Le *distributed Service Manager*

7.1 Motivations

Nous avons défini un modèle générique basé sur une nouvelle approche du concept de service (cf. chapitre 6). De nombreuses propriétés et concepts ont été dégagés de cette étude, nous permettant de dessiner les mécanismes clés d'un telle solution globale capable d'assurer la continuité dans un environnement hétérogène. Cependant nous n'avons pour l'instant proposé qu'une approche conceptuelle, de haut niveau, qui adresse les problématiques rencontrées jusqu'à présent de manière théorique.

Ce chapitre sera consacré à la définition concrète d'une solution directement issue de ce modèle, en précisant les mécanismes nécessaires aux différentes phases identifiées. Nous proposerons une architecture et détaillerons les fonctions de mobilité correspondantes tout en restant focalisé sur une contrainte : une expérience utilisateur optimale.

7.2 Principe

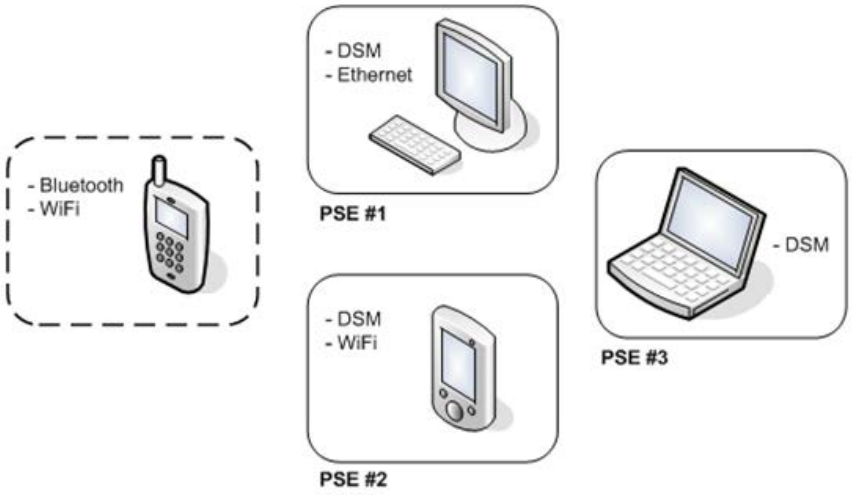
L'implémentation du modèle que nous proposons est appelée Gestionnaire de Service Distribué (noté dSM pour *distributed Service Manager*). Comme son nom l'indique, et logiquement par rapport à l'absence d'infrastructure dans le PSE, nous adoptons une approche distribuée. La logique de mobilité décrite de manière abstraite dans le chapitre précédent et qui existe au sein d'un PSE est implémentée

de manière collaborative, distribuée, par chaque interface qui le composent. Ces travaux ont été publiés dans [88].

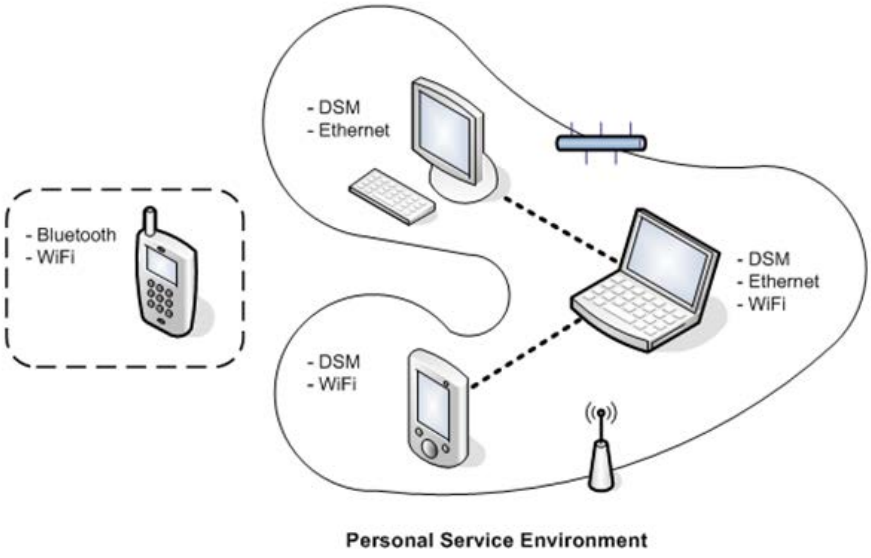
Le dSM est donc une application distribuée présente dans tous les terminaux du PSE qui collaborent entre eux pour réaliser les mécanismes de mobilité. Chaque instance est en charge du contrôle des applications et des ressources locales, elle apparaît donc comme une fonction supplémentaire au système d'exploitation de l'hôte. Un réseau pair-à-pair est ainsi établi grâce au dSM, incluant l'ensemble des interfaces du PSE. Ce réseau est géré par un protocole réseau ad-hoc capable d'assurer les fonctions de base : ajout et suppression de nouveaux pairs (terminaux de l'utilisateur), identification et authentification (pour protéger l'utilisateur et ses services), routage des messages de signalisation entre les interfaces (le PSE pouvant couvrir plusieurs réseaux physiques IP), etc. Les Figures 7.1 illustrent comment les environnements personnels de services d'un utilisateur sont gérés en fonction de la connectivité des dSM embarqués dans chaque interface : la Figure 7.1(a) montre plusieurs dSM déconnectés qui forment autant de PSE puis la Figure 7.1(b) montre les même terminaux dont la connectivité a été établie via une technologie d'accès commune (hôte de droite) et forme alors un unique PSE. On peut noter également que l'unique terminal dépourvu de dSM est exclu de l'environnement et ne pourra bénéficier des mécanismes de mobilité.

Afin de garantir la généricité, une partie des mécanismes est assurée par les applications, imposant une contrainte sur celles-ci qui doivent être compatible avec le dSM pour que les services qu'elles délivrent soient transférables. Cette approche est inévitable et ce pour deux raisons. Premièrement nous devons gérer un environnement hétérogène, il est donc difficile de connaître a priori le comportement et les propriétés de chacun de ses composants. Il est plus évident d'adresser les problématiques de l'intérieur, l'application par exemple connaît la logique de ses processus internes, il est donc judicieux de lui conférer une partie du mécanisme de mobilité. Deuxièmement, nous apportons une nouvelle fonction de mobilité à chaque terminal, constituant une sorte d'amélioration du système existant, exploitable par les applications qui le souhaitent ; la contrainte majeure étant de garantir une transparence totale pour les anciennes applications non compatibles (qui seront simplement ignorées du PSE).

L'impact sur ces applications compatibles est cependant minimal, nous détaillerons ces contraintes dans la section 7.2.2 tandis que le chapitre suivant 8 apportera



(a) Terminaux déconnectés formant plusieurs PSE.



(b) Terminaux connectés formant un PSE unique.

FIGURE 7.1 – Gestion des environnements personnels de service.

des données précises sur cet impact grâce à un prototype. Mais tout d'abord, un exemple simple du dSM en action.

7.2.1 Exemple

Afin d'illustrer l'utilisation du dSM dans un cas concret et avant d'entrer dans les détails techniques, nous allons présenter un exemple simple qui reprend le cas du service d'édition de texte.

Bob est dans le bus. Il profite du trajet pour rédiger un courrier sur son PDA (*Personal Digital Assitant*) grâce à une application très basique qui ne gère que du texte brut. Bob arrive enfin à son bureau, il met le PDA dans sa poche et allume son ordinateur. Une fois démarré, le PC active le dSM qui détecte aussitôt le PDA également doté d'un dSM. Via la connectivité Wi-Fi de l'environnement de travail, Bob vient d'établir un *pse* constitué de deux interfaces : le PC et le PDA. Dès lors les applications compatibles exposent leur service au PSE conférant à Bob une liberté totale de mouvement.

Bob est notifié sur son ordinateur qu'un service d'édition de texte appelé « *lettre au boss* » a été découvert dans le PSE. Sachant pertinemment qu'il s'agit du courrier qu'il a commencé à rédiger plus tôt dans le bus, il double-clique directement sur la notification et MS Word se lance, et une seconde plus tard la lettre en cours de rédaction s'affiche à l'écran. Il peut alors faire quelques ajustements, impossibles depuis le PDA : mise en page, correction orthographique, et enfin imprimer le document prêt à poster !

7.2.2 Les applications

Les applications doivent donc être compatibles avec le dSM pour que les services fournis soient considérés par le PSE, et donc que les mécanismes de continuité soient applicables. En effet, le dSM doit être capable de manipuler les applications et les ressources durant tout le processus de mobilité : notifier de l'état des services aux interfaces distantes, mettre en pause un service et récupérer le contexte, le transférer, lancer ou arrêter une application, reprendre l'exécution d'un service, etc. La plupart de ces fonctions peuvent être réalisées sans collaboration spécifique (démarrer ou arrêter un programme par exemple), cependant les actions portant

sur les propriétés du service et les ressources, particulièrement proches de l'implémentation requièrent la participation de l'application qui est la mieux placée pour apporter des informations correctes.

Des interfaces sont donc définies afin d'encadrer les échanges entre le dSM et l'application, précisant de manière indépendante au type de service, la nature des fonctions requises dans une application compatible. Parmi ces fonctions, deux sont essentielles : la fonctions *pause* et *resume*.

Fonction *pause*.

La fonction *pause* correspond en partie au mécanisme de *capture* défini dans le modèle générique (cf. 6.3.3).

Lorsqu'un transfert est déclenché, le dSM du terminal d'origine appelle la fonction pause de l'application qui instancie le service en question. Celle-ci interrompt alors la fourniture du service dès que possible et fournit un contexte stable au dSM (que ce dernier transfèrera au dSM du terminal destination).

L'application est uniquement interrompue (en pause), l'utilisateur ne peut plus bénéficier du service (ou de manière très limitée), cependant l'application n'est pas nécessairement arrêtée car le contexte doit rester intact et disponible en lecture pour le dSM. La mort des processus de l'application pourrait entraîner la fermeture de flux de données d'où une perte de ressource dommageable. De plus, si le transfert échoue pour une raison quelconque, l'application doit reprendre son cours sur le terminal origine avec le contexte préservé. Pendant la pause et jusqu'à la reprise du service à distance (signant le succès du transfert), le contexte ne peut pas être modifié, ce qui entraînerait le cas échéant une incohérence entre les états du service sur les terminaux origine et destination

Il est à noter que l'interruption causée par la pause de l'application est généralement très courte (directement liée au contexte) et masquée par le déplacement physique de l'utilisateur entre les deux interfaces désignées pour le transfert.

Fonction *resume*.

La fonction *resume* assure en partie le mécanisme de *reprise* défini dans le modèle générique (cf. 6.3.5).

La *pause* de l'application du terminal origine correspond au *resume* du terminal destination. Une fois le transfert du contexte effectué d'une interface à l'autre, l'application censée instancier le service en mobilité sur le terminal destination est automatiquement lancée. Elle est exécutée dans un mode spécial (reprise) qui prend en paramètre le contexte. L'application doit alors exploiter au maximum les ressources fournies, dans la limite de ses capacités (et de celles du terminal hôte), afin de restituer à l'utilisateur les références contextuelles du service adaptées à son nouvel environnement.

On constate ici que le dSM conserve une correspondance entre les types de services et les applications locales. Cette association peut être effectuée lors de l'installation de l'application ou lors de chaque utilisation, via le système d'exploitation ou directement auprès du dSM.

Autres fonctions.

Outre les mécanismes *pause* et *resume* qui sont les deux principales fonctions que doit implémenter les applications compatibles et qui réalisent en partie les phases de mobilité définies dans le modèle générique, d'autres fonctions sont nécessaires permettant notamment au dSM et à l'application d'échanger des messages de signalisation (orchestration du transfert, collecte d'informations, etc).

Durant la phase de découverte par exemple, de nombreuses informations concernant les services existants (nom, type, version, identifiant, etc) sont transférées entre les terminaux afin de notifier l'utilisateur et d'initier éventuellement des déclenchements. On reviendra également dans la section suivante sur certaines fonctions nécessaires à la gestion des ressources au niveau applicatif.

7.2.3 Gestion des ressources

La gestion des ressources est la problématique centrale de la mobilité de service tel qu'identifié dans le chapitre précédent 6. La continuité nécessitant le transfert du contexte, un mécanisme spécifique a été imaginé afin de régler les questions d'optimisation et d'adaptation. Nous avons mentionné à plusieurs reprises, par souci de simplification, que le contexte transitait d'un terminal à un autre. La réalité est plus nuancée. Lors d'un transfert, le contexte est d'abord figé dans un état stable, empêchant toute modification ultérieure des ressources (fonction *pause*

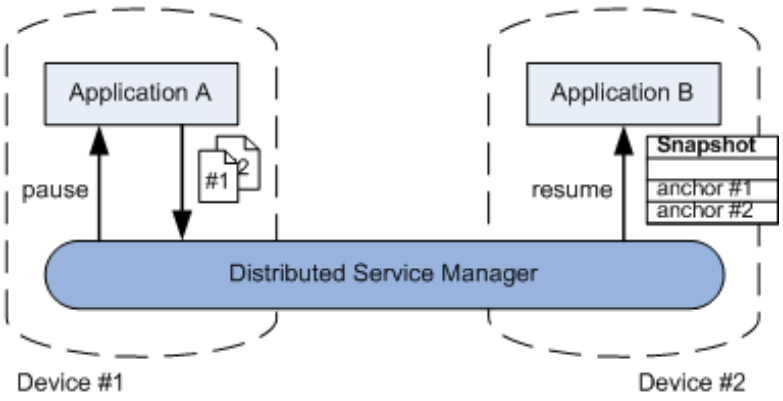
7.2.2). Chaque ressource est alors associée à un identifiant, un pointeur (ou *anchor*) unique pour une instance dSM donnée. Un cliché (*snapshot*) du contexte comportant uniquement la liste des pointeurs est alors réalisé et c'est ce *snapshot* qui est envoyé au dSM du terminal destination. D'où la nécessité également de conserver, jusqu'à la fin complète du transfert, les ressources intactes sur le terminal origine.

L'application destination est alors exécutée en mode *resume* avec comme paramètre le *snapshot* contenant les pointeurs de chaque ressource du contexte (cf. 7.2.2). En fonction de ses capacités, l'application demande au dSM local les ressources prioritaires qu'elle désire. Les dSM vont alors collaborer pour transmettre, à la demande, toutes les ressources requises et correspondant aux besoins du terminal destination. Enfin, les ressources qui n'auront pas été transférées car non supportées ou optionnelles le seront de manière automatique et asynchrone en « tâche de fond », de façon totalement transparente, afin de libérer progressivement le terminal origine. La Figure 7.2 illustre le transfert de ressources via les *anchors* entre deux terminaux lors des fonctions *pause* 7.2(a) et *resume* 7.2(b).

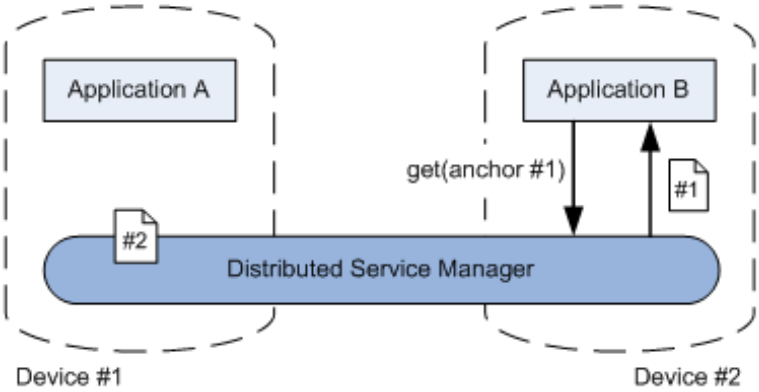
Cette gestion originale des ressources offre de nombreuses possibilités : adaptation fine du contexte à l'application, optimisation du transfert, transitivité des ressources permettant une adaptation positive, etc. Ainsi les pointeurs (*anchors*) permettent de masquer l'hétérogénéité des ressources, les flux de données par exemple étant des ressources non transférables seront simplement redirigées (*tunneling*) du dSM origine au dSM destination de manière transparente pour l'application, évitant ainsi toute coupure du flux.

7.2.4 Interface graphique

À de nombreuses reprises nous avons fait référence à l'interaction entre l'utilisateur et le dSM et plus particulièrement lors de la phase de déclenchement. Nous ne rentrerons pas dans les détails de la conception d'une telle interface qui sort du cadre de ces travaux, cependant de nombreuses problématiques émergent de cette question. En effet, nous nous sommes volontairement positionnés dans une approche de déclenchement manuel où l'utilisateur désigne explicitement les services et les terminaux concernés par un transfert. L'interface du dSM doit notifier l'utilisateur de l'existence des éléments distants tels que les terminaux et les services



(a) Fonction *pause*.



(b) Fonction *resume*.

FIGURE 7.2 – Principe de transfert du contexte.

tout en lui permettant d'agir directement sur son PSE en fonction de ses besoins de mobilité.

Une approche automatique qui limiterait l'importance de l'interface graphique est aussi réalisable, aucun déclenchement de ce type n'est intégré nativement au système, néanmoins le DSM prévoit un certain nombre d'interfaces (de programmation) permettant à un tiers d'implémenter un mécanisme de gestion automatique de la mobilité et d'agir sur le PSE.

7.3 Cas des services connectés

Certains services tels que ceux de télécommunications ou de streaming multimédia que nous avons étudiés au début de cette partie présentent une contrainte commune qui est une dépendance forte aux sessions de données. Si le flux correspondant est interrompu, le service n'est plus délivré. La nouvelle approche du concept de service et le modèle générique de mobilité qui en a découlé s'appliquent indifféremment à ces services connectés. Cependant, l'application des fonctions de continuité dans ce cas précis mérite quelques explications.

Dans le cas des services connectés, le contexte contient au moins une ressource particulière qui est la session, donc une donnée de taille illimitée, intransférable. Pour traiter ce type de ressource, deux possibilités de mobilité se présentent : soit le flux est substitué (cas des solutions étudiées jusqu'alors), soit le flux est redirigé. Nous avons déjà défini dans l'état de l'art ce qu'est une session (cf. 1.1.2), nous avions identifié qu'elle était composée de deux couches : contrôle et données. Les deux solutions de mobilité, substitution et redirection, correspondent respectivement aux approches de niveau contrôle et données.

Nous allons décrire les approches contrôle et données dans les sections suivantes. Il est important de noter qu'il n'est pas question ici du mécanisme de mobilité de service qui reste inchangé, nous ne faisons qu'expliciter la méthode de transfert des ressources de type « flux » qui est caractéristique des services connectés. Ces travaux ont été publiés dans [89].

7.3.1 Approche contrôle

L'approche « contrôle » est la plus classique, elle est d'ailleurs employée dans l'ensemble des mécanismes de mobilité de session, cf. 2.3.2. Dans le cadre du PSE, lors de la demande de la ressource par le terminal destination, les propriétés de la session sont envoyées par le DSM origine au terminal destination, ce dernier va alors mettre en œuvre des mécanismes spécifiques au protocole afin d'exécuter le transfert du flux de données ; on peut imaginer une application SIP qui utilise le mécanisme REFER tel que nous l'avons vu au chapitre 3. La Figure 7.3 illustre cette approche.

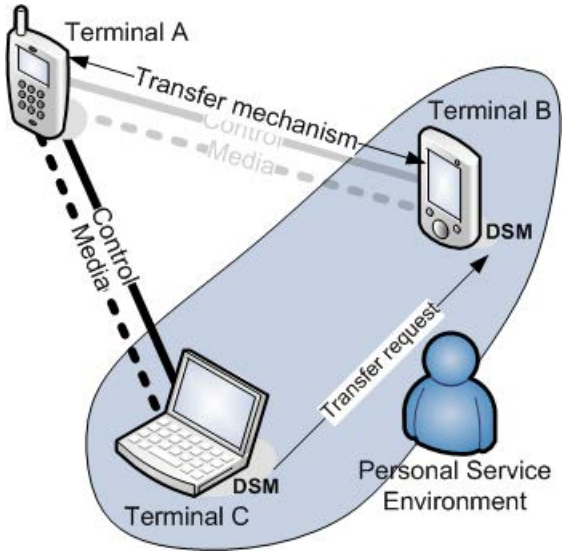


FIGURE 7.3 – Approche orientée « contrôle » de transfert des flux.

Cette solution est purement protocolaire et dépend directement de l'existence d'un mécanisme au niveau de la couche contrôle de la session. Le problème de la dépendance entre le mécanisme de mobilité et le service se pose sérieusement. En effet, le modèle générique que nous avons introduit pose comme principe la séparation des fonctions de mobilité de l'implémentation des services. Une telle solution n'est pas compatible avec l'hétérogénéité de l'environnement, son application sera restreinte à des implémentations, des protocoles spécifiques, limitant les transferts aux interfaces similaires.

Enfin, cette solution n'est applicable que si le terminal destination (A) naturellement connecté au terminal origine (B) peut également se connecter au terminal

correspondant (C), l'autre extrémité de la session. En effet, si les connectivités A-B et B-C sont évidentes, la connexion A-C n'est pas garantie or c'est là que la nouvelle session doit prendre place.

7.3.2 Approche données

L'approche « données » applique simplement le mécanisme de pointeurs présenté précédemment (cf. 7.2.3). Le flux de données n'est finalement qu'une ressource parmi les autres qui peut être désignée par un identifiant (*anchor*). Lorsque l'application destination souhaite obtenir cette ressource, elle fait une requête au dSM sous-jacent en indiquant ce pointeur. Le dSM origine redirige alors le flux reçu vers le dSM destination qui transmet les données directement et de manière transparente à l'application. La figure 7.4 illustre cette solution.

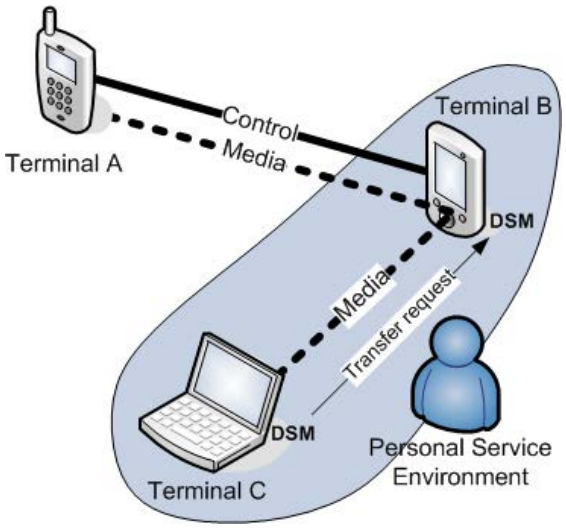


FIGURE 7.4 – Approche orientée « données » de transfert des flux.

Cette approche pose cependant un problème car la couche de contrôle qui est dépendante de l'implémentation est intrinsèquement perdue. Le flux pourra éventuellement persister car contrôlé par le terminal origine mais l'application destination ne pourra en aucun cas le manipuler. En conséquence, le flux ne pourra pas non plus être renégocié d'où l'incapacité d'adapter ses propriétés à la destination tel que l'encodage utilisé.

7.3.3 Approche hybride

Les deux approches contrôle et données n'étant pas satisfaisantes individuellement, elles se révèlent cependant complémentaires. À y regarder de plus près, l'ensemble des informations de session requises aux deux approches sont présentes dans le contexte du service : les propriétés de la session, son état au moment du *snapshot*, et le flux lui même. Les mécanismes de substitution de session sont eux implémentés par les applications.

En d'autres termes, le PSE est non seulement parfaitement compatible avec les deux approches mais elles se complètent : si l'application de destination supporte les ressources relatives à la couche contrôle, alors elle les utilisera pour réaliser le transfert directement, sinon le flux sera redirigé via la méthode générique classique.

7.4 Conclusion

Le *distributed Service Manager* est une tentative d'implémentation du modèle générique de mobilité défini dans le chapitre précédent. Comme prévu, cette entreprise a permis d'identifier de nombreuses problématiques qui ont suscité des mécanismes spécifiques. La généricité a été conservée, garante d'une continuité dans des environnements hétérogènes ; des fonctions telles que la gestion des ressources, les *snapshots* ou encore les *anchors* confèrent une grande souplesse à la solution proposée.

De nombreuses questions restent cependant en suspens : quel modèle de représentation des ressources, comment les organiser au sein du contexte (notion de groupe), faut-il normaliser les *snapshots* (représentation du contexte), comment optimiser le *tunneling* lors de la redirection d'un flux, comment réaliser la mobilité partielle des services (cf. 2.1.4) ?

Mais la question la plus immédiate est certainement celle de la faisabilité d'une telle solution. L'étape suivante dans ces travaux de recherche est le développement d'un prototype du DSM afin de prouver la possible réalisation d'un tel modèle et d'évaluer son impact sur l'utilisateur et son environnement.