9

# Memory management in smart home gateway

Beizhong Chen
*Rutgers, the State University of New Jersey*
*USA*

Ibrahim Kamel
*University of Sharjah*
*UAE*

Ivan Marsic
*Rutgers, the State University of New Jersey*
*USA*

## 1. Introduction

With the trend of making house-hold appliances network-enabled and the widespread use of the broadband connections to homes, Internet services are increasingly finding their way to home applications and gradually changing the traditional lifestyles. As opposed to connecting traditional computing devices such as PCs, laptops, and PDAs, the Internet is increasingly used for home appliances such as television sets, refrigerators, air conditioners and washers (Ishihara et al. 2006; Ryu 2006; King et al. 2006), and other remote services, such as remote patient monitoring. Remote medical diagnosis, and remote configuration and control of home appliances are some of the most attractive applications. In the entertainment field, there are several interesting applications, such as downloading movies on demand and an Electronic Programming Guide (EPG). Electric power companies are also keeping an eye on home networking because it will allow them to provide value-added services, such as energy management, telemetry (remote measurement), and better power balance that reduces the likelihood of blackouts. Consumer electronics companies have started to design Internet-enabled products. Merloni Elettrodomestici, an Italy-based company announced their Internet washer Margherita2000 (Jansen et al. 2006), that can be connected to the Internet through which it can be configured, operated, or even diagnosed for malfunction (Ishihara 2006). LG presented the GR-D267DTU Internet refrigerator, which contains a server that controls the communication to three other appliances and has full Internet capabilities. Matsushita Electric showed during a recent Consumer Electronic exhibition an Internet-enabled microwave oven, which can download cooking recipes and heating instructions from the Internet. All of these effort lead towards the so-called "Internet of Things," which is forecast to grow vastly larger than the current Internet.

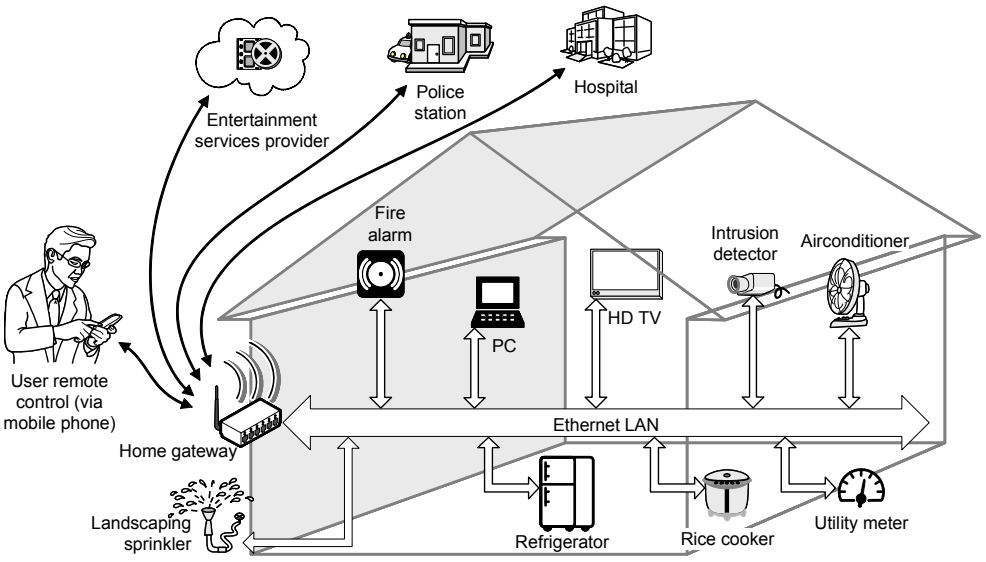



Fig. 1. A typical smart home network environment. See text for details.

Figure 1 illustrates a typical smart home network environment. Generally, there are several types of services in a digital smart home:

- **Basic services** (not shown in Figure 1):
  - Internet Access, Firewall, Personalized UI, Network Security, etc.

- **Information services and health care:**
  - Personalized e-Commerce services will automatically generate an order to a grocery shop when items in the refrigerator fall below a given threshold. This service can also support a more sophisticated shopping model that can bring improvements for the user, such as selecting a retail shop with the cheapest prices.
  - Remote Patient Monitoring will help reduce the cost of medical care. It mainly focuses on chronic diseases and monitors patient's physiological variables such as blood pressure, heart rate, blood sugar level, which will be recorded and transferred to a hospital for a timely professional consultation.
  - More information service models can be developed by service providers.

- **Communication:**
  - Unified Messaging services will allow users to have all the services that are expected in modern organizations, such as telephone call transfer, individual mail boxes, video sharing, etc. Voice over Internet Protocol (VoIP) can be part of this service, helping reduce the user's monthly telephone charges.
  - Other potential services include calendar, to-do list, e-mail, etc.

- **Entertainment:**
  - The Audio-on-Demand, Video-on-Demand, and Interactive TV bring to the home the newest popular music and movies. The users can select their favorites from a directory without leaving their couch. They can start, stop, or pause at will, just as they do for the discs or tapes at home.
  - Other potential services include updating electronic toys remotely, video games, virtual worlds, etc.

- **Control:**
  - Device Remote Diagnostics Service can test the appliances and system to ensure they are working properly. When a functional problem is detected, an alert can be sent. In the case when the problem cannot be solved remotely, the information will be sent to the service company and the service team can contact the user for proper parts. Device Integration Services allow the devices to communicate together and synchronize their work.
  - Energy Management can turn on the landscaping sprinkler when the power price is suitable, based on the weather. The user can use their mobile phone to send control signals to the air-conditioner at home to have it turned on before they arrive home.
  - More control services can be developed by appliances manufactures and service providers.

- **Home security:**
  - The home security systems, such as fire alarm system or intrusion detector, can be connected to a police station or security company to send alarm when unusual situation is detected for a quick response.

In this rapidly developing smart-home area, numerous companies compete to develop new technologies and products. Currently, there are several initiatives to define the specifications for network protocols and APIs suitable for home applications, such as Zigbee Alliance (2004), Sommers (2006), Jini by Sun Microsystems (2007), UPnP by Microsoft Corporation (2008), to name a few. It is expected that multiple home network protocols will coexist in the home and interoperate through the home gateway (Watanabe et al. 2007). Using a home gateway, previously isolated home devices such as digital camera, air conditioner, monitoring system, etc., can be integrated into a smart home system. The gateway also acts as a single point of connection between the home and outside world. Open Service Gateway initiative (OSGi) (The OSGi Service Platform Release 4 Core Specification version 4.2 2009; Binstock 2006) is a consortium of companies that are working to define common specifications for the home gateway. According to the OSGi model, the gateway can host services to control and operate home appliances. In the OSGi model, services are implemented in *software bundles* (or modules) that can be downloaded from the Internet and executed in the gateway (Maples & Kriends 2001). For example, HTTP service is implemented in one bundle, while security application could be implemented in another bundle. Bundles communicate and collaborate with each other through OSGi middleware and, therefore, some bundles may depend on other bundles. For example, a home security bundle uses an HTTP bundle to provide external connectivity (Lee et al. 2003).

To be deployed in customers' home, the price of the gateway is a main concern. Consumers might not be willing to pay for an extra box (home gateway). Also adding gateway functionality to an existing appliance, e.g., TV or set-top box (STB) would increase the appliance prices or shrink an already slim profit margin in this market. There is no consensus among consumer electronic industry on whether the gateway will be a separate box or it will be integrated in home appliances like digital TV or STB, or whether the gateway functionality will be centralized in one device or distributed among several appliances. However, the gateway will be, in general, limited in computational resources, especially main memory and CPU. The main memory in a home gateway will be used by various service bundles and home applications.

This chapter discusses the memory management in gateways and prioritizing the memory use to maximize the number of services running simultaneously in the home gateway. We propose new algorithms for efficient management of service bundles. Memory management has been studied extensively in the traditional operating systems field (Silberschatz & Peterson 1989). Due to different architecture, memory management for software bundles executed in home gateways differs from traditional memory management techniques in the following aspects:

- Traditional memory management techniques generally assume that memory regions used by different applications are independent of each other while some bundles may depend on other bundles in a gateway, as explained in Section 2.
- Due to the cost reason, many of the commercial gateways do not come with storage disks, which make the cost of stopping applications or services relatively high because restarting a service might require downloading the service bundle from the service provider through the Internet.
- Some home applications are real-time; therefore, removing a bundle from the memory may result in aborting the application or the service, while in traditional memory management model, removing a page from the memory costs only one disk I/O operation.

In a home gateway, generally, terminating a service might result in aborting one or more applications. However, in some applications it is possible to remove one service in the application and keep the application running. For example, audio-on-demand might still work without the equalizer service. However, if the application considers the terminated service critical to its operation, it might terminate all other services in the service tree as well. In this chapter, although the proposed model and algorithms work for the two cases mentioned above, we assume that terminating a node or a sub-tree from the service tree would terminate the whole application.

The main contributions of this work are:

- Identifying the difference between memory management in home gateway and traditional memory management problem in a general computing environment (addressed in operating systems literature).
- Introducing a model for the service replacement in home gateways using a directed dependency graph.

- Introducing SD (service dependency) Optimal algorithm which can work as a benchmark to compare different algorithms.
- Introducing SD Heuristic algorithm that performs significantly better than traditional memory management algorithms and close to the optimal solution based on Knapsack problem.

This chapter is organized as follows. The next section describes prior work, and introduces OSGi and the corresponding memory model. Section 3 presents an application scenario with counter example and a formal definition of the memory management problem. In Section 4 we first present the simple algorithms that are based on traditional methods and then propose our SD Heuristic algorithm and an analytical solution (SD Optimal) based on Knapsack problem. Evaluation results and discussion are presented in Section 5. Finally, conclusions and outlook are summarized in Section 6.

## 2. Prior work

Memory management has been discussed extensively in the operating systems literature. For the sake of comparison, we adopted two well-know memory management techniques, namely, best-fit, worst-fit and compared them with our proposed protocols in Section 4. Due to the different architectures, one of the main differences between memory management for smart home applications and general computer applications is that the former takes into account the dependencies among different services or bundles, as explained later. To our best knowledge, there is no study related to the memory management in the context of smart home applications. Vidal et al. (2006) addressed QoS in home gateway. They proposed a flexible architecture for managing bandwidth inside the home. However, they have not addressed memory management in home gateways. Aliet al. (2005) proposed architecture based on OSGi for wireless sensor networks, where data is processed in distributed fashion. They showed how to execute simple database queries like selection and join in a distributed fashion. Bottaro et al. (2007) addressed protocol heterogeneity and interface fragmentation when connecting several devices to OSGi-based gateway at home. The paper describes different scenarios and challenges for providing pervasive services in home applications.

### 2.1 OSGi framework introduction

Due to the different characteristics from traditional computer architecture, Ericsson, IBM, Motorola, Sun Microsystems found OSGi Alliance, which is an open standards organization in March 1999. Developed by this alliance, OSGi is a Java-based framework and (Helal et al. 2005; Lee et al. 2003) and wireless networks (Helal et al. 2005). The OSGi framework is completely based on Java technology. In fact, the specification itself is just a collection of standardized Java APIs plus manifest data. The use of Java technology has several important advantages. First, Java runtimes are available on almost all OS platforms, allowing the OSGi framework and services to be deployed to a large variety of devices across many different manufacturers. Java also offers superb support for secure mobile code provisioning, which allow developers to package and digitally sign a Java applications and send them over the network for remote execution. If the execution host cannot verify the digital signature or determines that the application does not have sufficient permission, it

could reject the application or put it in a sandbox with limited access to local resources. Furthermore, Java has an extensive set of network libraries. It supports not only HTTP and TCP/IP networking, but also advanced peer-to-peer protocols such as Jini, JXTA and BlueTooth. Services are implemented as plug-ins modules called bundles. (We will use the terms "bundle" and "service" interchangeably in the rest of this chapter). These bundles can be downloaded from the application service providers through the Internet when they are requested. Examples for services that are used for application development are Java development tools, J2EE monitor, crypto services, bundles that provide access to various relational database management systems (e.g., DB2, Oracle, etc.), HTML creation, SQL, Apache, Internet browser, XML plug-ins, communication with Windows CE, etc. Other system administration bundles like core boot, web application engine, event handling, OSGi monitor, file system services, etc. Bundles for various Internet and network protocols, like, HTTP service, Web services, SMS, TCP/IP, Bluetooth, X10, Jini, UPnP, , etc. There are many bundles that are already implemented by OSGi partners (Binstock 2006).

### 2.2 Service dependency graph

Figure 2 shows the software architecture of a gateway according to the OSGi model. Some of the basic bundles, which implement essential services, are already loaded in the gateway framework. The framework handles the basic bundle management functionality, e.g., install, uninstall, start, stop, communication, etc. Other service bundles, developed by the third party like device manufacturers and services providers, can be downloaded in real-time by the OSGi framework as needed.

Our proposed algorithm is implemented as a part of the framework to provide memory management. The gateway can download the corresponding bundles (that correspond to specific services) when it becomes necessary. In order to share its services with others, a bundle registers any number of services to the framework.

A bundle may import services provided by other bundles and therefore, its running may depends on other bundles. For example, a file downloading bundle needs services provided by an UDP service bundle or TCP service bundle, therefore, it is dependent on UDP service bundle or TCP service bundle. To model the relationship among services, we use a dependency graph. Formally, given a set of service instances $S = \{ s_1, ..., s_n \}$ which currently reside in main memory, let $G\ (S,E)$ be a directed acyclic graph with vertex set $S$ and edge set $E$, describing the dependence among these instances. There is a directed edge from $s_{ij}$, $(s_i,s_j) \in E$ if and only if $s_i$ depends on $s_j$. Since it is natural to assume that each application instantiates its own copy of a given service, the dependency graph will consist of a forest of rooted trees, where each tree represents the service instances instantiated by a given application as shown in Figure 3.

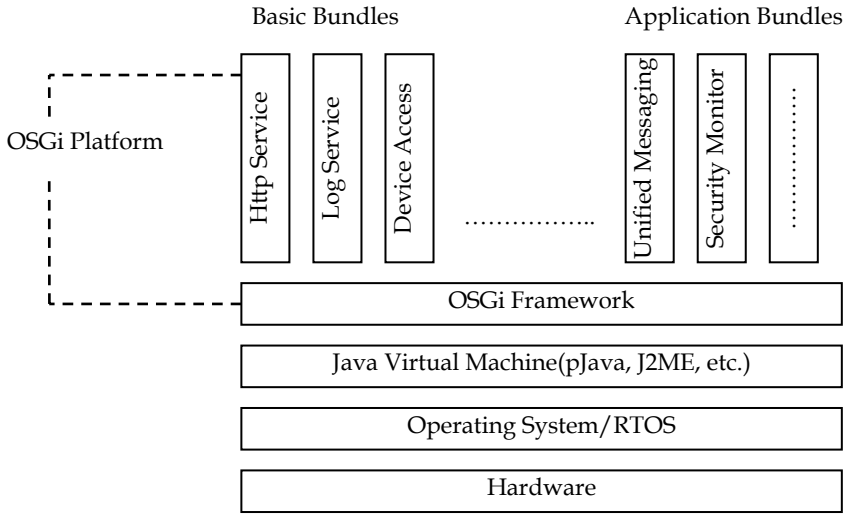



Fig. 2. Software architecture of a gateway in the OSGi model.

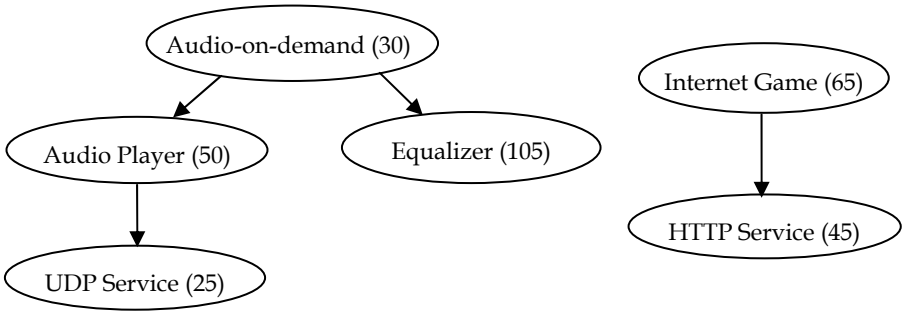



Fig. 3. Dependence graph for two currently running applications (numbers indicate memory requirements).

## 3. Problem definition

The gateway might need to free memory space to accommodate new services that are triggered by connecting a new device to the network or upon explicit local or remote requests. Although the amount of memory required to execute a service might change with time, the application service provider (or the author who provides the bundle) can give approximate statistical estimates of the amount of memory required to execute the services such as average, median, or maximum. Moreover, extra memory space might be requested by any one of the service instances (inside the home gateway) to continue its service. If such memory is not available, the gateway has to pick a victim service instance (or instances) to terminate to allow the new application to start. Note that because many of the smart home

applications are real-time in nature, thus, the gateway tends to terminate the victim service rather than suspending it. Ideally, the gateway memory management algorithm needs to meet the following desirable properties:

- The total amount of reclaimed memory is enough to fulfill the requested memory.
- The number of victim service instances should be minimal.
- Since the algorithm will be executed in real-time, it should be fast and does not require much memory itself.

### 3.1 Application scenario

The problem addressed in this chapter can be better described by the following motivating example. Suppose that there are two applications that are running on the gateway. The first application uses audio-on-demand service that depends on the audio player service, which in turn, depends on the UDP service. The service dependency graph for the audio-on-demand is shown in Figure 3. The second application is an Internet game, which consists of two services; a game service that depends on HTTP service. Now we would like to start one more application, for example, home security. Let us assume that there is no more free memory in the gateway and the total memory required by the home security application is 100 (memory units).

Apparently, home security application is more important than Audio-on-demand application and Internet-game. Thus, it is reasonable to kick out at least one of these services to start the home security application. As shown in Figure 3, the equalizer service uses 105 memory units. A wise decision would be to kick the equalizer service that belongs to the audio-on-demand application because it results in killing less number of service instances and fulfills the memory demand.

The challenge is to select those services to kick out from memory such that the number of services and applications affected is minimal and that the total memory reclaimed equal or greater than the memory requested. In Section 4, we propose new algorithms for service replacement for memory management in gateways.

### 3.2 Formal description of the problem

More formally, our problem can be described as follows. Let $S = \{ s_1, ..., s_n \}$ be the set of service instances currently resident in gateway memory (Table 1). Service instance $s_i$ occupies $M(s_i)$ storage. Let $G\ (S,E)$ be the forest of trees describing the dependency among the set of instances $S$. For a vertex $v$ in $G$, let us denote by $T(v)$ the set of vertices of the sub-tree of $G$ rooted at $v$ (including $v$ itself), and for a subset of vertices $V \subseteq S$, let $T(V) := U_{v \in V} T(v)$.

Given that a new service instance $s$, with memory requirement $M(s)$ has to be created, it might be required to remove some currently existing instances in order to free room for the new instance. Assume that the extra required memory for this operation is $M_t$ units, that is $M_t = M(s) - M_f$, where $M_f$ is the current amount of available memory. Here we assume that, when a service instance is terminated, all instances depending on it will be terminated and

removed as well. Our goal is to reduce the number of removed (stopped) services. More precisely, it is desired to find a subset $V \subseteq S$ of minimal number of service instances, whose ejection, together with all its dependents, will make available a total memory of at least $M_t$ units. Letting $M(S') := \Sigma_{s \in S'} M(s)$ for any $S' \subseteq S$, our problem can be formulated as finding

$$\min \{ | T(V) | : V \subseteq S, M(T(V)) \geq M_t \} \quad (1)$$

This last problem is closely related to the well-known Knapsack problem, which is NP-hard in general (Garey & Johnson 1979). However, the Knapsack problem admits a pseudo polynomial algorithm which runs in $O(n^2)$ (see, for example, Jain & Vazirani 2001). This solution is discussed in detail in Section 4.3.

| Term | Description |
|---|---|
| Service | A self-contained component that performs certain functionality |
| Bundle | The functional and deployment unit for shipping services |
| Service instance | The execution thread of a bundle in the framework |
| Applications | Consists of one or more services |

Table 1. Description of the terminology used in the chapter.

## 4. Service replacement algorithms

In this section we present several algorithms for solving service replacement problem in home gateway. The first three algorithms are direct adaptation of the well-known first-fit, best-fit and worst-fit algorithms which select the service(s) to be replaced based on the amount of memory that might become available. The other two algorithms take into account not only the memory size but also the service dependencies. SD (Size-Dependency) heuristic is a simple heuristic that runs in $O(nh)$ time and requires linear space, where $h$ is the height of the forest. Finally SD Optimal algorithm computes an optimal solution in $O(n^2)$ time and $O(nh)$ space.

As explained in the previous two sections each application is modeled as a tree of service nodes that are used by this application. The algorithm showed in Figure 4 selects a victim node (root) $X$. Note that $X$ can be either an application (a root of a tree) or a sub-tree that belongs to an application. If $X$ is the root node then the gateway will stop the corresponding application. But if $X$ was a sub-tree under an application, then deleting $X$ might stop some functionalities of the application without terminating the whole application. Our proposed model and solution work equally for both cases.

In some special cases, the application might still run with reduced functionality as a result of stopping the sub-tree rooted at $X$. However, it is highly possible that the deletion of the sub-tree $X$ seriously affect the execution of the application and consequently the whole application stops. Even though our algorithms work for both cases, in our experiments we count on the general case that deleting any sub-tree stops the whole application. Thus, without loss of generality, the following discussions and algorithms check only root nodes (application node) and not a sub-tree. In general service replacement algorithm is shown in

Figure 4. The differences between the different techniques are materialized in the way the algorithm selects the next victim for deletion in Step 2.1 in Figure 4. It is easy to see that the above heuristic can be implemented in *O(nh)* time and *O(n)* space. In the next two sections we discuss several alternatives for picking the victim bundle.

**Generic ($G$, $S$, $M_t$)**
Input: The current set of service instances $S$, the dependence forest $G$, and the memory requirement $M_t$.
Output: A new dependent forest $G$, describing the dependency among the bundles remaining after deleting a set of bundles whose total memory is at least $M_t$.

1. For each node $s \in S$, compute the accumulative size and memory:
   $c(s) \leftarrow |T(s)|$ and $m(s) \leftarrow M(T(s))$ using breadth first search on the dependency forest $G$
2. While $M_t > 0$
   2.1 Pick a victim node $s$, according to the selection strategy to be described later.
   2.2 Delete $s$ and all its dependents
   2.3 For every node $u$ on the path from $s$ to the root of the tree containing $s$, set
   $c(u) \leftarrow c(u) - c(s)$ and $m(u) \leftarrow m(u) - m(s)$
   2.4 Update $M_t \leftarrow M_t - m(s)$

Fig. 4. General service replacement algorithm.

### 4.1 The simple algorithm

This algorithm is similar to the traditional algorithms used in operating system literature for memory management in general purpose computers. These traditional techniques make selection based on the amount of memory used and ignore the dependencies. We modify these techniques to take into consideration the total *accumulative* memory of each service (bundle) resulting from stopping one service. We consider the following three algorithms:

**First Fit:** choose the first service $s$ in the list $S=\{ s_1, ..., s_n \}$ such that total memory $M(T(s))$ occupied by its sub-tree is *at least* the requested amount $M_t$:

$$k \leftarrow \min \{ 1 \le j \le n \mid M(T(s_j)) \ge M_t \}; \qquad s \leftarrow s_k$$

If no such node exists ($k=\infty$), pick the node with largest $M(T(s))$:
$s \leftarrow \text{argmax}\{ M(T(s)) : s \in S\}$
where argmax{ ... } denotes a maximum of a given function.

**Best Fit:** choose the service $s \in S$ with the smallest total memory that is $\ge M_t$:

$$s \leftarrow \text{argmax} \{ M(T(s)) : s \in S, M(T(s)) \ge M_t \}$$

where argmax{ ... } denotes a maximum of a given function.
If no such node exists ($k=\infty$), pick the node with largest $M(T(s))$.

**Worst Fit:** choose the service $s \in S$ with the largest total memory:

$$s \leftarrow \text{argmax}\{ M(T(s)) : s \in S\}$$

If no such node exists ($k=\infty$), pick the node with largest $M(T(s)$.

### 4.2 SD heuristic

Different from the simple algorithms discussed above, our proposed heuristic greedily tries to pick, as a victim for deletion, the service instance whose removal will free the minimum amount of memory larger than $M_t$ and, at the same time, it has the smallest number of dependents. Towards this end, the heuristic will pick for deletion the service instance $s$ which maximizes the ratio of the total memory to the number of dependents:

$$s \leftarrow \text{argmax}\{ M(T(s))/ \mid T(s) \mid \ : s \in S\}$$

This selection tends to decrease the number of deleted instances. Looking back at the example in Figure 3, we can see that the ratios $M(s^*)/ \mid s^* \mid$ for the different service instances are as follows: Audio-on-demand (52.5 = (30 + 50 + 25 + 105)/4), Audio-player (37.5=(50+25)/2), UDP Service (25), Equalizer (105), Internet game (55=(65+45)/2), and HTTP Service (45). Thus, the service instance with maximum ratio is the Equalizer whose removal will give enough memory to start the new service (requiring 100 memory units). Should we have used the First Fit strategy, on the other hand, we might have selected to remove the Audio-on-demand instance, which results in removing four instances instead of only one. Note also that, for this particular example, the Best Fit algorithm would also remove the same instance (the Equalizer) selected by the SD Heuristic.

### 4.3 SD optimal

It is well known that the Knapsack problem admits a pseudo polynomial algorithm. In this section, we extend this solution to problem 1 using dynamic programming (Johnson & Niemi 1983). Specifically, let $S = S_n = \{s_1,...,s_n\}$ be the current set of service instances listed in post-order traversal (that is, we recursively traverse the children from left to right then we traverse the root). We shall consider incrementally the sets $S1 = \{s_1\}$, $S2 = \{s_1, s_2\}$, $S3 = \{s_1, s_2, s_3\}$, ... , computing for each set the maximum amount of memory that can be achieved by deleting a subset of nodes. In order to compute these maxima, we will need to compute for each node $s_i$, the largest index $k \in \{1,..., i-1\}$ such that $s_k$ is *not a descendant* of $s_i$. Let $L(s_i)$ denotes such an index. The following procedure gives the post-order traversal of a given forest and computes the required indices $L(s_i)$ for each $i=1,...,n$.

If the connected components of the forest $G$ are $C1, C2, ..., Cr$, then in order to compute the post-order traversal for G, the above procedure is called $r$ times.

**Traverse ($v$, $G$, $k$)**
Input: a sub-tree of the dependence forest $G$ rooted at $v$ and an integer $k$.
Output: the post-order traversal $\{ s_k, s_{k+1}, \ldots, s_{|T(v)|+k+1} \}$of $T(v)$, and the set of indices $\{L(s) \mid s \in T(v)\}$

1. If $|T(v)|=0$ // *tree is empty*
   return
2. If $|T(v)|=1$ // *v is a leaf node*
   $L(v) \leftarrow k$
3. else for each child $u$ of $v$:
   Traverse($u$,$G$,$k$);
   $L(v) \leftarrow L(\text{leftmost}(v))$
4. $k \leftarrow k+1$
5. $s_k \leftarrow v$

In what follows, nodes $u, v \in V(G)$, are considered to be incomparable if neither is a descendant of the other, i.e., $v \notin T(u)$ and $u \notin T(v)$. Note that $n$ is a trivial upper bound on the total number of instances (or weight) that can be achieved by any solutions.

**Traverse-Forest ($G$)**
Input: the dependence forest $G$
Output: the post-order traversal $\{s_1,...,s_n\}$ of $G$, and the set of indices $\{L(s) \mid s \in V(G)\}$

1. Find the connected components *C1, C2, ..., Cr* of $G$
2. $k \leftarrow 0$
3. For $i$ =1 to $r$
   Traverse (root ($C_i$), $G$, $k$)

For each $i \in \{1,...,n\}$ and each $w \in \{1,...,n\}$, let $S_{i,w}$ denote a subset of incomparable elements of $S_i = \{s_1,..., s_i\}$, whose total weight is exactly $w$, and whose total memory is maximized. Let $A(i,w)=M(T(S_i,w))$ if the set $S_{i,w}$ exists, and $A(i,w) = -\infty$ otherwise.

$$|T(S)|=w,\ A(i,w) = \begin{cases} -\infty \text{ if the set } S_{i,w} \text{ does not exist} \\ 0, \text{ if } i=0 \text{ or } w=0 \end{cases}$$

Clearly $A(1,w)$ is known for every $w \in \{1,...,n\}$. The other values of $A(i,w)$ can be computed incrementally using the following recurrence:

$$A(i+1, w) = \max\{A(i,w), M(s_{i+1}) + A(L(s_{i+1}), w - |T(s_{i+1})|)\} \quad (2)$$

if $|T(s_{i+1})| \le w$ and $A(i+1,w)=A(i,w)$ otherwise.

Proof of Eq. 2: Let $S' \subseteq S_{i+1}$ be a subset of incomparable elements that achieves $A(i+1, w)=\max\{M(T(S)) \mid S\ S_{i+1}, |T(S)|=w\}$. There are two possible cases:

Case 1: $s_{i+1} \notin S'$. Then $S' \subseteq S_i$ achieves $A(i,w) = \max\{M(T(S)) \mid S \subseteq S_i, \mid T(S) \mid = w\}$.
Case 2: $s_{i+1} \in S'$. Let $S'' = S' \setminus \{s_{i+1}\}$. Since the elements of S′ are incomparable and the dependence graph is a forest, we have $T(S') \cap T(s_{i+1}) = \emptyset$, and therefore,

$$\mid T(S'') \mid = \mid T(S') \mid - \mid T(S_{i+1}) \mid \text{ and } M(T(S'')) = M(T(S')) - M(T(s_{i+1})).$$

By the definition of $L(s_{i+1})$, we know that for $L(s_{i+1})+1 \le j \le i$, $s_j$ is a descendant of $s_{i+1}$, i.e., $T(s_j) \cap T(s_{i+1}) \neq \emptyset$, implying that $S''$ must be a subset of $S_k$, where $k = L(s_{i+1})$. Thus $S'' \subseteq S_k$ is a subset that achieves $A(i,L(s_{i+1})) = \max\{M(T(S)) \mid S \subseteq S_k, \mid T(S) \mid = w - \mid T(s_{i+1}) \mid\}$, which when combined with $s_{i+1}$ gives $M(T(S')) = M(T(S'')) + M(T(s_{i+1})) = A(i,L(s_{i+1})) + M(T(s_{i+1}))$.

Equation 2 then follows by taking the maximum achievable memory over cases 1 and 2. □

Now we state the optimal algorithm.

*Optimal* ($G$, $S$, $M_t$) The current set of service instances $S$, the dependence forest $G$, and the memory requirement $M_t$.

Output: A new dependence forest $G$, describing the dependence among the bundles remaining after deleting a set of bundles whose total memory is at least $M_t$.

1. For each node $s \in S$, compute the accumulative size and memory:
   $c(s) \leftarrow \mid T(s) \mid$ and $m(s) \leftarrow M(T(s))$
2. Call Traverse-forest(G) to get the post-order traversal $\{ s_1, ..., s_n\}$ of $G$, and the set of indices $\{ L(s) \mid s \in V(G)\}$.
3. Initialize:
   $A(i,0)=0$ for all $i=1,...,n$,
   $A(0,w)=0$ for all $w=1,...,n$,
   $A(1,1)=m(s_1)$, and $A(1,w)=-\infty$ for all $w=2,...,n$.
   *// Build a dynamic programming table*
4. For $i=1$ to $n$
5. For $w=1$ to $n$
   if $c(s_{i+1}) \le w$
      if $A(i,w) \ge m(s_{i+1}) + A(L(s_{i+1}), w - c(s_{i+1}))$
         $A(i+1,w) \leftarrow A(i,w)$, $B(i+1,w) \leftarrow 0$
      else
         $A(i+1,w) \leftarrow m(s_{i+1}) + A(L(s_{i+1}), w - c(s_{i+1}))$, $B(i+1,w) \leftarrow 0$
   else
      $A(i+1,w) \leftarrow A(i,w)$, $B(i+1,w) \leftarrow 0$.
   // now compute optimal solution
6. $S \leftarrow \emptyset$; $i \leftarrow n$; $k \leftarrow min\{w \in [n]: A(i,w) \ge M_t\}$.
7. while $i > 0$
   if B $(i,k) = 1$
      $S \leftarrow S \cup \{ s_i \}$; $i \leftarrow L( s_i )$; $k \leftarrow k - c(s_i)$.
   else
      $i \leftarrow i-1$.
8. For each $s \in S$, delete $T(s)$.

Thus we get an O($n^2$) time, O($nh$) space algorithm for solving problem 1.

## 5. Performance evaluations

We carried extensive studies to evaluate the proposed algorithms. First, we compared the performance of the different algorithms in terms of the number of removed services to verify our new proposed algorithms. And then evaluate the algorithm execution time to show that the SD heuristic is practical in a home gateway. We considered different scenarios e.g., different distributions of bundle (or service) sizes, different number of existing bundles, etc. First we describe how the experimental data is generated, and then we present our results.

### 5.1 Experiment setup

Initially, services are generated with random sizes and loaded into the gateway memory, until the memory becomes almost full. Each service can depend on a number of randomly selected services with probability varying from 0 to 1. Service sizes are selected randomly in the range from 100 Kb to 50 Mb according to different probability distributions: uniform distribution in the given range, exponential distribution with a mean 5M, and a normal distribution with a mean of 5M.

Because home gateways are new, it was difficult to find real data (traces) of the service arrival. In our experiments, we used statistical service arrival model. We used both uniform distribution and exponential distributions for new service arrival to the home gateway. We conducted experiments to compare the performance of the following algorithms:

- Traditional algorithms: Best-fit and Worst-fit
- SD heuristic
- SD Optimal algorithm

A new service, with memory requirement varying uniformly 100K–50M, is created. We find out which services (bundles) should be kicked out to make enough room for the incoming bundle. Two performance measures were considered:

1. The number of services need to be stopped (or kicked out) to free enough space for the new service
2. The cost of each algorithm, in terms of execution time, required to determine the victim services (bundles).

Each performance measure was averaged over 1,000 experiments.

### 5.2 Experimental results

In our first experiment, we fixed the number of existing bundles in the home gateway and then compared how the different algorithms behave in terms of the number of kicked out services, as the size of the new coming service ($s_{new}$) is increased from 100K to 50M. In all our experiments, we assumed uniform and exponential service arrival. However, service

arrival distribution does not affect the number of victim services. In Figures 5, 6 and 8, service arrival is assumed to be uniform. Exponential distribution gives similar results and thus not shown. Figure 5 shows our results when the number of services currently running in the gateway=100. Just as we have expected, it can be seen from Figure 5, the SD heuristic and the SD Optimal perform much better than the traditional techniques. This result verifies that our proposed algorithms perform much better than the traditional techniques, after taking the dependency between different bundles into account. We also note that the SD heuristic performs very close to the SD Optimal for various size of the new service $s_{new}$.

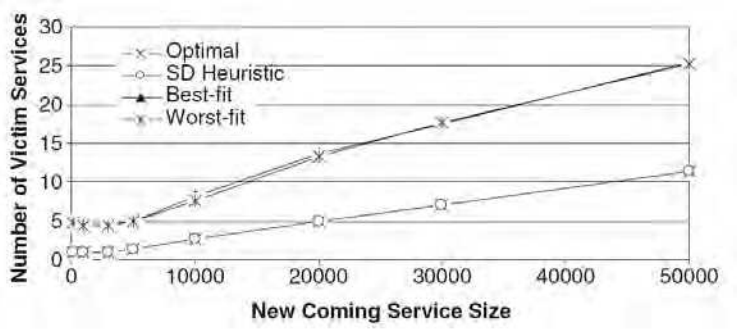



Fig. 5. Performance of the different algorithms as function of $S_{new}$ for uniform distribution.

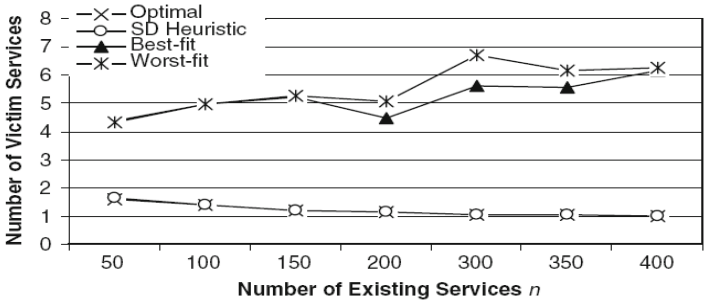



Fig. 6. Performance of the different algorithms as function of $n$ for uniform distribution.

In the second experiment, we compare the performance of the different algorithms as the number of existing bundles $n$ is increased. The result is shown in Figure 6. As we can see from the result, the performance of SD optimal and SD heuristic remain almost invariant under the change of number of bundles. The performance of the traditional techniques, on the other hand, degrades as the number of services running in the gateway increases. This can be explained as follows. With a large number of existing bundles, the chances that the memory requirement will be fulfilled with a few number of bundles from the lower levels (i.e., having a few levels of descendants) is higher. Since SD heuristics and SD optimal take dependencies into consideration, the likelihood to find better solution increases with the increasing of the number of existing services. Their performance will improve with the increase in chances of finding bundles which have less dependent bundles, and therefore, fewer services are terminated. On the other hand, the traditional techniques do not consider

the dependencies between different services in the OSGi platform and provide no optimization, and therefore, might have to delete a few bundles from the top levels, resulting in a much higher number of kicked out bundles.

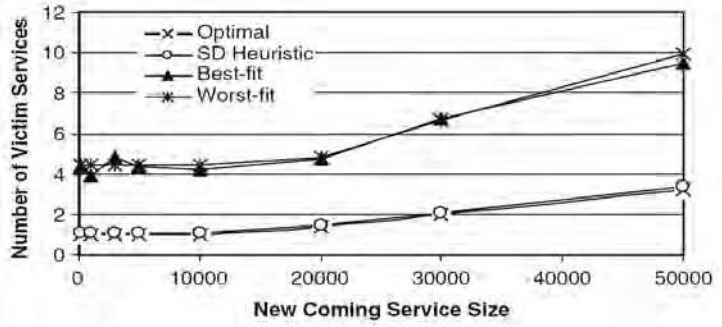



Fig. 7. Performance of the different algorithms as function of $s_{new}$ for exponential distribution.

In the next experiment, we examined the effect of using a non-uniform distribution on the performance of the algorithms. We used an exponential distribution with mean 5M for the size of the existing bundles. Figure 7 presents our results for this experiment. Clearly, the number of kicked out bundles has decreased relative to the uniform case, since in this case it is easier to satisfy the memory requirement with a smaller number of bundles. However, we notice that the relative performance of the different algorithms remains invariant.

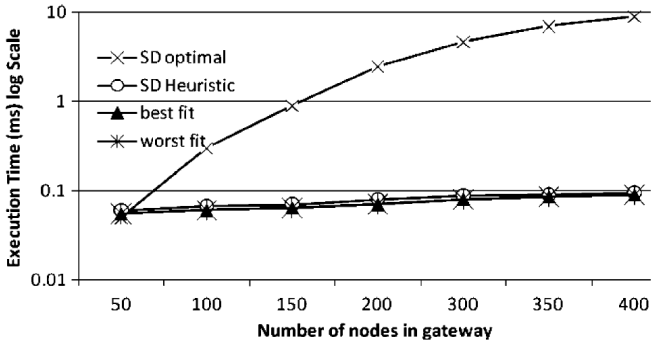



Fig. 8. Running time of the different algorithms as function of $s_{new}$ for uniform distribution.

From the above experiment results, we can see that the SD heuristic gives satisfactory results in terms of the number of kicked bundles, as compared with the SD optimal algorithm. At the same time, SD heuristic significantly outperforms the traditional techniques, e.g., best fit and worst fit. This naturally raises the question of whether SD heuristic is practical in terms of running time, as compared to the traditional techniques. To answer this question, we carried experiments that compare the execution time of the different algorithms. The results are shown in Figure 8. The y-axis shows the response time of each algorithm in milliseconds; the x-axis shows the number of services running in the gateway. As we see from this figure,

while the optimal algorithm is significantly slower than the others, SD heuristics performs very well compared to the traditional techniques in terms of their running time. It is just what we have expected.

## 6. Conclusions

In this chapter, we have considered the problem of managing services and bundles in home gateways with limited amount of main memory. Because of the different architecture of home gateway using OSGi from the traditional computer architecture, a key difference between our problem and the traditional memory management is that the dependencies among different services have to be taken into consideration for a higher customers' satisfaction.

We use a dependency graph to model the relationship among services. This chapter proposes two algorithms. The first one is an extension of Knapsack problem which finds the optimal solution in a polynomial time. The second one is a heuristic that spans the dependency graph and tries to free the required amount of memory while minimizing the number of terminated services. We compared the proposed techniques with the traditional memory management algorithms such as the best fit and worst fit. Our experimental results indicate that SD (service dependency) heuristic is a good candidate for use in practical environments, as its performance is close to the optimal solution in terms of the number of stopped services. SD heuristic performs much better than the traditional memory management techniques. From the execution time point of view, SD heuristic is almost as fast as the traditional memory management techniques.

In this chapter, we have not taken into account of the priorities of different services. Our future work will focus on extending the proposed model to include the service priority. Different services may have different priority which determined by their specific characteristics or set by users. For example, an Internet game should not force out from the gateway a home security service (which is much more important than the internet game). Each service defines a priority value that reflects the importance of this service relative to other services. We will introduce the priority as a new factor in both the heuristic and the optimal solution.

## 7. References

Ali, M., Aref, W., Bose, R., Elmagarmid, A., Helal, A., Kamel, I., &Mokbel, M. (2005). NILE-PDT: A phenomenon detection and tracking framework for data stream management systems. In Proceedings of the Very Large Data Bases Conference, August.

Binstock, A. (2006). OSGi: Out of the gates. Dr. Dobb Portal, January.

Bottaro, A., Gérodolle, A., & Lalanda, P. (2007). Pervasive service composition in the home network. In Proceedings of the 21st International IEEE Conference on Advanced Information Networking and Applications, Niagara Falls, Canada, May.

Garey, M., & Johnson, D. (1979). Computers and intractability. New York: Freeman.

Helal, A., Mann, W., El-zabadani, H., King, J., Kaddoura, Y., & Jansen, E. (2005). Gator Tech Smart House: A programmable pervasive space. IEEE Computer, 38(3), 50–60.

Ishihara, T. (2006). Home Gateway architecture enabling secure appliance control service. In Proceedings of the 10th International Conference on Intelligence in Network (ICIN'06).

Ishihara, T., Sukegawa, K., & Shimada, H. (2006). Home Gateway enabling evolution of network services. Fujitsu Science Technical Journal, 24(4), 446–453.

Jain, K., & Vazirani, V. V. (2001). Approximation algorithms for metric facility location and k-Median problems using the primaldual schema and Lagrangian relaxation. Journal of the ACM, 48 (2), 274–296.

Jansen, E., Yang, H., King, J., Abdul Razak, B., & Helal, A. (2006). Acontext driven programming model for pervasive spaces. In 4thInternational Conference on Pervasive Computing, May.

Johnson, D. S., & Niemi, K. A. (1983). On Knapsacks, partitions, and a new dynamic programming technique for trees. Mathematics ofOperations Research, 8(1), 1–14.

King, J., Bose, R., Pickles, S., Helal, A., Vander Ploeg, S., & Russo, J.(2006). Atlas: A service-oriented sensor platform, the 4th ACMConference on Embedded Networked Sensor Systems (Sensys), Boulder, CO, USA.

Lee, C., Nordstedt, D., & Helal, A. (2003). OSGi for pervasive computing. the Standards, Tools and Best Practice Department, IEEE Pervasive Computing, A. Helal, Dept. Editor, Volume 2, Number 3, September.

Maples, D., & Kriends, P. (2001). The open services gateway initiative: An introductory overview. IEEE Communication Magazine, 39(12), 110–114.

Margherita2000, The first washing machine on the Internet. http://www.margherita2000.com/sito-uk/it/home.htm.

Microsoft Corporation, (2008). Universal plug and play device architecture reference specification, version 2.0. http://www.upnp.org/.

Ryu, I. (2006) Home network: Road to ubiquitous world. In Proceedings of the International Conference on Very LargeDatabases (VLDB).

Silberschatz, A., & Peterson, J. (1989). Operating system concepts. Boston, MA: Addison Wesley.

Sommers,F.(2006). Dynamic clustering with Jini Technology. www.artima.com/lejava/articles/dynamic_clustering.html, January.

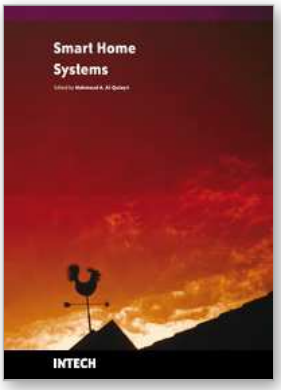
Sun Microsystems Inc. (2007) Jini architectural overview. http://www.jini.org/.

The OSGi Alliance. (2009). The OSGi Service Platform release 4 core specification Ver 4.2. http://bundles.osgi.org/browse.php, September.

Vidal, I., García, J., Valera, F., Soto, I., & Azcorra, A. (2006). Adaptive quality of service management for next generation residential gateways. In Proceedings of the 9th International conference on Management of Multimedia and Mobile Networks and Services, Ireland, Dublin.

Watanabe, K., Ise, M., Onoye, T., Niwamoto, H., & Keshi, I. (2007). An energy-efficient architecture of wireless home network basedon MAC broadcast and transmission power control. IEEETransaction on Consumer Electronics, 53(1), 124–130.

Zigbee Alliance, (2004). Zigbee specification: Zigbee document 053474r06 Version 1.0, 14 Dec.




**Smart Home Systems**

Edited by Mahmoud A. Al-Qutayri

ISBN 978-953-307-050-6
Hard cover, 194 pages
**Publisher** InTech
**Published online** 01, February, 2010
**Published in print edition** February, 2010

Smart homes are intelligent environments that interact dynamically and respond readily in an adaptive manner to the needs of the occupants and changes in the ambient conditions. The realization of systems that support the smart homes concept requires integration of technologies from different fields. Among the challenges that the designers face is to make all the components of the system interact in a seamless, reliable and secure manner. Another major challenge is to design the smart home in a way that takes into account the way humans live and interact. This later aspect requires input from the humanities and social sciences fields. The need for input from diverse fields of knowledge reflects the multidisciplinary nature of the research and development effort required to realize smart homes that are acceptable to the general public. The applications that can be supported by a smart home are very wide and their degree of sophistication depends on the underlying technology used. Some of the application areas include monitoring and control of appliances, security, telemedicine, entertainment, location based services, care for children and the elderly… etc. This book consists of eleven chapters that cover various aspects of smart home systems.



**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821