Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Programação de Dispositivos Móveis
Aula 3
Licenciatura em Engenharia Informática
Licenciatura em Informática Web

Programming of Mobile Devices
Lecture 3
Degree in Computer Science and Engineering
Degree in Web Informatics

Sumário
Discussão da pilha de software que define a plataforma Android™, bem como das 4 camadas que a compõem. Apresentação e discussão de algumas ferramentas para depuração de aplicações Android™.

Summary
Discussion of the software stack that defines the Android™ platform, as well as of the 4 layers it contains. Presentation and discussion of some of the tools availavle for debugging Android™ applications.

1 A Plataforma Android™
The Android™ Platform

1.1 Introdução
Introduction

Estudos de mercado recentes mostram que **o Sistema Operativo Android™ corre em cerca de 85% de todos os *smartphones* em utilização** e a nível mundial[1]. Só no ano de 2016 estima-se que foram **vendidos cerca de 255 milhões destes dispositivos com Android™**. Em alguns países, **os programadores de Android™ são dos mais bem pagos** nesta indústria. A plataforma e a procura de profissionais capazes **está ainda a crescer** rapidamente.

Ultimamente tem-se assistido, contudo, à tentativa de **estender a utilização do Android™ a outros dispositivos**, nomeadamente ***smart TVs***. A popularidade da plataforma, o facto de ter o código aberto e ser **relativamente simples desenvolver aplicações para a mesma**, tem ditado **uma evolução fora do comum**, tornando-a hoje **num sistema bastante completo e complexo, no sentido de albergar já um vasto conjunto de componentes**.

1.2 Pilha de Software
Software Stack

De uma maneira geral, pode dizer-se que **a plataforma Android™ é composta por**:

1. **uma pilha de software, com várias camadas**, desenhada para permitir a construção e execução de aplicações móveis;
2. **um kit de desenvolvimento de software** (da designação *Software Development Kit* (SDK)); e
3. uma extensa **documentação**.

Esta pilha de software foi **desenhada sobretudo, mas não exclusivamente, para dispositivos móveis**, nomeadamente *smartphones* e *tablets*. É composta por **4 camadas** que se estendem desde **o nível do núcleo do Sistema Operativo (SO) até às aplicações que o utilizador pode manipular** (e.g., *browser*, atendedor de chamadas ou aplicação de e-mail). A pilha de software, bem como alguns dos seus componentes e organização costuma ser representada como se mostra na figura **??** [2]. Cada uma das camadas vai ser alvo de uma breve discussão nas secções seguintes.

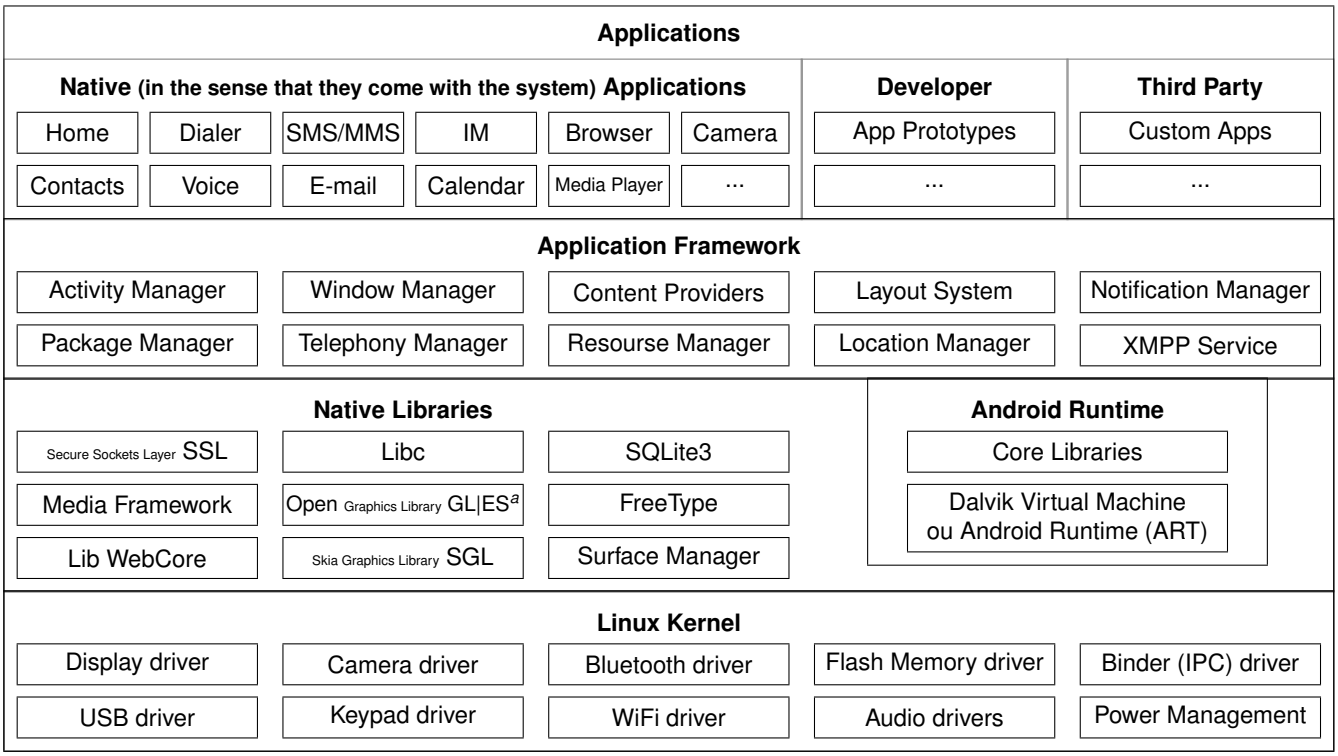
1.3 Camada do Núcleo Linux
Linux kernel Layer

A **camada mais baixa** da pilha de software compreende **o próprio núcleo do SO**. É esta camada que **fornece os serviços base** de que dependem as restantes funcionalidades de qualquer dispositivo com Android™. Por usar **o *kernel* Linux**, disponibiliza muitos dos serviços que nele estão implementados, nomeadamente **uma arquitetura de permissões** (para restringir acesso a recursos), **mecanismos padrão de gestão de memória e de processos**, **suporte a comunicações entre-processos**, **operações de baixo nível de leitura e escrita em ficheiros**, e **comunicações em rede**. É nesta camada que são colocados **os *drivers*** que podem ser adicionados ou incluídos para suporte de **dispositivos de hardware adicionais** como câmaras fotográficas, antenas de rádio ou sensores.

[1] Ver, e.g., https://www.idc.com/promo/smartphone-market-share/os.

[2] Esta figura, ou a original que circula pela Internet, é das mais populares em projetos de fim de curso com desenvolvimento de aplicações Android™.

Note que **o núcleo Linux usado no Android™ não é exatamente igual ao núcleos tipicamente usados para *desktops***. O facto é que os dispositivos móveis **têm funcionalidades e necessidades diferentes** dessas máquinas, pelo que tiverem de ser feitas **adaptações**. Por exemplo, o núcleo utilizado no Android™ tem **os seus próprios gestores de energia e de memória**, mais porque os dispositivos móveis são tipicamente alimentados por baterias e podem ser mais limitados em termos de recursos computacionais. Por exemplo, o ***low memory killer* constitui uma das funcionalidades do gestor de memória**. Outro bom exemplo é o mecanismo de **comunicação entre-processos** do Android™ (*binder*), que permite que os processos partilhem memória e informação de uma **forma simples e eficiente**.



[a]ES abrevia *Embedded Systems*.

Figura 1: Pilha de software da plataforma Android™.

1.4 Bibliotecas Nativas

Native Libraries

A **segunda camada** (a contar de baixo) **contém as bibliotecas nativas** disponibilizadas pela plataforma, nomeadamente a `bionic libc`, `sqlite` e SSL. Estas bibliotecas nativas são normalmente **implementadas em C ou C++**, e estão encarregues de **atividades criticas relacionadas com o desempenho do dispositivo**, como por exemplo **refrescar o ecrã** (feito pelo *Surface Manager*) ou **renderizar páginas web** (da responsabilidade do *LibWebcore*). A biblioteca `sqlite` **permite a gestão de bases de dados** em aplicações Android™. **A** `bionic libc` concretiza outra diferença para sistemas Linux padrão, já que esta também **foi adaptada para Android™** tendo em conta as particularidades do núcleo e do dispositivo. A biblioteca *Media Framework* disponibiliza o conjunto de funções base para lidar com áudio e vídeo, enquanto que a *Open Graphics Library* (OpenGL) dá suporte a aplicações gráficas de alto desempenho.

É nesta camada que também se inclui **o ambiente de execução virtual** de aplicações Android™, composto por **dois componentes principais**: as **bibliotecas Java base** e a **Máquina Virtual (VM) Dalvik** ou o **ambiente de execução *Android Runtime***. Em versões anteriores à 5.0, a máquina virtual padrão deste sistema era a Dalvik, enquanto que a partir dessa versão as aplicações passaram a executar num ambiente designado por *Android RunTime* e simplesmente conhecido por **ART**.

As aplicações para Android™ são normalmente implementadas em Java e, para que isso seja possível, é **disponibilizado um conjunto de classes que podem ser prontamente utilizadas**. Por exemplo, **classes dos bibliotecas** `JAVA.*` **ou** `JAVAX.*` contêm software para **manipulação de ficheiros ou definições de estruturas**, enquanto que **a biblioteca** `ANDROID.*` contém classes relacionadas com o **ciclo de vida de aplicações** Android™, nomeadamente **para criação da interface de utilizador** ou *logs*, etc.

Note que as aplicações móveis para esta plataforma **são escritas em Java**, e até são compiladas usando ferramentas Java comuns (`javac`), **mas não correm nas típicas VMs Java**. A Google **desenvolveu a sua própria VM** e ambiente de execução. O formato do código desta máquina é diferente do que corre nas VMs normais. O **processo de implementação e compilação** de uma aplicação Android™ é normalmente decomposto

nos seguintes passos:

1. A aplicação é implementada em Java;
2. O código é **compilado para *bytecode* usando um compilador padrão**;
3. O *bytecode*, ainda que pertencente a várias classes, é **traduzido para o formato `dex` e compilado para um único ficheiro** (`classes.dex`) por uma ferramenta chamada `dx`;
4. O **código, dados e ficheiros** contendo os mais variados **recursos** relativos à aplicação (e.g., ficheiro com imagens, ícones, *layouts*) **são, por fim, empacotados para um arquivo** com extensão `.apk` por uma ferramenta designada por `aapt`.

A aplicação resultante é **carregada e executada numa VM em ambiente totalmente isolado** (*sandboxed*) de outras aplicações. A execução de cada aplicação corresponde à criação de **uma VM com o seu próprio *User ID***. Este modelo de operação garante que, **durante a execução e em condições normais, uma aplicação não deve conseguir aceder aos dados e ficheiros de outra diretamente**.

Note que, apesar de parecer disruptiva, a escolha da Google deve-se novamente ao facto das aplicações móveis executarem num ambiente diferente dos computadores de secretária. A Dalvik ou o ART **foram desenhados para ambientes com potenciais limitações em termos de recursos**, nomeadamente em termos de memória, bateria e processamento.

1.5 Framework Aplicacional
Application Framework

A *framework* aplicacional **contém software ou recursos que as aplicações Android™ podem necessitar e reutilizar**, como por exemplo ficheiros com imagens de botões ou elementos gráficos (componente *View System*). **Alguns dos componentes** incluídos nesta camada podem ser resumidamente descritos da seguinte forma:

- O **gestor de pacotes** (*Package Manager*) mantém **o registo de todas as aplicações instaladas no sistema**. É o funcionamento deste componente que permite que algumas aplicações encontrem outras e **registem serviços que queiram disponibilizar**;
- O **gestor de janelas** (*Windows Manager*) que **lida com as várias janelas e partes mostradas no ecrã** aquando da utilização de uma aplicação ou, por exemplo, **com as sub-janelas** que são despoletadas pela abertura de um menu;
- O **gestor de recursos** (*Resource Manager*) **manipula os recursos de uma aplicação que não foram compilados**, nomeadamente *strings*, os ficheiros de *layout* ou imagens;
- O **gestor de atividades** (*Activity Manager*), que **coordena e suporta a navegação entre atividades** (um ecrã de interação), bem como o seu **ciclo de vida**;
- Os **provedores de conteúdos** (*Content Providers*) são, na sua essência, **bases de dados que as aplicações usam para guardar ou partilhar informação do sistema** (e.g., a base de dados dos contactos num *smartphone*). O componente com o mesmo nome tem a responsabilidade de os gerir. Note que, normalmente, não é possível aceder aos dados de uma aplicação noutra de uma forma direta. **Os *content providers* constituem uma das formas de agilizar** essa troca.
- O **gestor de localização** (*Location manager*), que disponibiliza **informação acerca do movimento e localização** *Global Positioning System* (GPS);
- O **gestor de notificações** (*Notification Manager*), que **é o componente que controla o conteúdo da barra de notificações**. Esta barra é um dos componentes mais importantes do sistema, porque **fornece uma área que está quase sempre visível para o utilizador** e fornece **um meio para informar** utilizadores acerca de **eventos que possam ocorrer fora do âmbito da atividade** que está desenvolver.

1.6 Camada de Aplicação
Application Layer

A camada no topo da pilha é onde se **incluem as aplicações que vêm com o sistema** (de fábrica) e as que são **instaladas subsequentemente**, e que podem incluir **aplicações em desenvolvimento** (*debug*) ou **desenvolvidas por terceiros**. O **utilizador final interage com esta camada diretamente**. Em Android™, nenhuma das aplicações que vêm com o sistema são de uso obrigatório. É sempre possível construir uma aplicação para determinado fim e usá-la em detrimento da que vem instalada com o sistema.

2 Depuração de Aplicações Android
Android Applications Debugging

A **depuração** de aplicações pode ser feita recorrendo tipicamente a uma **panóplia de ferramentas**. Uma dessas ferramentas consiste na **definição de pontos de rutura** no código e na **análise do estado da aplicação após paragem nesse ponto**, depois de executada. O ambiente de desenvolvimento integrado Android Studio tem suporte a este tipo de depuração para aplicações Android™, semelhante ao que já deve conhecer do desenvolvimento de aplicações em Java. Contudo, nesta secção enumeram-se **outros recursos que podem ser usadas para depuração**, mais ligados ao facto das aplicações serem testadas em dispositivos virtuais ou reais,

mas que se podem **monitorizar**. Antes de enumerar esses recursos, apresentam-se algumas vantagens e desvantagens da utilização de dispositivos virtuais.

2.1 Dispositivos Virtuais Android
Android Virtual Devices

Conforme já mencionado antes, o desenvolvimento de aplicações Android™ é dominada pelo uso de emuladores para virtualização de dispositivos móveis. O facto é que, mesmo que se tenha acesso a um ou mais dispositivos reais, **assegurar que uma aplicação funciona para todos ou parte dos dispositivos** disponíveis no mercado irá requerer, quase seguramente, **o uso de virtualização**. Apesar de lentos no arranque e por ventura na execução, por requererem que a máquina virtual processe, simultaneamente, a emulação do dispositivo móvel e o funcionamento do sistema operativo, **os dispositivos virtuais Android™ têm a grande vantagem de poderem ser facilmente monitorizados**. As **vantagens** de usar um emulador Android™ são:

- Em termos **financeiros** – não é necessário comprar um dispositivo móvel real;
- Relativamente à **versatilidade** – o hardware pode ser virtual e facilmente configurado (e.g., o tamanho do cartão de memória SD);
- Em termos de **confinamento** – as alterações que forem feitas, e.g., ao sistema, pela aplicação móvel desenvolvida, são confinadas ao dispositivo.

As **desvantagens** de usar um emulador são:

- Em termos de **desempenho** – a emulação é normalmente mais lenta que o uso de um dispositivo real;
- Quanto às **funcionalidades** – algumas funcionalidades não estão disponíveis em emuladores ou podem ser emuladas de forma não satisfatória em alguns casos (e.g., não há *bluetooth*);
- No que se refere ao **realismo** – mesmo que um emulador esteja esteja próximo de um dispositivo real, no que diz respeito a imitar o seu funcionamento, pode sempre falhar algum detalhe que só notado após se experimentar em ambiente real (e.g., sensores).

A **plataforma Android™ disponibiliza** atualmente um **vasto conjunto de ferramentas e serviços de depuração**. Em baixo referem-se apenas alguns desses recursos.

2.2 Logcat
Logcat

Um dos recursos mais utilizados na depuração de aplicações é a análise de *logs*. Este não deve, contudo, concretizar o principal recurso utilizado, embora aconteça frequentemente. Aquando da depuração de programas escritos em Java, é comum a utilização do procedimento `System.out.println()` para obter valores de variáveis ou estimar o pontos de falha durante a execução da aplicação na consola.

A **plataforma Android™ disponibiliza o seu próprio sistema de *logging***, bem como funcionalidades para colecionar e visualizar informação de depuração. Os *logs* de várias aplicações ou porções do sistema **são colecionadas em pilhas circulares**, que podem depois serem **analisadas ou filtradas através do comando** `logcat`, também fornecido com o *Android Debug Bridge* (ADB). Repare-se que é o facto do sistema de *logging* ser fornecido com a plataforma que permite que **a informação seja produzida e também analisada de uma maneira uniforme** para todas as aplicações. O facto do comando `logcat` estar integrado no `adb` permite que **as mensagens do *log* possam ser lidas em tempo real**, tanto num dispositivo virtual como real (desde que ligado via USB e com o o modo de depuração ativado).

O **comando** que permite aceder ao `logcat` é o seguinte:

```
$ adb logcat
```

Também é possível **executar diretamente o comando `logcat` na *shell*** oferecida pelo `adb`, nomeadamente através do encadeamento das seguintes instruções:

```
$ adb shell
$ logcat
```

A informação devolvida pode ser filtrada ou tratada através de opções do comando.

Uma aplicação pode escrever entradas no `logcat` através da classe `Log` (`import android.util.log;`). Alguns dos métodos que podem ser usados para esse efeito enunciam-se a seguir, realçando-se, de imediato, a sua **interface simplificada e consistente**:

```
v(String, String) // (verbose)
d(String, String) // (debug)
i(String, String) // (information)
w(String, String) // (warning)
e(String, String) // (error)
```

Cada um dos métodos exibidos antes **aceita duas *strings*** e estão **apresentados por ordem de verbosidade, da maior para a menor**. O primeiro parâmetro (*string*) deve ser uma cadeia de caracteres que **identifica a aplicação ou parte do sistema** que está a escrever no *log* (normalmente designada por *tag*), enquanto que **o segundo é a mensagem em si**. É recomendado que a *tag* seja declarada estaticamente no código, para que seja usada de forma uniforme durante toda a aplicação, como se mostra no excerto de código seguinte:

```
private static final String TAG = "pt.ubi.di.pdm.
    example";
...
 @Override
    public void onCreate(Bundle savedInstanceState)
    {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Log.i(TAG,"All fine up to this point!")
```

```
    ...
}
```

Note que **o grau de verbosidade é importante no momento de definir uma entrada** para o `logcat`. Por exemplo, uma entrada do **tipo *verbose*** (`Log.v(...)`) **só deve ser usada na fase de depuração** (desenvolvimento) da aplicação, e deve ser **manualmente** removida antes da compilação da versão final da aplicação. As e**ntradas do tipo *debug*** (`Log.d(...)`) **são compiladas para a aplicação final, mas retiradas durante execução da mesma**, enquanto que **as entradas do tipo *error*, *warning* e *info* são sempre mantidas** (i.e., mesmo aplicações assinadas e distribuídas via *Google Play* podem emitir entradas destas).

É possível obter uma ideia do aspeto do `logcat` em duas das capturas de ecrã incluídas em baixo. O `logcat` emitido em tempo real por um dispositivo virtual a emular um Samsung Galaxy S5 pode ver-se na parte inferior direita das figuras **??** e **??**.

2.3 Visualizador de Hierarquia
Hierarchy Viewer

O SDK Android™ disponibiliza também **uma ferramenta para depuração e otimização da interface de utilizador**. Esta ferramenta, designada por **Visualizador de Hierarquia** (da designação inglesa *Hierarchy Viewer*) e da qual se inclui uma captura de ecrã em baixo (figura **??**), mostra, de uma maneira **muito intuitiva, a forma como os vários elementos da interface de utilizador estão interligados**. De acordo com o que foi dito anteriormente, a perspetiva é obtida através da representação de **uma árvore deitada e com orientação da esquerda para a direita**, sendo que a ramificação é indicativa de como os **elementos estão contidos dentro de outros**. A ferramenta é **completa ao ponto de permitir que se visualize o conteúdo exibido no dispositivo móvel acedendo** interativamente **a cada um dos nós** que compõem a interface gráfica (e.g., se clicar no nó relativo à barra de título de uma atividade, é mostrada essa barra de título com o aspeto e texto definido).

A ferramenta pode **ser acedida** via menus em **ambientes de desenvolvimento integrado** (e.g., no Android Studio poderá ser acedida via Tools → Android → Android Device Monitor, seguido de Window → Open Perspective, e escolher `Hierarchy View`).

2.4 Servidor do Monitor de Depuração Dalvik
Dalvik Debug Monitor Server (DDMS)

O Servidor do Monitor de Depuração Dalvik, conhecido pelo acrónimo da sua designação inglesa ***Dalvik Debug Monitor Server* (DDMS)** é uma aplicação que congrega, numa só interface gráfica e de uma forma mais uniforme, diversas ferramentas e serviços de monitorização de um dispositivo Android™, algumas delas já antes referidas. **Disponibiliza serviços de reencaminhamento de portas, captura de ecrã, informação acerca de *threads*, processos e memória** nos dispositivos ligados ou emulados, **visualização de logs** *logcat*, **simulação de chamadas e mensagens** *Short Message Service* (SMSs) e modificação de informação de localização. Para além de um visualizador para o *logcat* e do visualizador de hierarquia, o DDMS **integra** ainda **uma ferramenta para análise dos registos de execução de métodos**[3], perspetivas sobre a atividade de rede, e um explorador de ficheiros com acesso a todos os ficheiros e diretorias do sistema. O DDMS está atualmente integrado no *Android Device Monitor*.

2.5 Monitor de Dispositivos Android
Android Device Monitor

Versões mais recentes do SDK sugerem utilizar o ***Android Device Monitor* (ADM) para monitorização de dispositivos Android** ou de outras ferramentas que possam funcionar em modo solitário. É possível despoletar esta ferramenta emitindo `$ monitor` na diretoria `tools`. Ilustra-se o seu funcionamento na captura de ecrã incluída na figura **??**. O ADM integra o *Hierarchy Viewer*, o DDMS e muitas outras ferramentas úteis para depuração de aplicações.

2.6 Funcionalidades Avançadas dos Dispositivos Virtuais Android
Advanced Features of the Android Virtual Devices

Apesar de lentos no arranque, os Dispositivos Virtuais Android são **extremamente ricos em termos de funcionalidades**. Por exemplo, é possível **controlar alguns aspetos simulados** no dispositivo através de **uma ligação telnet**. O destino da ligação deve ser **a porta atribuída ao dispositivo no** `localhost`, e.g.:

```
$ telnet localhost 5555
```

Note que é normalmente possível obter a porta onde está o dispositivo à escuta através do comando `$ adb list devices`.

Após obter ligação, podem-se controlar diversos aspetos do dispositivo emulado com **simples instruções** no terminal, e.g.:

- Mudar a carga da bateria no emulador para 50%
  `$ power capacity 50` ;
- Simular uma rede *EDGE*[4]
  `$ network speed edge`;
- Enviar uma mensagem SMS para o dispositivo virtual
  `$ sms send 555555555 "ola mundo"`;

[3] Permite capturar a execução de determinados procedimentos em registos, que podem ser posteriormente analisados.

[4] Enhanced Data rates for GSM Evolution. GSM é o acrónimo de *Global System for Mobile Communications*.

- Iniciar uma chamada com o dispositivo virtual `$ gsm call 555555555`;
- Ajustar as coordenadas GPS para as da fase VI da Universidade da Beira Interior (UBI) `$ geo fix 40.27 -7.50`.

Note que também **é normalmente possível fazer chamadas ou enviar SMSs entre dois dispositivos virtuais** Android™ a correr ao mesmo tempo no mesmo SO. **O número a marcar** num dos dispositivos **corresponde à porta** *Transmission Control Protocol* (TCP) **onde o outro está à escuta**.

Nota: o conteúdo exposto na aula e aqui contido não é (nem deve ser considerado) suficiente para total entendimento do conteúdo programático desta unidade curricular e deve ser complementado com algum empenho e investigação pessoal.

2.7 Capturas de Ecrã
Screenshots

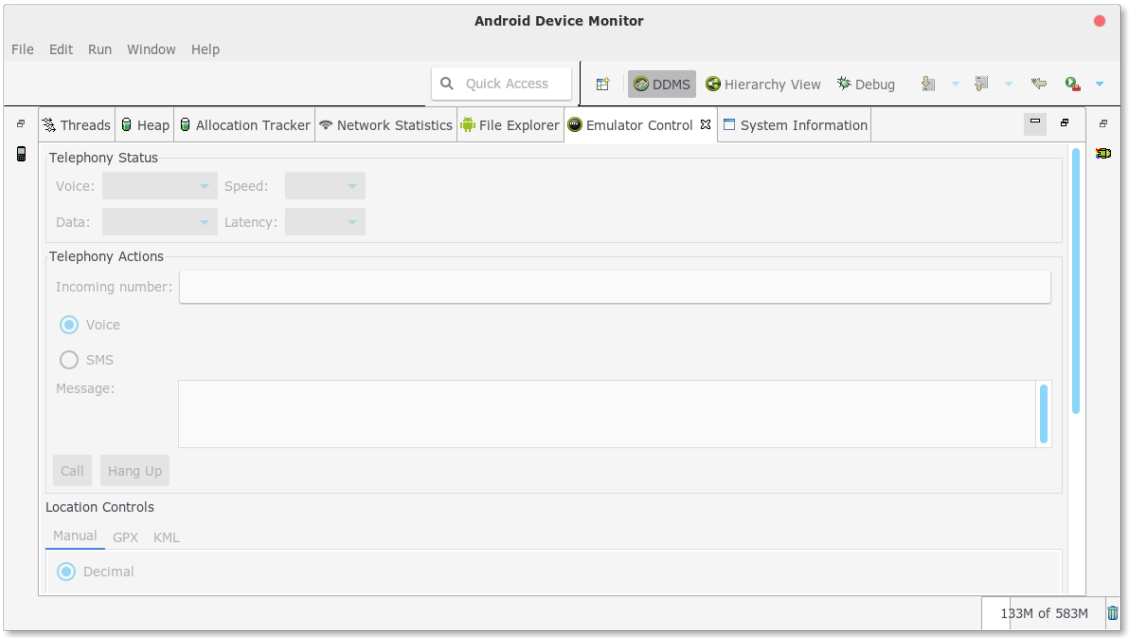


Figura 2: Captura de ecrã da ferramenta de depuração DDMS.

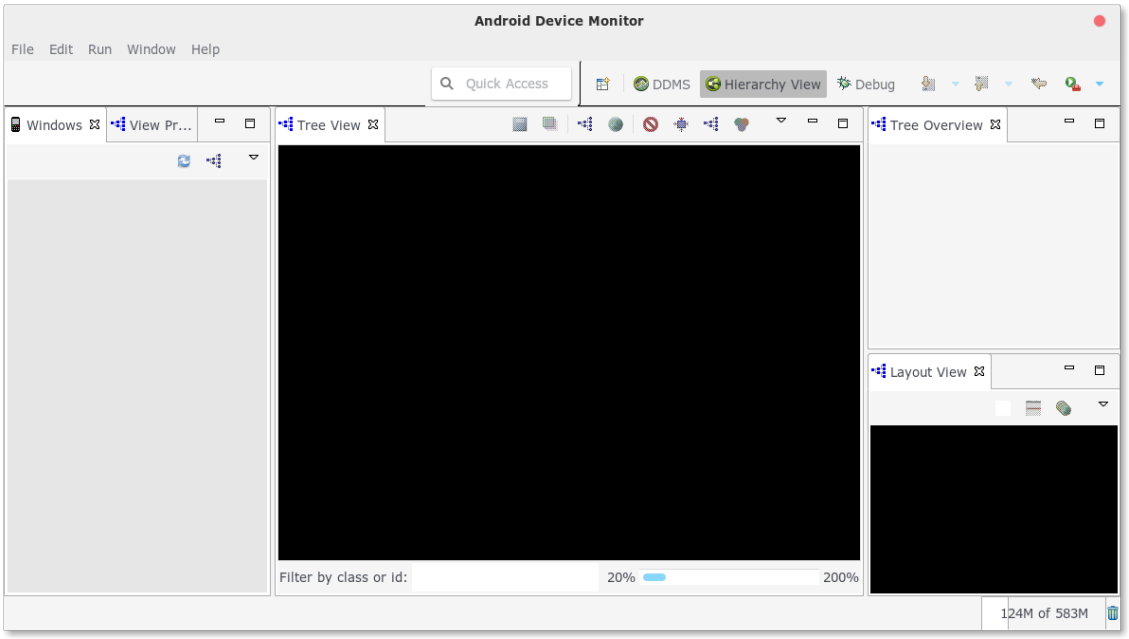


Figura 3: Captura de ecrã da ferramenta de depuração da hierarquia de elementos da interface de utilizador em execução.

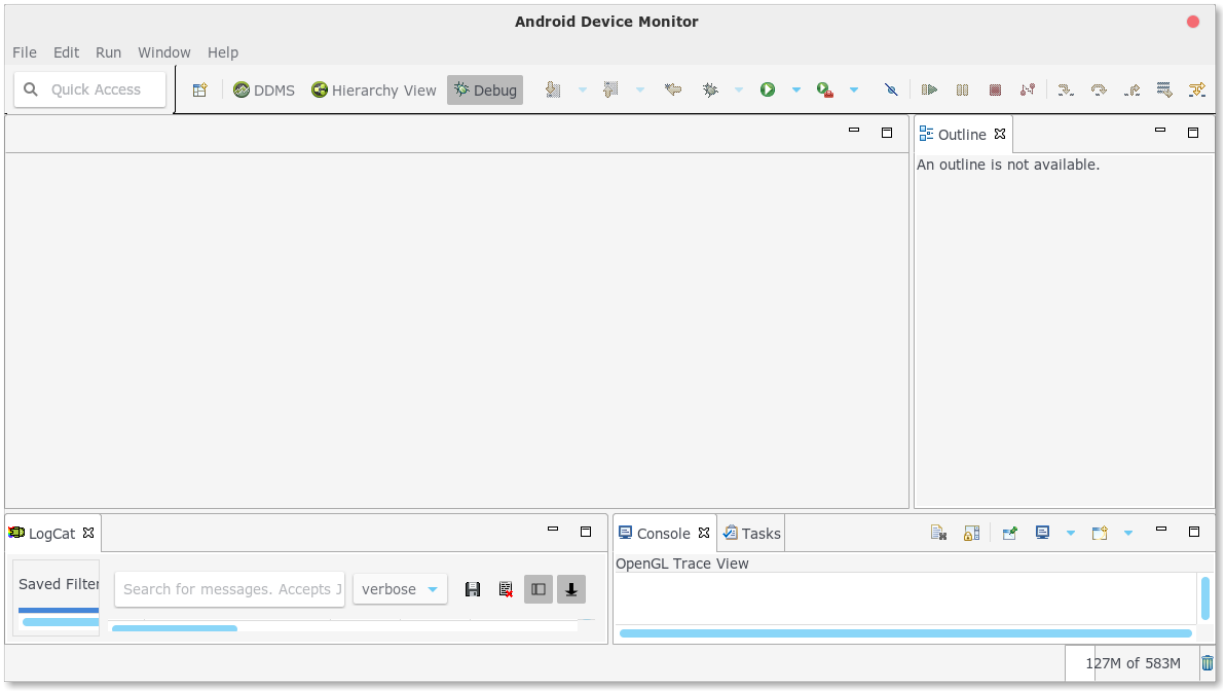


Figura 4: Captura de ecrã da ferramenta de depuração Android Device Monitor.