Reducing Configuration Overhead with Goal-oriented Programming

Justin Mazzola Paluska
MIT CSAIL, 32 Vassar Street 32-G788, Cambridge, MA 02139

1. Introduction

Recent years have seen an explosion in the number and diversity of consumer-level electronic devices. Many of these devices work extremely well in one environment in a few scripted ways, but have the possibility of working in many other, unanticipated ways if users are willing to expend considerable effort configuring the devices and connecting them together. Unfortunately, this effort even confounds determined users with time and money to hire professional installers and technical support (Marcus, 2005).

As a motivating example, suppose a user would like to play a video from her laptop computer. The computer normally plays video on its own screen and internal speakers, but the user enjoys the experience more on her television and home theater system. As such, every time the user wants to watch a video from the laptop, she user must:

1. Connect the laptop’s television output—if it has one—to one of television’s inputs,
2. Connect the laptop’s audio outputs to an adapter bought from RadioShack then that to the speakers’ inputs,
3. Activate the inputs of the speakers and television,
4. Start the media player on the computer,
5. Instruct the laptop to render the media to its external outputs, and
6. Press “play” and hope that the television and computer agree on aspect ratios and other miscellany.

If the user would rather move the laptop to another room and play on the laptop’s built-in components, she must reconfigure the laptop once again.

We believe that these configuration hassles can be reduced most of the time to “just play”.

2. The “Just Play” Proposal

Configuration is difficult because devices usually cannot be controlled in a standardized way and, as such, leave it up to users to explicitly mediate between each device. Ideally, users should only need to tell their devices what to do at a very high level and the system should figure out how to accomplish the low-level details.

We propose the following four part architecture to enable users to better use their devices by enabling automatic configuration:

A Common Wireless Interface and Control Mechanism
A commoditized wireless medium reduces hardware connection mismatches to software problems that can be fixed with filters and adaptor functions.

Low-level Descriptions of Device Capabilities
A television is not just a television, but a display device, NTSC tuner, audio output device, and a remote control receiver. These capabilities may be used in a variety of ways beyond simply watching television.

Specification of High-level User Intents
Current systems don’t know what the user wants to do and as a result can only do what the user explicitly tells it to do.

Goal-oriented Configuration System
that can match high-level user intents with recipes for achieving those intents.

In such a system, the user tells the system to “Play Video” and the system responds by searching out for a way to “Play Video” given the resources at hand.

3. Initial Architectural Experiments

We built a prototype “Just Play” system for playing music using a Mac Mini, a laptop, and a speaker modified with an 802.11b wireless interface and MP3-to-analog decoding hardware.

Our prototype uses the O₂S Resources (Pham, 2005) over 802.11b wireless networks to provide the first two parts of the “Just Play” architecture. In particular, the O₂S resources framework provides us with a simple discovery system and a common interface through which to access devices as objects.
to PlayMusic(criteria):
via MP3s:
subgoals:
  source = MP3MusicStream(criteria)
sink = AudioSink()
eval:
  satisfaction = (source.satisfaction + sink.satisfaction) / 2
exec:
  fader.connect(source, sink)

Figure 1. Sample PlayMusic Technique

3.1 Goal-oriented Programming

For the last two architectural requirements, the prototype uses an extended version of the O2S Goal-oriented programming system. Goal-oriented programming (Mazzola Paluska, 2004) provides a way of writing applications so that the algorithms, devices, and resources used can be evaluated and exchanged for ones with similar functionality as needed. There are two primary abstractions. Goals act as a specification layer. We use Goals to capture user and programmer intents. Goals are implemented by Techniques, a mixture of declarative statements and arbitrary code.

Goals are not bound to any particular Technique until runtime and this binding may change as better devices come up or the context in which a particular binding choice was made is no longer valid. The binding of Goals to Techniques is handled by a Planner that cooperates with Techniques to find the best way to satisfy the user’s top level goals. Techniques may be Goal-oriented by declaring subgoals that the Planner recursively tries to satisfy.

3.2 “Just Play” Techniques

Our prototype Techniques use subgoals whenever possible to allow the Planner more choices in implementation. At the highest level is the PlayMusic Goal that the user invokes when she wants to listen to music. Figure 1 shows a sample Technique to PlayMusic via MP3s. A multitude of lower-level Techniques satisfy the subgoals of the PlayMusic via MP3s Technique down to Techniques for the capabilities of each device in our system. As new ways of playing music or new devices come along, we just add new Techniques to the Planner so it has more choices in satisfying the user’s PlayMusic Goal.

A Technique may define eval code that refines the its satisfaction, or suitability to its Goal. For example, living room speakers satisfy the AudioSink Goal much better than do the speakers in a typical laptop, so the living room speaker Technique rates itself higher than that for a laptop speaker. From this evaluation, the Planner can choose the right AudioSink for the job at hand. The Planner also monitors satisfaction at runtime so if a device is no longer usable due to power or other problems, the Planner will notice and switch to new Techniques.

4. Related and Future Work

Configuration is a major burden in Grid Computing and provides a wealth of related work. For example, HP’s SmartFrog (Goldsack et al., 2003) also offers a script-based auto-configuration, but concentrates mainly on starting and stopping services across the Grid, rather than connecting them together. Closer to our work is ISI’s Pegasus (Deelman et al., 2004), which applies AI planning algorithms to generating scientific workflows on the Grid. However, Pegasus is based on moving files and programs around for execution, not connecting devices.

There are two areas in which we need to extend our prototype. First, we need to find a way to secure devices on the device network without configuration hassles. Whereas a laptop and speaker wired together are authenticated to each other by the wire between them, wireless networks offer no such authentication. This problem is compounded if users want to be able to share their device networks or use devices—such as projectors—that are a part of a shared infrastructure owned by no one user.

Second, we need to determine how to best give non-programmers a choice among equally good ways of satisfying a high-level goal. Programmers can write their own Techniques to do so, but we cannot expect most users to do this. A simple interface would be to choose a single plan and switch among them if the user rejects it. Alternatively, another interface lets users explicitly tell the system to use a device in a particular way, perhaps using speech recognition or gesture recognition.

References


