Self-Optimization of Task Execution in Pervasive Computing Environments

Anand Ranganathan, Roy H. Campbell
University of Illinois at Urbana-Champaign
{ranganat,rhc}@uiuc.edu

Abstract

Pervasive Computing Environments feature massively distributed systems containing a large number of devices, services and applications that help end-users perform various kinds of tasks. However, these systems are very complex to configure and manage. They are highly dynamic and fault-prone. Another challenge is that since these environments are rich in devices and services, they offer different ways of performing the same task; hence, it is sometimes difficult to choose the "best" resources and strategies to use at any point of time. In this paper, we describe a framework that allows the development of autonomic programs for pervasive computing environments in the form of high-level, parameterized tasks. Each task is associated with various parameters, the values of which may be either provided by the end-user or automatically inferred by the framework based on the current state of the environment, context-sensitive policies, and learned user preferences. A novel multi-dimensional utility function that uses both quantifiable and non-quantifiable metrics is used to pick the optimal way of executing the task. This framework allows these environments to be self-configuring, self-repairing and adaptive, and to require minimal user intervention. We have developed and used a prototype task execution framework within our pervasive computing system, Gaia[1].

1. Introduction

Pervasive Computing advocates the enhancement of physical spaces with computing and communication resources. However, along with the benefits of these enriched, interactive environments comes the cost of increased complexity of managing and configuring them. One of the characteristics of these environments is that they contain diverse types of devices, services and applications and hence, they often offer different ways of performing the same task, using different resources or different strategies. This diversity increases the complexity of choosing an appropriate way of performing the task. The best way of performing a task may depend on the current context of the environment, the resources available and user preferences. The developer, administrator or end-user should not be burdened with configuring the task. Hence, we need mechanisms for the system to configure the execution of the task, automatically and optimally.

There are other challenges in the way of configuring pervasive computing environments. These environments are highly dynamic and fault-prone. Besides, different environments can vary widely in their architectures and the resources they provide. Hence, many programs are not portable across environments. Developers, often, have to re-develop their applications and services for new environments. This places a bottleneck on the rapid development and prototyping of new services and applications in these environments.

In this paper, we propose a framework that allows programming the environment using high-level, parameterized tasks. A task is essentially a set of actions or a workflow performed collaboratively by humans and the pervasive system to achieve a goal and it consists of a number of steps called activities. The parameters of the task influence the way the task is performed. These parameters may be devices, services or applications to use while performing the task or may be strategies or algorithms to use. A task may be triggered either automatically by an event or, manually, by an end-user. Our framework loads the task specification and discovers possible values of the task parameters. It then asks the end-user for the best value or figures out the best value, automatically, using a utility function. The framework, then, executes the actions in the task in a reliable manner.

Some of the key features of the framework are the use of ontologies for specifying hierarchies of entities and their properties, the use of learning to obtain end-user preferences, incorporation of security and other policies, and the use of a generic, parameterized task model that allows the same tasks to be run in different environments with different resources. The framework can also recover from failures of one or more actions in a task by using alternate resources. While executing the actions, it monitors the effects of the actions through various feedback mechanisms. In case any of the actions fail, it handles the failure by re-trying the action with a different resource.

[1] This research is supported by a grant from the National Science Foundation, NSF CCR 0086094 ITR and NSF 99-72884 EQ

2. Task Model

In our framework, developers, first, use a high-level programming model[2] to develop basic activities that perform actions like starting, moving or stopping components, changing the state of devices, services or applications or interacting with the end-user in various ways. There are three kinds of activities allowed in our framework: parameter-obtaining, state-gathering and world-altering. Parameter-obtaining activities involve getting values of parameters by either asking the end-user or by automatically deducing the best value. State-gathering activities involve querying other services or databases for the current state of the environment. World-altering activities change the current state of the environment by creating, re-configuring, moving or destroying other entities like applications and services.

Developers can compose a number of basic activities, into a task or a workflow that achieves a certain goal. These tasks may be associated with a number of parameters. For example, even the relatively simple task of displaying a slideshow has a number of parameters like the devices and applications to use for displaying and navigating the slides, the name of the file, etc. The automatic discovery of values of parameters by our framework frees developers and end-users from the burden of choosing myriad parameter values for performing a task, although it does allow end-users to override system choices and manually configure how the task is to be performed.

3. Discovering possible values of parameters

There are various constraints that need to be satisfied while discovering parameter values. These include constraints specified by the developer as part of the task program, constraints specified in ontologies of task parameters and context-sensitive policies specified by an administrator for the pervasive computing environment.

The task execution framework uses a semantic discovery process to discover the most appropriate resources that satisfy the above constraints[2]. A key concept employed in the discovery process is the separation of class and instance discovery. This means that in order to choose a suitable entity, it first discovers possible classes of entities that satisfy class-level constraints. This semantic class discovery is done with the help of ontologies. Then, it discovers instances of these classes that satisfy instance-level constraints. Separating class and instance discovery enables a more flexible and powerful semantic discovery process since even entities of classes that are highly different from the class specified by the developer can be discovered.

4. Optimizing Task Execution

In order to optimize the execution of tasks, our framework picks the “best” values of various task parameters. The metric used to decide the best value may be one of the more conventional distributed systems performance metrics like bandwidth or computational power. In addition, pervasive computing is associated with a number of other metrics like distance from end-user, usability and end-user satisfaction. However, some metrics are difficult to quantify and hence, difficult to maximize.

Our framework uses a novel multi-dimensional utility function where the different dimensions are different kinds of metrics - both quantifiable and non-quantifiable. Quantifiable metrics (like distance and bandwidth) are evaluated by querying services or databases that have the required numerical information. Non-quantifiable metrics (like usability and satisfaction) are evaluated using policies and user preferences. Policies are written in Prolog and specify an ordering of different candidate parameter values. User preferences are learned based on past user behavior.

Entities can have different utilities in different dimensions. A particular entity may be better than others in one dimension, but may be worse in other dimensions. It is often difficult to compare entities across dimensions. Hence, in order to rank all candidate entities for choosing the best one, the developer or administrator picks one of the dimensions as the primary one. This primary dimension is the metric for the task parameter.

5. Conclusion

The task execution framework has been implemented within Gaia[1], our infrastructure for pervasive computing. The framework has been used to implement and execute various kinds of tasks in smart office and conference room settings like displaying slideshows, collaborating with others, messaging other people and migrating applications. We found that the framework helped to prototype new tasks rapidly and port them to different environments.

6. References

[1] M. Roman, et al, "Gaia: A Middleware Infrastructure to Enable Active Spaces," IEEE Pervasive Computing Magazine, pp. 74-83, Oct-Dec 2002.

[2] A. Ranganathan, et al, “Olympus: A High-Level Programming Model for Pervasive Computing Environments,” in IEEE PerCom 2005, Hawaii, 2005