Declarative Control of the Future Home Environment

Tope Omitola
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue Cambridge CB3 0FD UK
too20@cam.ac.uk

ABSTRACT

In this paper, we show how declarative rules, and declarative programming, are used to control a complex, heterogeneous system where rules from disparate sources must interoperate. We show how imperative codes can be automatically generated from declarative rules to bind with the resources in the execution environment. Our approach shows how declarative programming can be used to effectively compose disparate systems together. We apply our approach to the control and co-ordination of home devices.

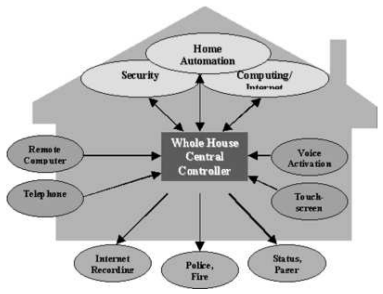

Figure 1: Computational Environment of the Future Home

Keywords
Rules, Rule-Based Programming, Rule-Based Control, Declarative Programming.

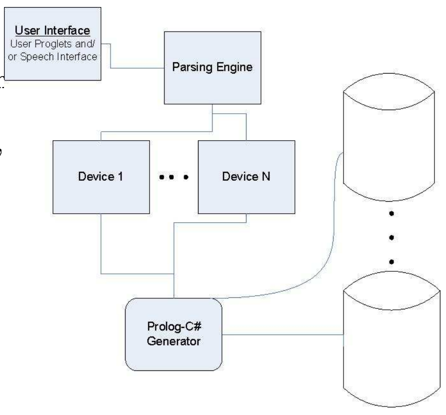

Figure 2: Schematic diagram of our declarative control system

1 INTRODUCTION

The future home environment will be complex and highly dynamic consisting of myriad devices with different properties providing different functionalities, an environment where the autonomous devices interact directly but have to be responsible for their own behaviour. [Figure 1 shows our view of the computational environment of the future home].

A mechanism is needed to co-ordinate and control the services offered by these devices, and harness their services to create a more responsive environment for their users.

There have been various attempts to provide a unifying framework to connect devices. Some solutions have focused on the physical media, such as X.10, others have focused on the network layer, such as Bluetooth, and others have focused on the middleware layer, such as Jini. Some of the aforementioned solutions describe proprietary architectures, therefore not open, and others do not have the registry, transaction, nor the events services needed for device networking. The challenges of verifying services' reliability and of usability are still open problems.

We describe a declarative control system that allows users to write small scripts, called **user proglets**, and converts the functionalities provided by these devices together with these proglets, into Horn clauses, for validating against inconsistencies, and for verifying the interactions. The declarative control system generates imperative C# codes that elicit the desired behaviour from these devices. We use UPnP as the middleware mechanism in our work. UPnP defines a language-neutral non-proprietary architecture using XML and the TCP/IP protocols. The devices expose the functionalities over the UPnP network using XML descriptions.

Our aim is to build an environment that the user can control the behaviour of by specifying policies, i. e. rules, and a mechanism that verifies the interaction of these rules with the devices' behaviour.

2 ELEMENTS OF DECLARATIVE CONTROL

Our declarative control system consists of:

- A user-interface mechanism allowing users to describe device policy behaviour(s)
- The devices to be controlled
- Parsing Engine
- Rule Base Controller (RBC), including Rule Base Engine(s) and the Devices' Registrar
- Prolog-C# Generator

A schematic diagram is shown in Figure 2.

The present user-interface mechanism is a scripting interface that allows users to create **user proglets**, such as " ...switch on the living room light", " ...turn dining room light on for one hour". [1] Our Parsing Engine parses the user proglets and the devices' XML descriptions, turning them into Prolog (Horn) clauses, and storing them in the Rule Base(s), where validation, consistency, and verification are performed. We are presently experimenting with federating our rule bases to provide adequate scalability. The Prolog-C# Generator generates imperative C# codes from the Prolog clauses passed to it by the RBC.

3 EXAMPLE APPLICATION

Our current implementation controls and coordinates light switches and light devices. A user-created proglet specifies how to control the switches and light devices, is converted into (Voice)XML and then to Prolog rules. These rules are fired at designated times and sent to the Prolog-C# generator, after validation and verification, by the RBC. The Generator generates the C# codes that control the devices.

4 EVALUATION

We have tested our declarative system using virtual devices, and are currently building embedded device hardware that will be controlled by our system.

5 CONCLUSION

In this paper, we have shown how declarative rules and rule-based programming are useful to control disparate systems of disparate functionalities, and how to mix declarative and imperative programs to control devices.

[1]We have done some work using speech to control a UPnP-enabled network security camera (using a speech recognition system from Linguamatics (www.linguamatics.com)), and we are currently evaluating how to incorporate this speech recognition system into our declarative control system.