On Building Task Computing

Ryusuke Masuoka[1], Yannis Labrou[1], Zhexuan Song[1], Bijan Parsia[2] and James Hendler[2]

[1] Fujitsu Laboratories of America, USA
{ryusuke.masuoka, yannis.labrou, zhexuan.song}@us.fujitsu.com
[2] MIND Lab, University of Maryland, USA
bparsia@isr.umd.edu, hendler@cs.umd.edu

Task Computing is a technology designed to fill the gap between a user's task, that is what the user wants to be done, and the services available as means to that end. Task Computing is an evolutionary step from our previous agent research with a stronger emphasis on end-users and an initial application focus targeting ubiquitous environments.

Task Computing (TC) is built on agent concepts, utilizing Semantic Web and Web Services technologies. TC is distinct in how it combines these technologies in order to serve users' goals. In building Task Computing, we derived much from past experience with using agents in enterprise computing but we were also motivated by the need to deliver solutions that can be deployed in real application environments and used by average users.

In this article, after a brief introduction of Task Computing, we discuss some design decisions based on those considerations.

1. Task Computing

Task Computing is a user-oriented framework that lets end-users accomplish complex tasks in environments rich with applications, devices, and services. Task Computing provides many ways for users to interact with such environments. Our technology is based on Semantic Web technologies, such as OWL (Web Ontology Language) and OWL-S (see page 9) as its core enablers.

Task Computing enables end-users to accomplish a variety of tasks easily, using the functions from disparate sources such as a user's computer or PDA, devices in the environment, and remote e-Services on the Internet. For example:

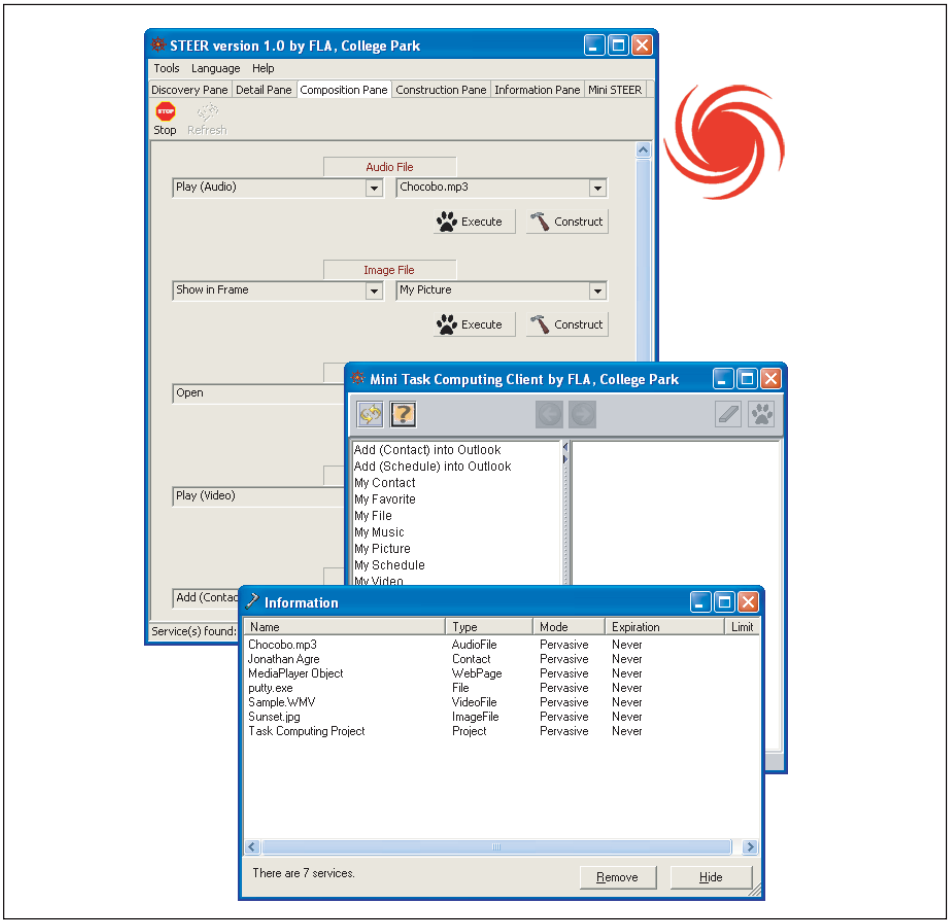

Figure 1 Task Computing Client Desktop: The figure shows two kinds of Task Computing Clients (tab and mini versions), White Hole, a tool to create services from objects, at the upper right corner, and PIPE, a tool to manage the semantic services, at the bottom. For details of those GUI elements, see [3].

- *Display your presentation file from your mobile computer on the projector in the room you visit for the first time without connecting a VGA cable;*
- *Show the photo that you have just taken with your digital camera on the photo frame in the same room and print it on a photo printer without moving memory cards around or driver setups on your mobile computer;*
- *Display the current weather at an address in your PIM (Personal Information Manager) on the projector.*

With Task Computing, end-users can do all of the above and more with just a few point and click operations through simple graphical user interfaces (see Figure 1). The Task Computing client provides a user interface based on human language and the experience is like composing a sentence such as "View on Projector" "My File" and "Show in Frame" "Picture from Digital Camera" and telling the system to execute the sentence. The user can also specify and execute tasks through a voice-based Task Computing client, VoiceSTEER (see Figure 4). The user can compose and execute compositions defined in semantically consistent OWL-S process models, such as "View on Projector" and "Email to" "Ryusuke Masuoka", "Weather of" "Business Address of" "FLA, College Park".

In order to realize such a system in very dynamic environments like ubiquitous computing ones, Task Computing employs the architecture of Figure 2. In this architecture, "semantic service" is adopted as a single abstraction for all the functions (means to assemble tasks). Semantic Service Descriptions (SSD's, which are OWL-S files in our implementation) are used to represent the semantic services in the system. Based on SSD's, the system creates a user-centric task view of the environment and lets the user manipulate tasks at the granularity of the semantic services. One goal of the semantics is to help the user specify what they want to do by eliminating whatever can not be done given the available means at any given time. All the functions provided by Task Computing clients, such as service discovery, composition, execution, and publishing, are made possible by those SSD's. In addition, most Task Computing functions are provided as Web Services so that it is an easy task to create new kinds of Task Computing clients or to integrate

those functions with existing applications (for further details on how TC works, see [1, 2, 3]).

2. Considerations

Task Computing takes into account a number of the issues we encountered in an earlier project where we applied software agent technologies to the virtual database integration of distributed and independently maintained databases in an enterprise [5]. The challenges discussed next arose from that project but we still find them relevant to the deployment of agent technologies-driven applications involving real users in general.

Ontology infrastructure

If agents are to realize their promise, a quality ontology infrastructure is indispensable. Especially when agents need to do logical processing, higher quality ontologies are necessary. The requirements of some mechanisms to support logical processing over multiple ontologies also arise if information originates from multiple data sources with different ontologies. It was a tough job then, and we are afraid it is still difficult to provide such an infrastructure in a large, although having an ontology standard (OWL) helps.

Autonomy

The autonomy of agents can liberate users from many minor details in accomplishing tasks. However it inherently introduces unpredictability and unpredictability is difficult to accept in an enterprise environment. Firstly, there is the problem of accountability for the sytem's actions. Users require transparency of the decision making processes for system actions but the agents' reasoning and motivation behind chosen actions is expressed and carried out in a language that is different than the users' language. This gap makes it difficult for users to understand how a set of actions came about and consequently accept responsibility for them.

Secondly, autonomy implies some form of delegation, which in turn, poses security challenges. A user must provide all the potentially necessary credentials, such as passwords, for an agent to represent him and to do things on his behalf. A user can not tell in advance which credentials will be necessary for a given course of action. Moreover, if an agent asks other agents to further represent the user and to act on his behalf, it is going to be even more difficult to secure the overall system as it gets difficult to retain the control over credentials. In the end, the approach to this problem ought to be consistent with the accepted security practices of the organization that deploys the system.

User interface

The goal of a successful user interface is to bridge the gap between the user's language (spoken language or language of thought), which is the medium through which the user understands the environment she interacts with and the language of the underlying system that carries out the computational actions that result into the user's interaction with the physical environment. As humans, for example, do not talk in ACL (Agent Communication Languages), user interfaces that are easy for humans to understand, need to be provided for the human's intentions and tasks, which in turn ought to be interpreted into the ACL. If it were necessary to program a user interface for each task or each category of tasks, it would be very difficult to provide sufficient flexibility for agent systems.

3. Design Decisions for Task Computing

In this section, the design decisions made for Task Computing are discussed based on the considerations discussed in the previous section. TC uses many of components from agent technologies such as an inference engines and ontologies, but in TC they are combined in an innovative manner.

Ontology in ubiquitous computing

We chose ubiquitous computing as a first application domain for Task Computing. This was a decision driven by the realization that ontologies in ubiquitous computing can be manageable. This domain has certain advantages: end-user functionality is grounded in a physical reality that is intuitively understood by all users, semantic descriptions can be massively reused, and the cost of ontology development is relatively manageable. Those advantages rise from the fact that most functionality in ubiquitous environment stems from devices, which are produced in large numbers and have relatively narrow functionality (for a detailed argument, see [4]).

Less autonomy and more user–centricity - User in the loop

For Task Computing, we decided that each user must always be in the loop and at the center of the world. To help the end-users accomplish complex tasks in ubiquitous environments, we try to balance autonomy with user participation. We use semantics to give users a higher task abstraction of the environment for easier task manipulation by the users themselves. The user always decides when and what to be executed. Ultimate authority is always with the users. We also take the approach of user-driven service composition rather than that of automated planning. The role of the semantics is to steer the user towards permissible compositions and away from invalid ones.

Having the user at the center of all actions means that there is no delegation. This design decision lead us to a hub-and-spoke arrangement[1] of the client and the services (with the client at the hub) for Task Computing see Figure 3. The client is connected directly to each service and there is no delegation between the client and the services. Though this arrangement is sometimes inefficient, this has made things simple and secure in the sense that fewer steps are involved for credentials as the user's client directly invokes all services and the user can provide credentials as necessary.

Human language in the user interface

It is a challenge to provide a user interface for

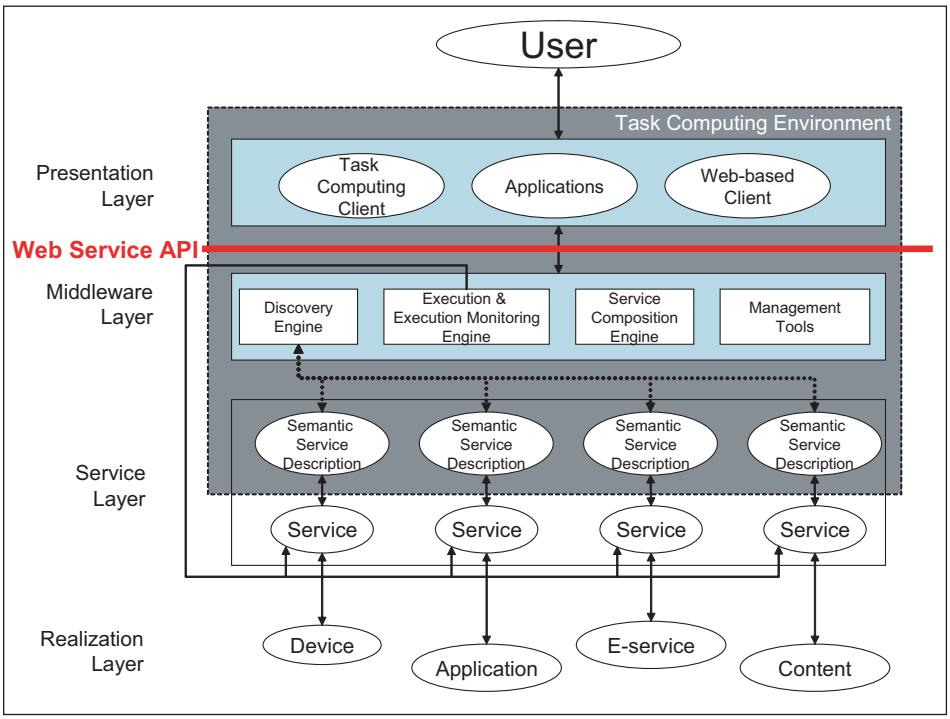

Figure 2 Task Computing Architecture: It has a four-layer architecture with the realization, service, middleware, and presentation layers. For details, see [3].

1 Which is reminiscent of mediator architecture.

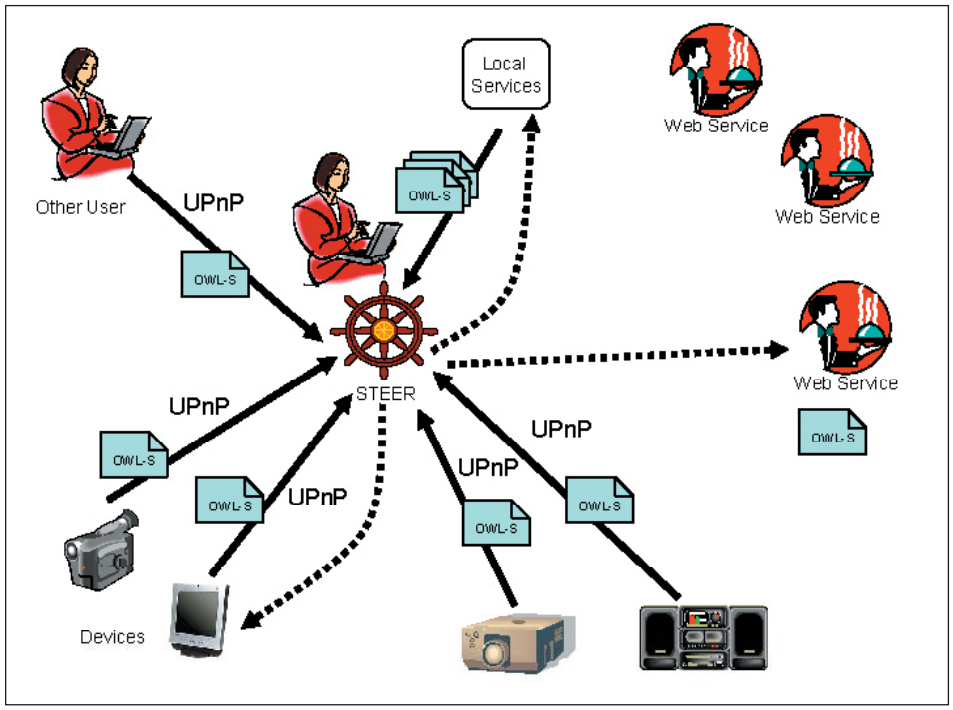

Figure 3: Hub and Spoke: With the semantic service descriptions (SSD's) in OWL-S obtained through discovery (ex. UPnP) and other mechanisms (See [3]), a Task Computing client such as STEER lets the user manipulate the tasks based on those SSD's. When the user asks the STEER to execute a task, each service in the task will be executed independently from other services. (Solid and dotted lines represent service discoveries and executions, respectively.)

the user to interact with the services, some of which they may encounter for the first time in the dynamic ubiquitous environment. The set of available services for the user constantly changes according to a user's movement from one environment to another and the changes in the surrounding people and devices.

Our approach is to base the user interfaces on a "Human Language" metaphor (see Figure 1 for textual UI and Figure 4 for voice UI for TC.). Language makes for a natural and flexible, yet permissive user interface. First of all, the language is the window for the user to understand the world around her. Secondly the user would give tasks in sentences if she is asked what she wants to have done. Therefore it is a natural representation for user's tasks.

We use two very powerful aspects of "Language" for Task Computing supported by the semantics of services: "Composability" and "naming a complex concept." Sentences are composed from many semantically compatible words and phrases. If we give a service the granularity of a word or a phrase and the word or the phrase as its name, the service composition based on their semantic compatibility can map to an understandable sentence. Using semantics, we can constrain the user's manipulation of service composition within her reasonable expectations (such as "View on Projector" "My File" and "Show in Frame" My Picture", but not "Show in Frame" "My Music").

The user can also save any composition, give it a name, and use it as a single service in Task Computing. TC relies on language's "naming a complex concept" aspect, which lets the user manipulate the complex concept (or composition) easily once it is given a name.

4. Conclusion

We have introduced Task Computing (TC) and discussed its design decisions based on the considerations of our previous agent application to enterprise computing. We believe TC and agent technologies are complementary and the two approaches will eventually merge in the future. We see that TC and Semantic Web Services (SWS) technologies will eventually help provide infrastructure for such high quality ontologies and semantically described functions to pave a road to the bright future of agent technologies.

TC software is available for academic institutions. There are many TC clients and interesting TC services (more than fifty, at the time of writing), which allow the users to interact with the ubiquitous environment in many modalities. There are also Web Service API's for Task Computing functions, which make it easy to integrate or to build agent systems on top of TC to try new ideas. If you are interested, please visit the Task Computing site, http://taskcomputing.org.

References

1. Masuoka, R., Parsia, B., and Labrou, Y., "Task Computing - The Semantic Web meets Pervasive Computing -." In D. Fensel et al. (Eds.), "The Semantic Web - ISWC 2003," the Second International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA October 2003 Proceedings, LNCS 2870, pp. 866-881, 2003.
2. Masuoka, R., Labrou, Y., Parsia, B., and Sirin, E., "Ontology-Enabled Pervasive Computing Applications," IEEE Intelligent Systems, vol. 18, no. 5, Sep./Oct. 2003, pp. 68-72.
3. Song, Z., Labrou, Y., and Masuoka, R., "Dynamic Service Discovery and Management in Task Computing," pp. 310 - 318, MobiQuitous 2004, August 22-26, 2004, Boston, USA.
4. Masuoka, R., Labrou, Y., and Song, Z., "Semantic Web and Ubiquitous Computing – Task Computing as an Example –,"AIS SIGSEMIS Bulletin, Vol. 1 No. 3, October 2004, pp. 21 - 24.
5. Tamami Sugasaka, Ryusuke Masuoka, Akira Sato, Hironobu Kitajima, and Fumihiro Maruyama, "SAGE and Its Application to Electronic Commerce - SAGE:Francis: A System Based on ``Virtual Catalog'", Systems and Computers in Japan, vol. 30, no. 6, pp. 36 - 46, June, 1999.

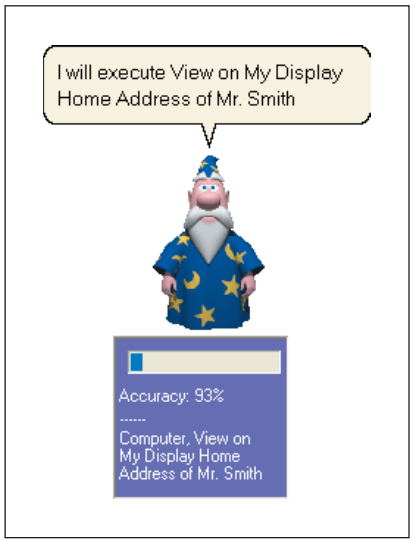

Figure 4: VoiceSTEER: One can speak any semantically compatible service composition of any length (three-service composition, "View on My Display" "Home Address of" "Mr. Smith" in this case) directly to VoiceSTEER for it to execute the task (Microsoft Agent and Speech SDK are used on top of TC Web Service API.)