Technical Features

Hardware Collaboration and Resource Sharing

The key features for hardware collaboration and resource sharing include DSoftBus, Distributed Device Virtualization, Distributed Data Management, and Distributed Scheduler.

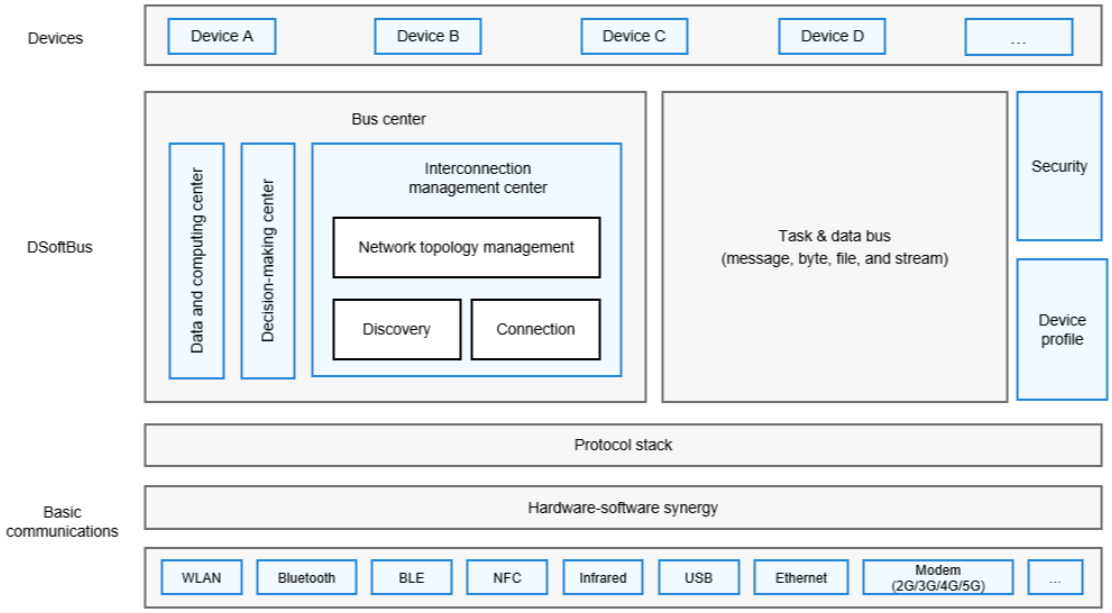
DSoftBus

DSoftBus is a communication base for interconnecting devices, such as mobile phones, tablets, wearables, smart TVs, and head units. It powers devices with distributed communication capabilities, allowing for auto discovery and zero-wait transmission among devices. For you, the application itself is the only thing you need to focus on. Figure 1 shows the diagram of DSoftBus.

Typical scenarios:

- Smart home: While cooking, a user can enable OneHop to connect their mobile phone to an oven, with parameters automatically set. Likewise, users can connect mobile phones to food processors, range hoods, air purifiers, air conditioners, lights, curtains, and more, so as to control and configure related parameters.
- Multi-screen classroom: A teacher can use a smart TV to give lectures and interact with students, and students can use their mobile phones to learn courses and answer questions in class. The unified and fully-connected logical network ensures high bandwidth, low latency, and high reliability of transmission channels.

Figure 1 DSoftBus



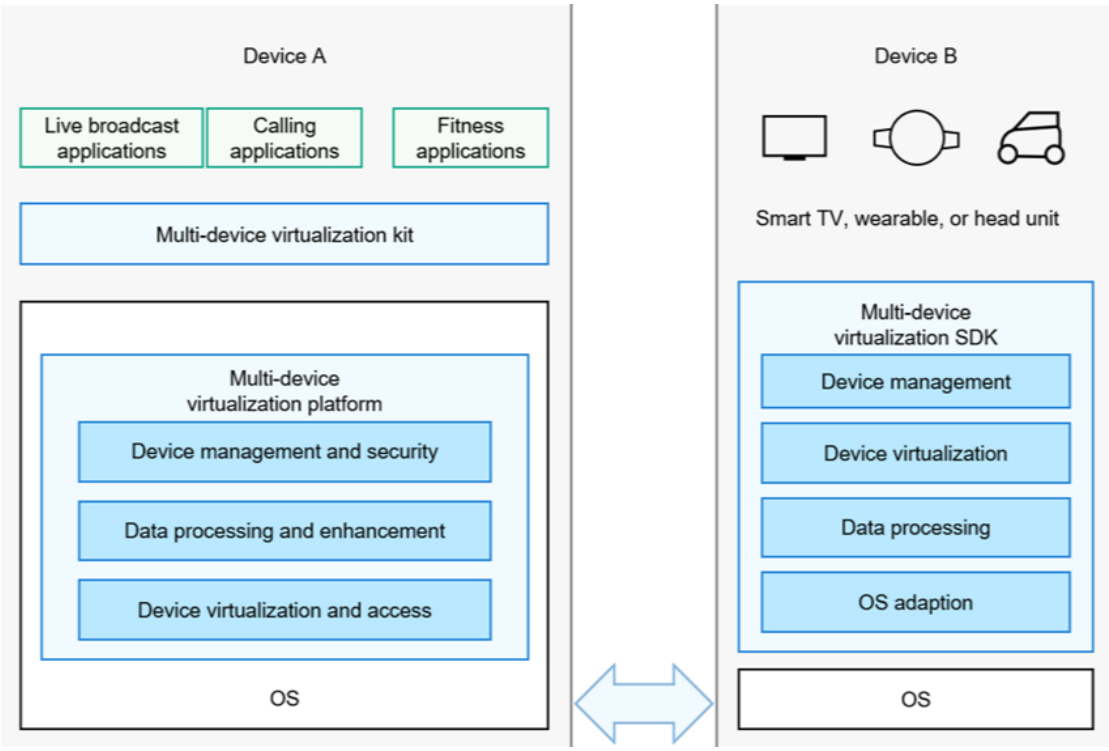
Distributed Device Virtualization

The distributed device virtualization platform enables cross-device resource convergence, device management, and data processing so that multiple devices jointly function as a super virtual device. This platform virtualizes devices and fully utilizes their advantages by assigning the most appropriate hardware to execute particular user tasks. This ensures that services are continuously transferred between different devices. This way, the capability advantages, such as those regarding display, camera, audio, interaction, and sensor, can be fully unleashed for specific devices. Figure 2 shows the diagram of distributed device virtualization.

Typical scenarios:

- Making a video call: While doing housework, users can make a video call over the connection between their mobile phone and smart TV, with the screen, camera, and speaker of the smart TV virtualized as local resources for the mobile phone.
- Playing games: While playing games, users can connect their mobile phones to smart TV, with the gravity sensor, acceleration sensor, and touch control capabilities of the mobile phone virtualized as a remote control, to provide convenient and smooth gaming experience.

Figure 2 Distributed device virtualization



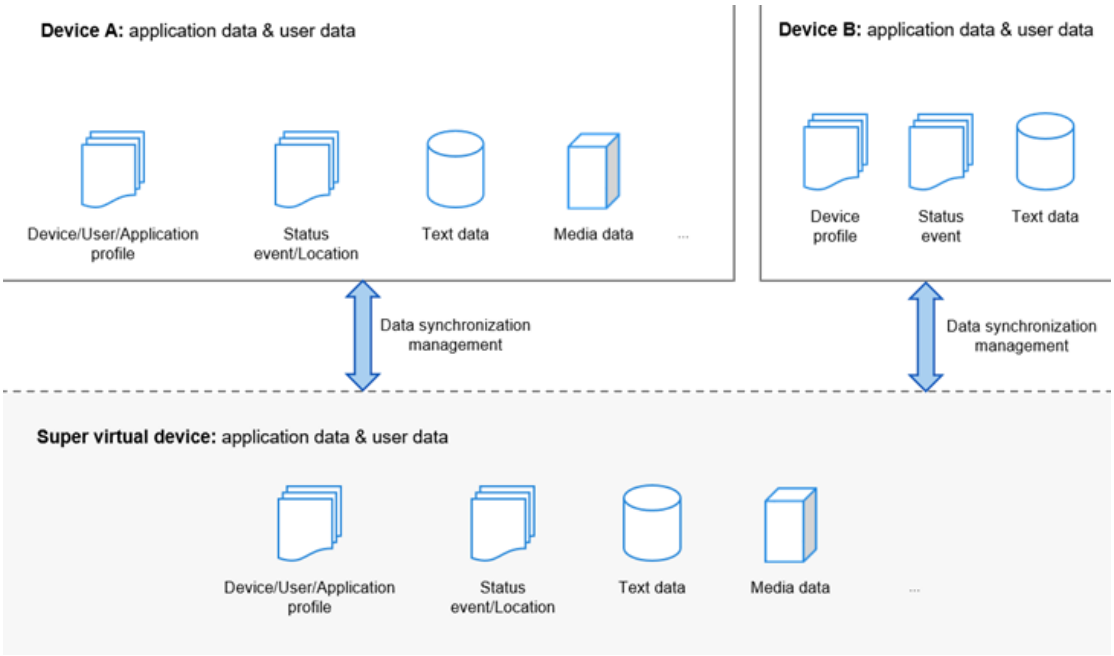
Distributed Data Management

Distributed data management leverages DSoftBus to manage application data and user data distributed on different devices. Under such management, user data is no longer bound to a single physical device, and service logic is separated from data storage. In this case, cross-device data processing is as fast and convenient as local data processing. This facilitates multi-device data storage, sharing, and access in all scenarios, therefore creating a foundation for consistent and smooth user experience. Figure 3 shows the diagram of distributed data management.

Typical scenarios:

- Collaborative office: Users can project a document from their mobile phone to smart TV, and perform operations such as page turning, zooming, and graffiti on the document on the smart TV. They can view the document changes on their mobile phone in real time.
- Family outing: During a family outing, Mom takes a lot of photos on her mobile phone. Via family photo sharing, Dad can browse and save these photos, and also add them as favorites on his mobile phone; grandparents at home can also view these photos on their smart TV.

Figure 3 Distributed data management



Distributed Scheduler

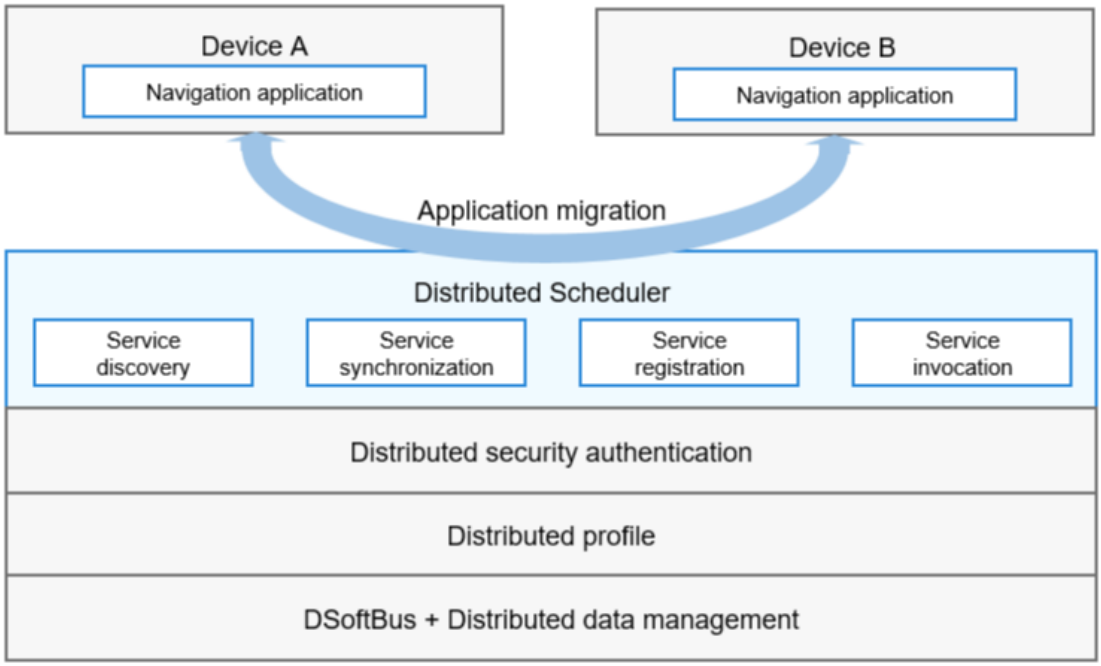
Distributed Scheduler is designed based on technical features such as DSoftBus, distributed data management, and distributed profile. It builds a unified distributed service management mechanism (including service discovery, synchronization, registration, and invocation), and supports remote startup, remote invocation, remote connection, and migration of applications across devices. This way, your applications can select a suitable device to perform distributed tasks based on the capabilities, locations, running status, and resource usage of different devices, as well as user habits and intentions.

Figure 4 takes application migration as an example to illustrate Distributed Scheduler.

Typical scenarios:

- Navigation: If users go outing by driving a car, they can plan a navigation route on their mobile phone before getting on the car. The navigation route will be automatically migrated to the automotive head unit and in-car speaker when users get on the car, and automatically migrated back to the mobile phone when they get off. If users go outing by riding a bicycle, they can pan a navigation route on their mobile phone and then continue checking the navigation information on their watch while riding.
- Takeaway food delivery: After ordering takeaway food via a mobile phone, users can continue checking food delivery information on their watch.

Figure 4 Distributed Scheduler

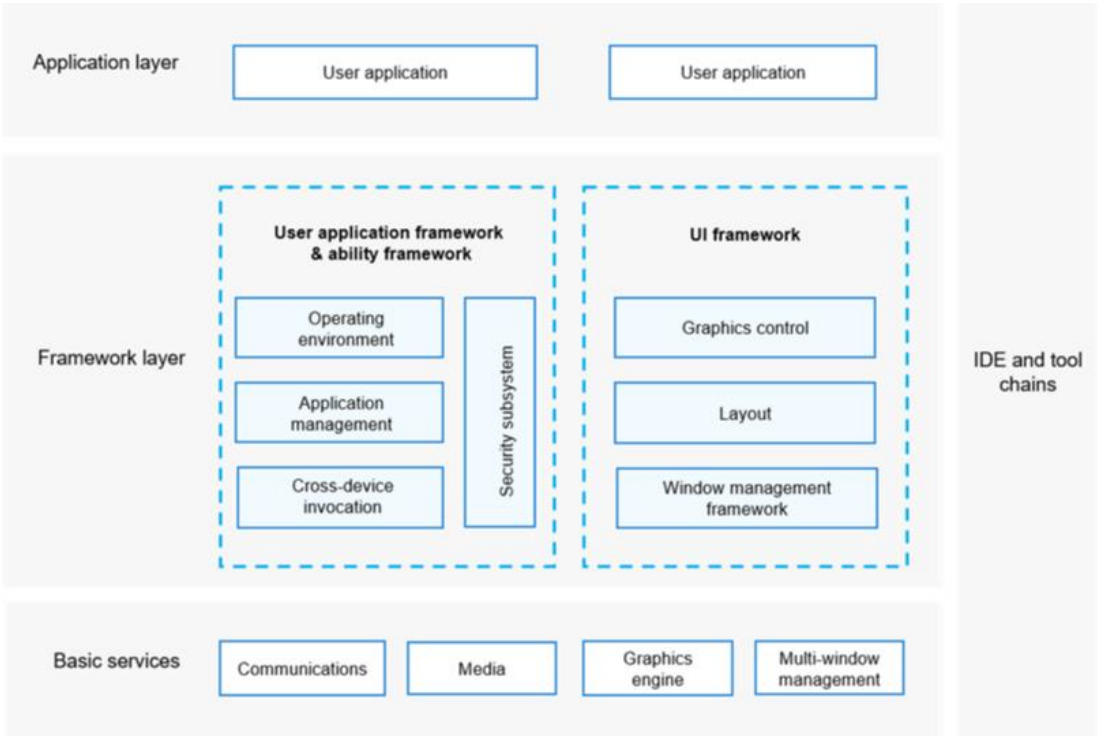


One-Time Development for Multi-Device Deployment

HarmonyOS provides the application, ability, and UI frameworks, which allow you to reuse service and UI logic during application development. This way, you can develop your applications once, and then deploy them across a broad range of devices, improving your development efficiency. Figure 5 shows the diagram of one-time development and multi-device deployment.

To be specific, the UI framework supports Java and JS languages and provides extensive polymorphic components to achieve different UI effects on mobile phones, tablets, wearables, smart TVs, and head units. Also, the UI framework complies with the mainstream design concepts in the industry and provides multiple responsive layout solutions, including grid layouts, to make GUIs adaptive to different screens.

Figure 5 One-time development and multi-device deployment



Unified OS for Flexible Deployment

HarmonyOS leverages component-based and miniaturized-oriented designs to allow on-demand deployment for diversified devices, adapting to different hardware resources and business characteristics. Specifically, component dependencies are automatically generated based on the cross-compilation toolchain to form a tree diagram illustrating component dependencies, facilitating convenient development and making development available for various devices, regardless of their hardware capabilities.

- **On-demand component selection:** You can select required components based on hardware forms and requirements.
- **Mutable function set configuration:** You can tailor function sets for each component based on hardware resources and function requirements. For example, you have an option of configuring only certain components for the UI framework.
- **Associative inter-component dependencies:** You can have inter-component dependencies automatically generated based on the cross-compilation toolchain. For example, if you select components for the UI framework, their associative graphics engine-specific components will be automatically selected.