Centralized Task Scheduling in Home Network

Srividya Doss K R, Pavithra SeshadriVijayakrishnan
University of Wisconsin- Madison

Abstract

We envision the digital home of future as a diverse mix of inter-connected devices, infrastructure and services that seamlessly collaborate through wired or wireless networks to share content and enrich the home user experience through advanced services. The present explosion of heterogeneous multi-vendor devices demands a centralized way for managing and scheduling tasks on all connected devices and for controlling the interaction between them. Motivated by the lack of effective approaches today in programming and managing these technologies, Microsoft has researched and developed a platform called HomeOS, which provides a simple service abstraction for programming and composing their functionality. We use this platform to develop a centralized scheduler which can run tasks (like sending reminder or running a program) seamlessly on any device connected to the home network by specifying the trigger. We have designed the application such that it is intuitive for non-technical users and can adapt in the face of fast-changing environment. We also maintain access control lists to provide security for the tasks performed on certain devices at a certain time and limit access to some users like guests. We set up a test bed with a variety of devices found in today’s homes using which various task scheduling scenarios with static calendar data and dynamic occurrence of real-time events as triggers were implemented using .NET framework in C# language. We show that our task scheduler application is extensible and scales and performs well in extreme scenarios.

1 INTRODUCTION

Modern homes are on the verge of becoming one of the richest everyday computing environments with a wide range of devices like computers, TVs, cameras, mobile devices, tablets and sensors. With the realm of technophiles expanding, the diversity of devices at homes increases, making their integration tedious and hence task management spanning multiple devices difficult. We need a robust ecosystem where all devices connected to the home network work together seamlessly without worrying about vendor specific issues. Software that would handle device heterogeneity and provide smooth and effortless execution of day to day tasks will be crucial to meet the goals of digital home of the future.

To address the limitations in the present home networks, Microsoft has developed HomeOS [1]- an experimental operating system which focuses on providing centralized control of connected devices in the home with useful programming abstractions for developers. We take this effort a step further by developing a centralized task scheduler application on top of HomeOS which enables a user to automatically perform routine tasks spanning multiple devices. The Task Scheduler does this by monitoring whatever criteria you choose to initiate the tasks (referred to as triggers) and then executing the tasks when the criteria is met. Static calendar data and dynamic occurrence of real-time dependent events act as triggers. With our task scheduler application, we enhance the user’s ability to perform cross-device tasks by giving a single point of control to schedule common tasks like sending a notification or running a program on any device in the home network.

The HomeOS software is structured like a plug-in framework with two core pieces – the host platform and the plug-in modules (a driver or application)[2]. Its programming model is service-oriented: modules provide functionalities by exporting a set of services. We have implemented various task scheduling scenarios by utilizing the services of device drivers available (for DLNA media renderers and servers[3], webcams, face recognition, sending notifications).
Following would be examples for static event scheduling: (i) notification about a meeting to the specified devices based on calendar date and time (ii) Media service, in which particular media content is played on the DLNA media renderer at a specified time which can also be used as an alarm. We have also implemented dynamic event scheduling like Alert service, wherein using image recognition we will send a notification to a member of the house on the arrival of someone at the front door by polling the web-camera set at the entrance during a time interval specified. We can also active the media service on this dynamic event. For efficient scheduling of tasks, we have provided an interface for enumeration of all tasks set for a particular user group. We have restricted access to some devices for certain members and during certain time of the day (e.g., children cannot schedule tasks for TV in parent’s room and on their TV after 9pm). We also provide only restricted access to guest for scheduling tasks (e.g., only on TV in the guest room & their own smart phone). Providing access control based on the content is beyond the scope of this project.

We have made the design based on the mental models of the user and hence can be used by non-expert users intuitively. It can be easily extended to encompass other new devices added to the network and to support additional task scheduling scenarios that they will enable. For example, we can setup an in-home elder-care service by using multiple sensors and devices to monitor the BP/glucose level and automatically trigger a signal or a notification to the caregiver when the reading is abnormal. Thus our design provides a centralized and an extensible way to manage and schedule multiple tasks spanning all devices in the network.

We first discuss the HomeOS architecture in Section 2. Section 3 describes the goals of our application design and gives an overview about the design. In Section 4 we elaborate the design and give a detailed view of our implementation. Section 5 talks about our experience with HomeOS platform. We present evaluation results for our prototype and future work in Section 6. Relevant works related to our application are discussed in Section 7. In Section 8 we summarize our conclusions.

2 HOMEOS ARCHITECTURE

HomeOS [2] is a home-wide OS written in C# on .NET 4.0 framework that runs on a centralized server. Devices at home that vary in their vendor, utility and capacity are all connected to this via the home network. The aim of HomeOS is to provide seamless interface between the devices and acts as a communication system between the smart devices. Just like how an operating system is a platform that allows the user to install various applications, HomeOS allows the inclusion of applications which serve the purpose of home device integration. HomeOS provides abstractions that enables easy development of applications and the developer is not required to know low level details of the devices.

HomeOS also provides basic security and access control features and thus managing the applications become effortless. HomeOS has datalog based access control rules. It provides better access control than traditional OSes with a low usage barrier. Both users and devices are divided into groups. User groups are arranged in three level hierarchy as shown in figure 1, similar to the concept of family tree. Access to devices are based upon these groups. For instance 'Television' in parent's room cannot be accessed by children. While the control plane is centralized and runs on a dedicated, always-on appliance (e.g., a home server), the data plane is not centralized.

![Figure 1: User Group Hierarchy](image)

HomeOS is a modularized software (figure 2) that provides enough isolation between individual modules and the developers need not worry about interaction with other modules. The modules are segregated into two types - drivers and applications, all written by third party developers. Driver modules are each specific to a device and enable seamless access to a device without
interoperability issues. Only these drivers have low level access to the devices encapsulating the details from application developers. Applications access the devices through these drivers. HomeOS thus provides service abstraction via drivers.

Figure 2: High level overview of HomeOS

HomeOS detects devices based on protocol-specific device discovery protocols. DLNA [3] and Z-Wave [4] protocols are used at present for device detection. HomeOS guarantees (1) simplified application development (2) ease of management (3) Enables future innovation and device differentiation.

Any functionality that the HomeOS provides is exposed as a service that accesses the HomeOS through a port. HomeOS services have unique names/identifiers and the set of modules are also given unique identifiers. A port is a communication abstraction, a functionality described in terms of roles and controls. Roles are text strings giving general functionality and controls are operations that can be performed with a service. Module that needs to access a service requests for capability. This capability can be used to access the service and also others services that can be accessed via this service. Applications can invoke the functionality offered by a role immediately or subscribe to it and they are invoked by the role on the occurrence of a particular event.

The centralized control over the devices at home, simple service abstraction, ease of management and more importantly the functionally independent modules of drivers and applications makes HomeOS attractive and an architecture easy to program home technology.

3 TASK SCHEDULER GOALS AND DESIGN

We now discuss the goals for the design of our task scheduler application and give an overview of how we achieve them by our design. The next section details the technology choices and the implementation.

3.1 Design Goals

3.1.1 Easy to schedule
The task scheduler application should make it easy for non-expert users to schedule and manage tasks for the devices in their homes. Many common scheduling tasks like adding time-based or dynamic event-based actions spanning across devices must be simple. This simplicity can be achieved not only by good user interfaces; research shows that building intuitive interfaces requires the system-level primitives themselves to match users’ mental models.

3.1.2 User Privacy
In the home environment, members of the house (adults, kids, guests) might want to keep their tasks private or want to have customized access control to tasks for a subset of the members. So our task scheduler should be able to group tasks based on different access control lists and make them visible only to appropriate users. We also need to limit access for certain members to device at various locations in the house and during specific time intervals.

3.1.3 Compliance with HomeOS architecture
Our design must fit into the existing infrastructure of HomeOS to re-use functionalities and take advantage of various features offered by HomeOS like user preferences for security and access control policies, co-ordination of device access etc.

3.1.4 Enable innovation and Future Extensibility
Task scheduler should support devices that are yet to emerge. Device vendors must be able to expose their new or differentiating features, and our application must be able to make use of them to enable additional scenarios for scheduling. So our architecture must be extensible for future innovation.
3.2 Design

Keeping the above goals in mind, we present the design of our task scheduler below. Figure 3 provides a high-level overview of our task scheduler application. It provides a centralized way to schedule tasks composing the functionalities of various devices in the home.

There are two major components of the Task scheduler back-end - Triggers & Actions. The upper part of the figure shows the various interactions for setting the trigger - both calendar-based and dynamic-event based. The virtual timer driver monitors if the set time is reached and notifies our application. The event drivers detect and notify our task scheduler application of the dynamic events monitored. The lower portion shows the interactions for taking the action for the scheduled task. We use the drivers of devices (can be virtual like notification) to invoke operations for running tasks on them. We have a store in our back-end where we serialize all the tasks for various access groups. A GUI interacts with the user and communicates with the Task scheduler back-end to store and schedule the tasks.

3.2.1 Back-end

When the application starts, it first initializes and gets the capability for all the ports registered that export services to our task scheduler application. Next, we parse the tasks stored in our data-store and monitor for the triggers using Timer Driver(monitors the DateTime) and the dynamic event Drivers (monitors for a specific event). This is done by subscribing to the notify operations and setting the value to monitor. This is done asynchronously and we handle the notification of the monitored parameters in our application.

We use the AsynReturn callback to take the required action depending on the monitored values returned from the Driver modules. We query our datastore for tasks with matching triggers and run the actions associated with them. This way we can have complex combinations of triggers and actions for each task scheduled.

3.2.2 Front-end

The GUI at the front-end interacts with the users and they initially login to our application with their password. We access the datastore of HomeOS which maintains the list of users and their details to verify this. This way we can enable access control and display tasks only for those groups that the user should have visibility to. In the next screen, we enumerate the list of all tasks which the user can view and edit. From this screen, the user can also choose to create a
new task which opens a window to collect all the details required for the task. The user will have an option to choose either calendar-based or event-based which will enable options for the appropriate parameters for the triggers. Once the trigger selection is done, the option for selecting one or more actions and their parameters are enabled.

When the user confirms the completion of entering the task details, two actions are taken - the task details are serialized to the data store corresponding to the group selected & the triggers are set to be monitored like in the application start-up phase. We have a simple self-guiding UI that takes the user through the steps involved in viewing and scheduling tasks. We make our UI intuitive as shown in the next section and consistent with common applications.

4 DETAILED DESIGN AND IMPLEMENTATION

4.1 Language and Technology choice
We implemented AppTaskScheduler, the task scheduler application in C# on top of .NET framework. XML has been used to encode the documents and XAML as the user interface markup language. The language set is same as the one used for any other application in the HomeOS. We chose it for easy integration with the HomeOS and for the re-usability of the existing drivers and applications in the task scheduler application.

The HomeOS uses XML (Extensible markup language) to encode details about the devices connected to home network, locations of the devices, modules present in the HomeOS, Roles in the HomeOS, Access rules for various devices and list of users. The user defined tags of the language allows for the simple specification of the hierarchy of the HomeOS users. To comply with the specifications and also for the simplicity of design we chose to write the task details in XML format. Access tags have been used to specify the tasks belonging to a particular access group. AppTaskScheduler allows for finite set of triggers and finite set of actions based on the triggers. Trigger can be static i.e., based on calendar input or dynamic i.e., based on an event. Separate tags have been used to specify the trigger and action details. Figure 4 shows a simple ‘Tasks’ XML file. We use LINQ (Language Integrated Query) to query from XML file and write to it.

4.2 User Privacy (Access Control)
The HomeOS user is requested to log in with the name and password to access and schedule tasks. Tasks can be added to one's own list or for a group based upon the user tree hierarchy. If the login is valid the user is presented with the set of scheduled tasks applicable to the groups that the user is a part of and is provided with options to edit/add tasks. For instance, Jeff is allowed to schedule a task for all residents/ adults/ kids/ Jeff. Rob can schedule a task only for himself.

4.3 Task Creation
Figure 5 shows the GUI for adding new task. The Name and a description for task along with user/group for which the task is scheduled is specified. Once the user/group is chosen the user must provide the details for trigger and action set for the task.
### 4.4 Triggers

#### 4.4.1 Static Triggers

Static trigger is based on calendar input. The user is required to enter the date and time for this type of trigger. The task is scheduled to run on the specified date and time by the AppTaskScheduler. User can also specify the repetition interval as ‘daily’, ‘weekly’, ‘monthly’ or ‘none’ based upon the requirement. Apart from storing the trigger detail in XML file, the scheduled time for the task is stored in the driver ‘Timer’. The driver 'Timer' stores a list of DateTime for all the tasks that have been scheduled using the task scheduler in a chronological order. AppTaskScheduler has subscribed to this driver, so when the date and time in the driver equals the current time of the HomeOS server it notifies the application. The application reads the task details which match the time and triggers the action set for the task.

#### 4.4.2 Dynamic Triggers

Dynamic triggers propel the action set based on the occurrence of an event within a specified time interval. The user is required to choose the time interval ie., date and time for activation and expiration and also the dynamic event. We have implemented dynamic image recognition event. The person name, whose image has to be recognised is obtained from the user. The ‘DriverPersonMonitor’ utilizes the ‘DriverWebCam’ to capture an image using the web camera at regular intervals and use ‘DriverImgRec’ the image recognition driver to identify the input image. We trained the DriverImgRec with a few images of the required user set. ‘DriverPersonMonitor’ notifies the task scheduler application when it recognizes one of the person in its monitor list. If the notified time is between the activation and expiration time, the action set is triggered.

### 4.5 Actions

We have implemented three types of actions. (i) Notification by email (ii) Notification by SMS (iii) Play media. Users are allowed to trigger one or many combinations of these actions. For (i) and (ii) ‘DriverNotification’ is used. The email id and phone number for the members of the house and are included in the XML file which has user details and we get only the message to be notified from the user. For (iii) ‘DriverDlnaDmr’ and ‘DriverDlnaDms’ are used for

![Figure 5: GUI for adding tasks](image-url)
invoking operations on the DLNA renderer and server. We have configured the Windows 7 laptops in the home network as media renderers and servers and any available media can be played on a chosen device as the result of a static or dynamic trigger.

5 HOMEOS APP DEVELOPMENT EXPERIENCE

5.1 Testbed
Our prototype runs on a testbed consisting of a Windows 7 laptop (running HomeOS), a desktop, a smart-phone and a web-camera. We implemented two virtual drivers, the ‘DriverTimer’ to keep track of scheduled date and time for tasks and ‘DriverPersonMonitor’ for the dynamic image recognition event. The Windows 7 laptop and the desktop which have Windows Media player running were used as media servers and renderers. The main logic for our application is in ‘AppTaskScheduler’. It includes the back end code that interacts with the drivers, the XML data store and also the user interface.

5.2 Experience
Our application spans across almost all devices/drivers currently supported by HomeOS. Hence we had to do a thorough study of the existing drivers/applications and HomeOS paradigm to come up with our design. So the initial phase of our application design and basic infrastructure implementation took substantial amount of time in the course of our project. Thereafter enabling more scenarios by adding additional triggers or actions consumed less time relatively. We found the HomeOS architecture to be well thought-out and extensible. Functional independence of the modules allowed for the re-usability of the existing drivers. The abstractions provided by HomeOS were very useful and simplified our coding process. The existing drivers and applications were a good starting point for our implementation.

The operations exported by the device drivers are used by all the applications therefore any changes in the existing driver operations (like updating it for including additional parameters) could break the existing dependent applications. So, appropriate guidelines for the device driver coding is necessary so that it is both extensible and compatible with existing applications.

HomeOS Development support in IDE: Our application involves the functionality of a lot of devices hence we had to work with many existing drivers and also write our own modules. Customized development support in Visual Studio for HomeOS modules viz. creating a new project which automatically adds the required references and generates the basic standard code for the drivers and application would enhance the efficiency of HomeOS application developers. Provision of a toolbox support for commonly used driver components reduces redundancy in coding and provides a better interface for developers. Developers can just drag and drop it as a component onto a project and thus generate the basic code automatically.

6 EVALUATION AND FUTURE WORK
We performed an evaluation of our task scheduling application by focusing on its ease of scheduling tasks, extensibility, and performance. In this section we have a brief discussion on these aspects and identify the scope for future work with our current working prototype.

6.1 Ease of use
The GUI abstracts the complex interactions at the back-end and provides user with an intuitive interface to create and manage tasks. To evaluate this, we performed a study by presenting our interface to a set of users without much prior technical knowledge in computers but just use it on a day-to-day basis. Once given the goals of the application, we found that all of them were able to easily create even relatively complex tasks with our intuitive interface.

6.2 Extensibility
In our task scheduler design, we see that we can extend our application by adding additional triggers and actions which will enable new scenarios. Our design is not tied to the service’s functionality and any kind of service can be registered and adapted into our architecture. This flexibility enables future innovation as it permits devices and applications that are yet to emerge. Providing extensibility for the residents who
wish to add additional devices or include more user is taken care by the HomeOS layer which our application gets automatically.

6.3 Performance
6.3.1 Robustness
Failure in running one task doesn’t affect the other tasks adversely or crash our application. Providing continuous execution even on device failure by running on alternative devices automatically depending on network topology is not straight-forward in home context as there are other constraints involved. Devising a system for this is a future work item. Like any other centralized system, this has a single point of failure. But since we have a store for all the tasks and the logs, we will be able to retrieve all the information for alternative execution till the system is running again. The data store can be backed up in regular intervals for additional reliability.

6.3.2 Latency
Since our application involves intensive back-end monitoring for DateTime or the event parameters, we had to take care that it is responsive to the user with minimal latency. As we have clearly separated the GUI processing from the back-end modules, our application provides a good scheduling experience to the user and the turnaround time is the same as those achieved by the existing applications on HomeOS which we had as benchmark.

We have an interrupt based mechanism for handling the asynchronous events of the static and dynamic trigger notification. This is useful in separating out the task trigger monitoring part from the task action execution component of our task scheduler and it enables running multiple tasks while simultaneously monitoring for the triggers. So the latency for task execution is very less and is achieved at the exact minute it was set. This works well even if a large number of tasks are scheduled at a particular time as the data plane is not centralized and is achieved by delegating the actions to appropriate drivers.

6.3.3 Scalability
We found our task scheduler architecture to be scalable for home scheduling scenarios where we need to accommodate additional devices and their functionalities, additional users, locations which can change over time. We use XML files in the back-end to store information related to tasks similar to that used for Windows System Task scheduler and use it for querying tasks matching a criteria. This can scale well even if the number of tasks increases. We have few integration points in our application using which additional device functionalities can be seamlessly integrated in the context of scheduling.

6.4 Troubleshooting
Our application maintains a log of all the completed actions in a file. This records any failure which happens at the devices or in the modules. The log can be analyzed to give a fair idea of the possible reasons for failure of an action. The basic infrastructure for troubleshooting common issues with home network that are application independent will be provided at the HomeOS layer. For our application specific issues, implementing a full-fledged module for analysing the logs and arriving at the possible errors and suitable fixes using a decision-tree based approach[8] will be a future work item.

7 RELATED WORK
While we draw on many strands of existing work, we are unaware of any application similar to the one we have designed and implemented that 1) centrally schedule tasks spanning all devices at home; 2) simplifies user experience for managing and securing even complex tasks; 3) provides scope for extension with more devices and scenarios to accommodate the rapid changes in home computing.

There are few sample applications that exist (as given below) that use the driver of each device to bring out the functionality.

- AppDlna: for playing any media on any renderer
- AppCamera: displays camera output by scribescrbing to image feeds from cameras in the home
- AppImgRec: a sample application that uses driver for Image recognition
- AppDoorNotifier: sends notifications when door/window sensors are triggered.
There are some vendor-supplied applications are often limiting because they vertically integrate their own devices with software. For instance, electronic locks come with their own software[6]. Such software offers the full set of device features but does not enable any form of device composition that the vendor did not foresee (e.g., tying it to the front door camera). There are some existing task scheduling applications which is limited in its context and application (e.g., windows task scheduler for a chosen computer, Googlecalendar which only sends email notifications and cannot be based on a real-time event trigger. Many systems and standards for providing device-to-device interoperability have been previously proposed including DLNA[3], Z-Wave[4] and numerous applications built using these technologies to provide customized solutions but our application can be agnostic to what interoperability standards are used and provides abstraction by incorporating any of these devices.

8 CONCLUSION

A centralized task scheduler that gives a single point of control to schedule tasks composing the functionalities of the devices in a home network was developed over the HomeOS architecture. The application lets the user schedule and run tasks on any device based on specific set of triggers. We implemented static trigger, which is calendar-based and dynamic trigger, which is event-based. The task scheduler application was evaluated for its performance and achieving its design goals. The prototyping experience using HomeOS platform was documented. HomeOS proved to be a good architecture providing the developers various levels of abstraction enabling ease of development. Our application has privacy and security support and handles multiple simultaneous schedules without errors and with minimal latency. Thus a robust and an intuitive design was developed over the HomeOS platform.

9 REFERENCES