we discussed some of our design choices and implementations, from the perspective of the abstract architecture of Task Computing. Specifically, we discussed in detail the dynamic service discovery and creation, management and manipulation of semantically described services in Task Computing.

Acknowledgements

References

[1] HAVi. http://www. havi. org/.

[2] JINI network technology. http://wwws. sun. com/ software/ jini/.

[3] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, 2003.

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

[5] R. Masuoka, B. Parsia, and Y. Labrou. Task computing - the semantic web meets pervasive computing. In *Proceedings of 2nd International Semantic Web Conference (ISWC)*, volume LNCS 2870, pages 866–881, Sanibel Island, FL, USA, October 2003. Springer-Verlag Heidelberg.

[6] The Obje software architecture. http://www. parc. com/ research/ csl/ projects/ obje/ default.html.

---

5   Task Computing relies on collaboration between Fujitsu Laboratories of America, College Park and the MINDSwap group of University of Maryland.
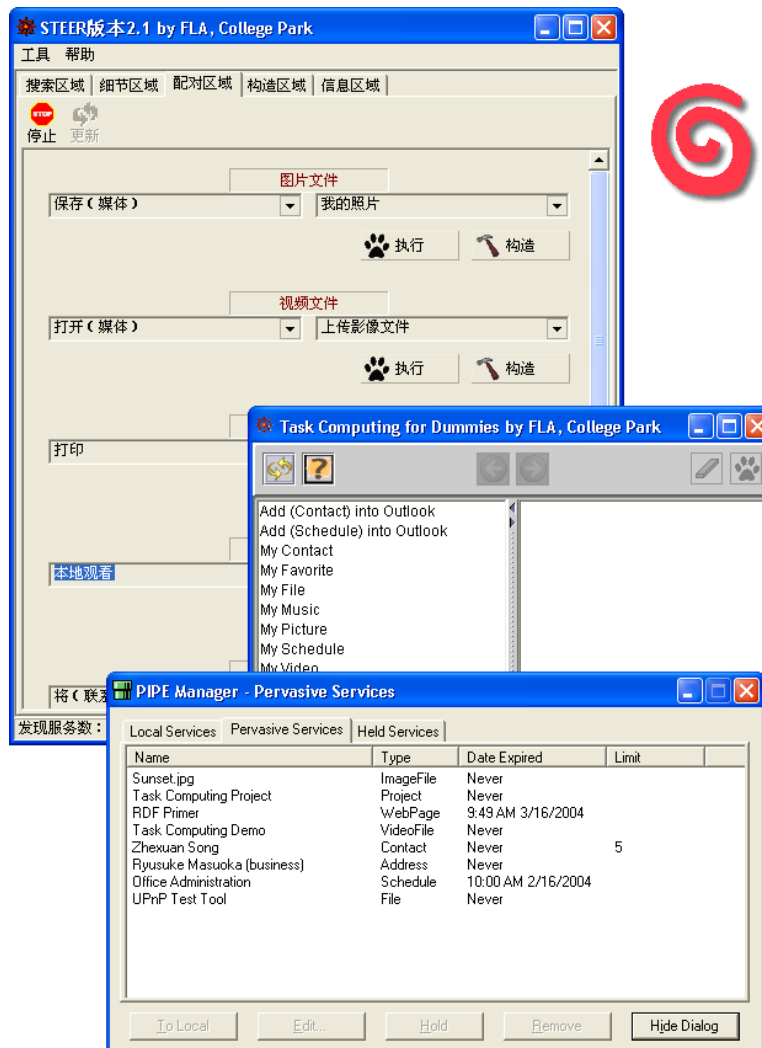
Figure 3. TCE Client Desktop - Two kinds of TCC (one in English and the other in Chinese), White Hole at the upper left corner, and PIPE at the bottom.

[7] Web Ontology Language (owl). http://www. w3. org/ TR/ owl-features/.

[8] OWL-S: Semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.html.

[9] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, April 2003.

[10] Simple Object Access Protocal (soap). http://www. w3. org/ TR/ SOAP/.

[11] Universal Description, Discovery and Integration of web services. http://www.uddi.org.

[12] UPnP forum. http://www.upnp.org.

[13] Web Services Description Language (wsdl). http://www. w3. org/ TR/ wsdl.