Operational Semantics of App-Hopping Mechanism on HarmonyOS

Table 1: Helper Functions

Function	Type	Description
rmvApp	$\mathsf{AppInst} \times \mathsf{AppStack} \to \mathsf{AppStack}$	removes an app instance from a given apps stack
addApp	$\mathsf{AppInst} \times \mathsf{AppStack} \to \mathsf{AppStack}$	adds an app instance into a given apps stack
getApp	$\mathsf{App} \times \mathsf{AppStack} \to \mathsf{AppInst} \cup \{\star\}$	returns $\star$ for absence or an app instance with a given name from a given app stack

We specify its operational semantics to help users to better understand the app-hopping behaviors. Figure 1 defines domains, stacks, and operations to describe the operational semantics. We write a for an app name, d for a device name, and γ for a service name. An app instance A is a triple of its activity name, service name, and audio stream status (a, γ, μ). An app stack α is a sequence of app instances. A device instance D is a pair of its device name and its app stack (d, α). A device stack β is a sequence of device instances. A hopping relation r is a triple of source device name, app name, and target device name (d, a, d'), or a dummy symbol ϵ representing no hop exists in the super device.

The operational semantics are defined as the relation of the form $\langle \beta, r\rangle \overset{C}{\longmapsto} \langle \beta', r'\rangle$, which the current devices stack is β and the current hopping relation is r, the operation C resulting in the new devices stack β' and the new hopping relation r'. The behaviors of $\mathsf{StartHop}$ and EndHop operations are defined in Figure 2, in which the helper functions are defined in Table 1.

We take the rule (1) as an example to explain the semantics in Figure 2,

Rule (1) and (4) specify the semantics of $\mathsf{StartHop}$, rule (2) and (3) specify the semantics of EndHop. Moreover

- The rule (1) specifies that if a user hops an app when there exists no hops in the super device, then the hopping relation (d_s, a, d_t) will change to ϵ and the app stack of the device d_t (resp. d_s) will remove (resp. add) the app instance of app a accroding to the *rmvApp*() (resp. *addApp*()) function. if a user hops an app when there is no hops in the super device, the semantics of it are dual of the second.

$$
\begin{array}{lcllll}
a & \in & \mathsf{App} & & \textit{application name} \\
d & \in & \mathsf{Device} & & \textit{device name} \\
\gamma & \in & \mathsf{Service} & & \textit{service name} \\
\mu & \in & \mathsf{AudioStatus} & = & \{\mathsf{PLAY}, \mathsf{PAUSE}, \mathsf{STOP}, \mathsf{DUCK}, \mathsf{PAUSE}^{*}\} \\
r & \in & \mathsf{HopRelation} & = & \mathsf{Device} \times \mathsf{App} \times \mathsf{Device} \cup \{\epsilon\} \\
A & \in & \mathsf{AppInst} & = & \mathsf{App} \times \mathsf{Service} \times \mathsf{AudioStatus} \\
D & \in & \mathsf{DeviceInst} & = & \mathsf{Device} \times \mathsf{AppStack} \\
\alpha & ::= & \epsilon \mid A :: \alpha & \in & \mathsf{AppStack} \\
\beta & ::= & \epsilon \mid D :: \beta & \in & \mathsf{DeviceStack} \\
C & ::= & \mathsf{EndHop} & \mid & d.\mathsf{StartHop}(a, d)
\end{array}
$$

Figure 1: Domains, Stacks, and Operations

$$\frac{r = (d_s, a, d_t) \quad \beta = \beta_1 :: (d_s, \alpha_s) :: \beta_2 :: (d_t, \alpha_t) :: \beta_3 \quad A = getApp(a, \alpha_s) \quad \alpha'_s = rmvApp(A, \alpha_s) \quad \alpha'_t = addApp(A, \alpha_t)}{\langle \beta, \epsilon \rangle \overset{d_s.\mathsf{StartHop}(a,d_t)}{\longmapsto} \langle \beta_1 :: (d_s, \alpha'_s) :: \beta_2 :: (d_t, \alpha'_t) :: \beta_3, r \rangle} \tag{1}$$

$$\frac{r = (d_s, a, d_t) \quad \beta = \beta_1 :: (d_s, \alpha_s) :: \beta_2 :: (d_t, \alpha_t) :: \beta_3 \quad A = getApp(a, \alpha_t) \quad \alpha'_s = addApp(A, \alpha_s) \quad \alpha'_t = rmvApp(A, \alpha_t)}{\langle \beta, r \rangle \overset{\mathsf{EndHop}}{\longmapsto} \langle \beta_1 :: (d_s, \alpha'_s) :: \beta_2 :: (d_t, \alpha'_t) :: \beta_3, \epsilon \rangle} \tag{2}$$

$$\langle \beta, \epsilon \rangle \overset{\mathsf{EndHop}}{\longmapsto} \langle \beta, \epsilon \rangle \tag{3}$$

$$\frac{\begin{array}{c} r = (d_s, a, d_t) \quad r' = (d'_s, a', d'_t) \quad \langle \beta, r \rangle \overset{\mathsf{EndHop}}{\longmapsto} \langle \beta', \epsilon \rangle \quad \beta' = \beta_1 :: (d'_s, \alpha'_s) :: \beta_2 :: (d'_t, \alpha'_t) :: \beta_3 \\ A' = getApp(a', \alpha'_s) \quad \alpha''_s = rmvApp(A', \alpha'_s) \quad \alpha''_t = addApp(A', \alpha'_t) \end{array}}{\langle \beta, r \rangle \overset{d'_s.\mathsf{StartHop}(a',d'_t)}{\longmapsto} \langle \beta_1 :: (d'_s, \alpha''_s) :: \beta_2 :: (d'_t, \alpha''_t) :: \beta_3, r' \rangle} \tag{4}$$

Figure 2: Operational Semantics of App-Hopping Mechanism

- The second specifies that if a user ends the app hopping when there exists a hop in the super device, then the hopping relation (d_s, a, d_t) will change to ϵ and the app stack of the device d_t (resp. d_s) will remove (resp. add) the app instance of app a accroding to the *rmvApp*() (resp. *addApp*()) function.
- The rule (3) specifies that if a user ends the app hopping when there is no hops in the super device, the devices stack and the hopping relation remain the same without any changes.
- The last specifies that if a user hops an app when there exists another hop in the super device, then the hop will be ended first, and a new hop will be started as the third semantic.

Now, we will show the details of the definition of the two functions *rmvApp*() and *addApp*(). We let $\alpha = A_1 :: \cdots :: A_n$ be an app stack and for each $i \in [1, n]$, $A_i = (a_i, \gamma_i, \mu_i)$, then

- $rmvApp(A_1, \alpha) = A_2 :: \cdots :: A_n$ if $\mu_1 \neq \mathsf{PLAY}$ or $\forall i \in [2, n], \mu_i \notin \{\mathsf{DUCK}, \mathsf{PAUSE}^*\}$.
- $rmvApp(A_1, \alpha) = A_2 :: \cdots :: A'_i :: \cdots :: A_n$ such that $A'_i = (a_i, \gamma_i, \mathsf{PLAY})$, if $\mu_1 = \mathsf{PLAY}$ and $\mu_i \in \{\mathsf{DUCK}, \mathsf{PAUSE}^*\}$ for some $i \in [2, n]$.

Intuitively, the function $rmvApp(A, \alpha)$ indicates the behaviors of removing an app instance A from a given app stack α.

- If app instance A is *not* in the status PLAY, or each app instance in α is not in the status DUCK or PAUSE*, $rmvApp(A, \alpha)$ will remove A from α without modifying other app instances in α.
- Otherwise, i.e., if app instance A is in the status PLAY, and there exists another app instance A' in the status DUCK or PAUSE*, since A is removed from α, the AC between A and A' disappears, then A' will turn into the status PLAY.

Similarly, the function $addApp(A, \alpha)$ indicates the behaviors of adding an app instance A from a given app stack α.

- If A is in the status STOP or PAUSE, $rmvApp(A, \alpha)$ will add A into α without modification of other app instances in α, since there is no ACs in α,
- Otherwise, i.e., if A is in the status of $\{\mathsf{PLAY}, \mathsf{DUCK}, \mathsf{PAUSE}^*\}$, when A attempts to be added into α, the AC may occur, the status of some app instance using audio-stream in α will change according to the resolution to solve the conflicts.