Vers un modèle générique de mobilité

6.1 Motivations

La continuité des services, que ce soit des points de vue temporel et contextuel ne peut aujourd'hui être assurée de manière générique dans un environnement hétérogène. Aucune des différentes approches présentées dans l'état de l'art de ce mémoire (cf. 2.3) n'ont permis de dégager de solution qui adresse cette problématique globale. L'étude de plusieurs cas, dans l'IMS puis dans le Web, avec des services de types distincts met en évidence l'absence de *modèle générique* qui traite le *Service* en tant que tel, indépendamment de son implémentation. L'absence également d'un *modèle complet* assurant la gestion de la mobilité jusqu'au bout, c'est à dire jusqu'à l'utilisateur.

Les différentes approches que nous avons étudiées ou même implémentées jusqu'à présent sont intéressantes à plusieurs titres : elles proposent des mécanismes efficaces dont nous chercherons à nous inspirer mais aussi elles mettent en évidence, dans leur réalisation, les fonctions qui manquent à une gestion complète de la continuité. S'il est un dénominateur commun entre la quasi-totalité de ces solutions, c'est certainement une mobilité appliquée à un très bas niveau, le *Service* étant délaissé au détriment de son implémentation : l'application, le protocole, la session. Le nombre important de mécanismes de mobilité purement réseau comparé aux quelques approches applicatives présentes dans la littérature est révélateur.

Certes, des mécanismes assurent aujourd'hui la mobilité de données spécifiques dans des conditions particulières mais le monde réel dans lequel vit l'utilisateur est hétérogène, composé de terminaux, de réseaux, d'applications différents, chacun possédant des caractéristiques propres qu'il est nécessaire de considérer et d'exploiter.

Ce double constat nous mène alors à considérer le *Service* d'un point de vue différent, plus haut niveau, plus abstrait, afin de dégager des propriétés générales qui puissent être exploitées par un modèle générique de gestion de la mobilité. C'est grâce à un niveau d'abstraction suffisant des concepts connus jusqu'alors qu'il sera possible de définir un tel modèle et de proposer (ou écarter) une solution de mobilité unifiée.

Ce chapitre sera dédié à la redéfinition de la notion de service, conformément à l'état de l'art présenté en première partie de ce mémoire mais vu sous un angle différent, plus haut niveau, afin de dégager des propriétés générales. Ensuite, d'après les travaux et les recherches déjà réalisés nous définirons les fonctions clés d'un modèle générique et complet de mobilité capable d'assurer la continuité des services et une expérience utilisateur optimale.

6.2 Une nouvelle vision du *Service*

6.2.1 Définitions

Il est nécessaire de reconsidérer le concept de *Service*, sans pour autant remettre en cause les définitions et les propriétés présentes dans l'état de l'art. Le terme de « service », comme nous l'avons vu, peut être employé dans de nombreux contextes et prendre des sens relativement différents. Dans le cadre de nos travaux de recherche qui se limitent au domaine des Technologies de l'Information et de la Communication, nous proposons une définition formelle du *service*, cohérente par rapport à la littérature tout en posant une base claire pour les réflexions à venir.

Cette définition doit être suffisamment précise et pragmatique pour permettre la définition d'un ensemble consistant de mécanismes tout en restant assez abstraite pour conserver une approche générique. La voici.

Un service est un algorithme (la logique) qui traite un ensemble de ressources (le contexte) via une application, et qui délivre une fonctionnalité à part entière à au moins un utilisateur.

Cette définition attribue au *Service* un grand nombre de propriétés dont certains nouveaux concepts que nous allons détailler.

Ainsi, un service est constitué de deux éléments : la *logique* qui est la raison d'être du service et le *contexte*, un ensemble de ressources (des données) produites et consommées par la logique. Cette représentation est indépendante de toute implémentation ; l'instanciation du service, nécessaire pour pouvoir être délivré à un utilisateur, est réalisée via une application qui implémente cette logique et gère la contexte (en totalité ou en partie), cf Figure 6.1.

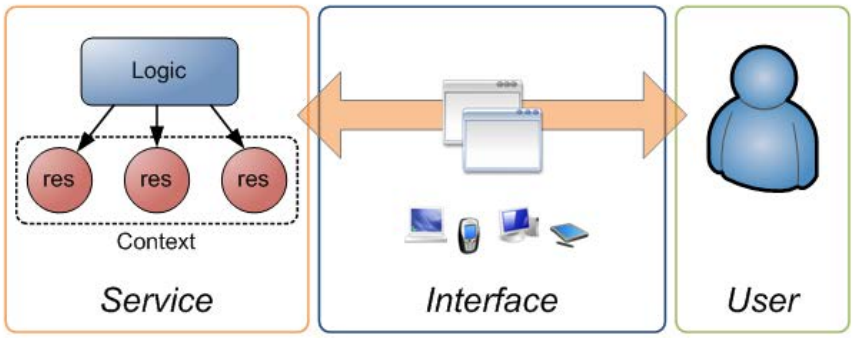

FIGURE 6.1 – Le concept de *Service*.

L'application qui peut être assimilée au système ou même au terminal hôte est qualifiée de manière simplifiée d'« interface ». Il est important de comprendre, et c'est la force de la vision du concept de service proposée, que l'interface n'est qu'une instance du service qui implémente la logique *à sa manière*. Le même service peut ainsi être perçu différemment par l'utilisateur en fonction de l'interface et de ses capacités matérielles, logicielles et environnementales, bien que la logique de service soit la même elle sera interprétée différemment. Cette propriété confère plus de souplesse et d'adaptabilité au service, l'utilisateur sera plus libre dans sa sélection d'interface qu'il ne fera plus par nécessité mais par choix. La Table 6.1 présente quelques exemples de services.

TABLE 6.1 – Exemple de services.

Service	Instance	Host	Resources
Text-edition	MS Word, Vim, Notepad	PC, PDA, Laptop	Raw text, formating, typing history
Telecommunication	H.323/SIP client, Skype	Cellular, PC, car	Audio stream, call history, address book
Video	WMP, Flash, VLC	PDA, Set-top box, PC	Data stream, playback position, bookmarks
Internet browsing	IE, Firefox, Opera	PC, PDA, Cellular	URL, bookmarks, history
Weather forecasts	Desktop/Web widget, mobile service	PC, cellular, coffee maker	Location, language, alarms

La logique d'un service correspond à son type, l'application *MS Word* par exemple, implémente un service d'« édition de texte ». Nous emploierons régulièrement l'exemple du service d'édition de texte car il est simple, connu par tous et s'écarte du service « cliché » réduit à des applications distantes, connectées et basées sur des sessions.

La logique du service d'édition de texte est comme son nom l'indique : éditer, formater, imprimer, ... du texte, rien de définitif, simplement les fonctions que l'utilisateur attend de ce type de service. Le contexte, pour rester dans l'exemple de l'édition de texte pourra être composé de diverses ressources : corps du texte, historique d'édition, styles, dictionnaire, etc.

Un service doit offrir une fonctionnalité « à part entière », cela signifie que la logique isolée doit avoir un intérêt pour l'utilisateur. Un programme qui consisterait par exemple à augmenter la taille de police d'un texte sera utile dans le cadre d'une application plus complexe d'édition de texte mais n'aura aucun sens de manière isolée pour l'utilisateur, il ne pourra être considéré comme un service. Il existe donc une granularité fonctionnelle minimale pour la notion de service.

Enfin, la fonctionnalité produite doit bénéficier directement à « au moins un utilisateur ». En effet, conformément à notre approche depuis le début de nos travaux, nous nous intéressons aux services au travers de l'expérience utilisateur. Certaines fonctionnalités rendent effectivement un service à l'utilisateur de manière indirecte : défragmentation de la mémoire, antivirus, pare-feu, backup, etc. Cependant ces programmes ne servent pas directement l'utilisateur, ils sont dédiés à des éléments spécifiques logiciels ou matériels, souvent dans un but de maintenance ou d'administration du SE, ils ne peuvent être qualifiés de service utilisateur. À noter que la mobilité de ce type de fonctionnalité a peu de sens, transférer une recherche de virus ou l'optimisation du disque dur d'un terminal vers un autre ?

La Figure 6.2 reprend la carte conceptuelle proposée en section 1.3.1, enrichie des nouvelles notions introduites ici.

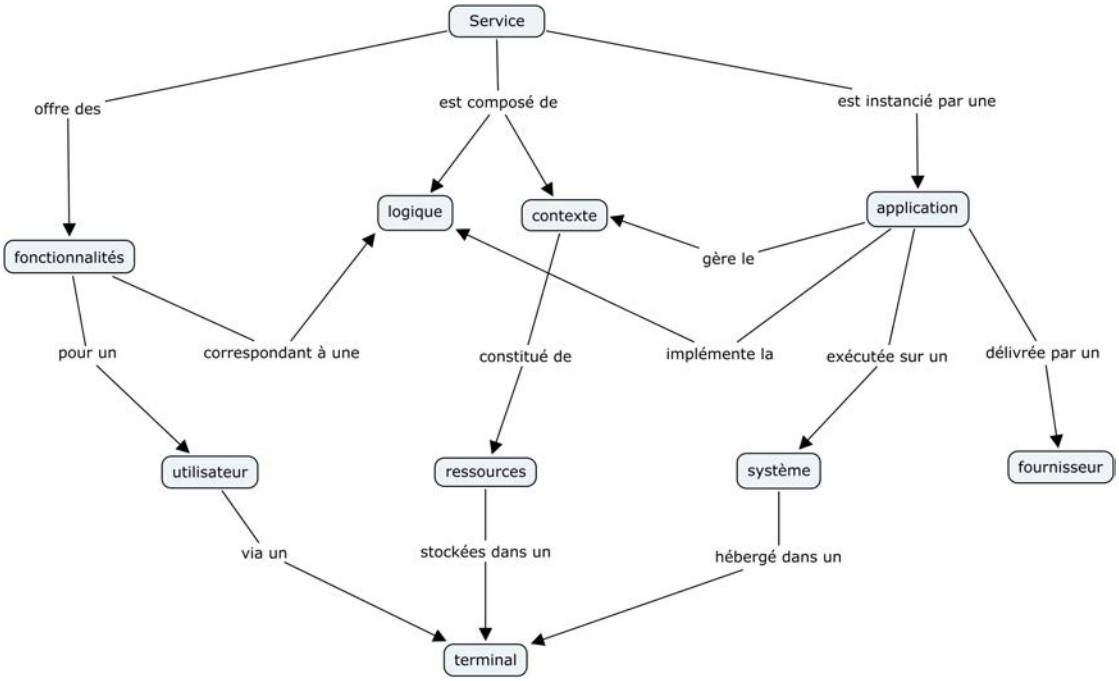

FIGURE 6.2 – Carte conceptuelle actualisée autour de la notion de *Service*.

6.2.2 Les ressources

La logique étant abstraite (implémentée par l'interface), le contexte qui est un ensemble de ressources est l'unique élément matériel du service. C'est donc la partie « encombrante » du service (en termes d'octets) qu'il sera question de transférer durant un handover.

Une ressource est une donnée (une suite de `1` et de `0`) stockée quelque part dans le terminal (mémoire vive, registre, disque dur, ...) et nécessaire à la logique du service en question. L'ensemble des données du service constitue le contexte. Sa taille est quelconque, du bit à la taille infinie pour le cas d'un flux continu de données. La taille d'un contexte rendra un service plus ou moins mobile, les ressources de grande taille ou de type flux poseront naturellement des problèmes particuliers qu'il faudra gérer. Les applications basées sur des protocoles de signalisation com-

plets tel que SIP disposent par exemple de mécanismes spécifiques permettant de rediriger les flux tel que nous l'avons vu dans en section 3.3.3.

Granularité.

L'implémentation de la logique étant laissée à la discrétion de l'application, le contexte ne sera pas toujours exploité à 100% après un transfert. En effet, une ressource peut être inutile si l'application destination ne propose la fonction correspondante. Un exemple simple pour illustrer ce cas : supposons qu'un service d'édition de texte soit transféré d'un terminal qui l'implémente via MS Word vers un terminal doté uniquement de MS Notepad pour gérer l'édition. MS Notepad est un éditeur très simple qui ne supporte que les fonctions de base, le contexte transféré provenant de MS Word, un éditeur plutôt riche, comportera un grand nombre de ressources non gérées par Notepad : styles, images, etc, cf. Figure6.3.

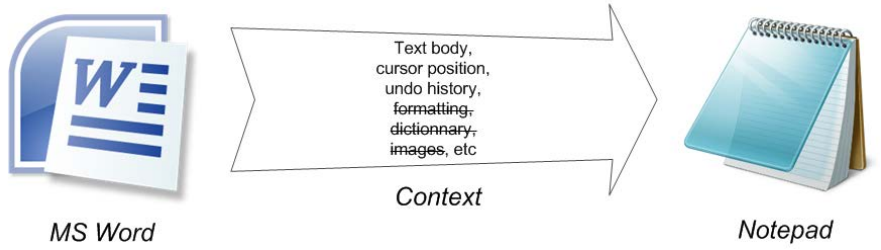

FIGURE 6.3 – Transfert de contexte entre applications hétérogènes.

Le contexte représente une quantité importante de données, l'approche applicative présentée en section 2.3.4 consistait justement à transférer un contexte maximal (à l'échelle du système) afin d'assurer une continuité contextuelle parfaite au détriment de la continuité temporelle. Avec l'exemple du transfert MS Word vers MS Notepad, on se rend rapidement compte de deux choses : premièrement le transfert du contexte peut être optimisé (des ressources sont transférées inutilement), deuxièmement la granularité de découpage des ressources dans le contexte est un facteur de cette optimisation.

Reprenons l'exemple ci-dessus, imaginons que le contexte du service d'édition de texte provenant de l'application MS Word soit le plus gros possible : une seule ressource, un fichier *.doc* (propriétarire MS Word) contenant le document complet.

La réutilisabilité d'un tel contexte par une autre application est très faible, rendant le service inadaptable. Si par contre, toutes les données du document sont séparées dans le contexte, chaque ressource sera plus petite et plus facilement assimilable par une autre application ; statistiquement une part plus importante du contexte sera adaptable à chaque transfert. On peut imaginer que lors du transfert MS Word vers MS Notepad, les ressources correspondant au contenu du texte brut et la position du curseur ont pu être gérées par le bloc notes

Un contexte détaillé, comportant des ressources minimalistes est la condition d'un transfert optimisé, une adaptation plus importante et donc une expérience utilisateur améliorée.

Organisation.

Nous avons vu au-dessus que par définition un contexte n'était pas forcément réutilisé dans son intégralité en fonction des capacités de l'application. Le contexte étant composé d'un grand nombre de ressources, il est nécessaire qu'il reste cohérent et consistant, on parle alors d'état « stable ». Cet état doit être atteint lorsque le contexte est transféré sinon la logique ne peut délivrer la fonctionnalité de manière correcte.

La consistance du contexte est assurée lorsque l'ensemble des ressources requises par la logique sont bien présentes et en nombre correct. La cardinalité de chaque ressource doit être respectée, par exemple un service d'édition de texte aura probablement une et uniquement une ressource correspondant au corps du document (texte brut) et un nombre indéterminé d'actions historiques (permettant d'annuler une édition précédente). De plus, il peut exister des relations entre les ressources, la présence de certaines en impliquant d'autres ; les ressources représentant l'historique d'édition ou le formatage du texte n'auraient aucun sens sans le corps du texte.

Ces contraintes du contexte de service pourraient être représentées sous forme de graphe permettant d'assurer la consistance du contexte mais aussi de vérifier qu'une application est apte à implémenter un service d'après son contexte, cf. Figure 6.4.

La consistance du contexte n'est pas suffisante pour garantir sa stabilité, il est également nécessaire de s'assurer de sa cohérence en considérant l'état de chacune

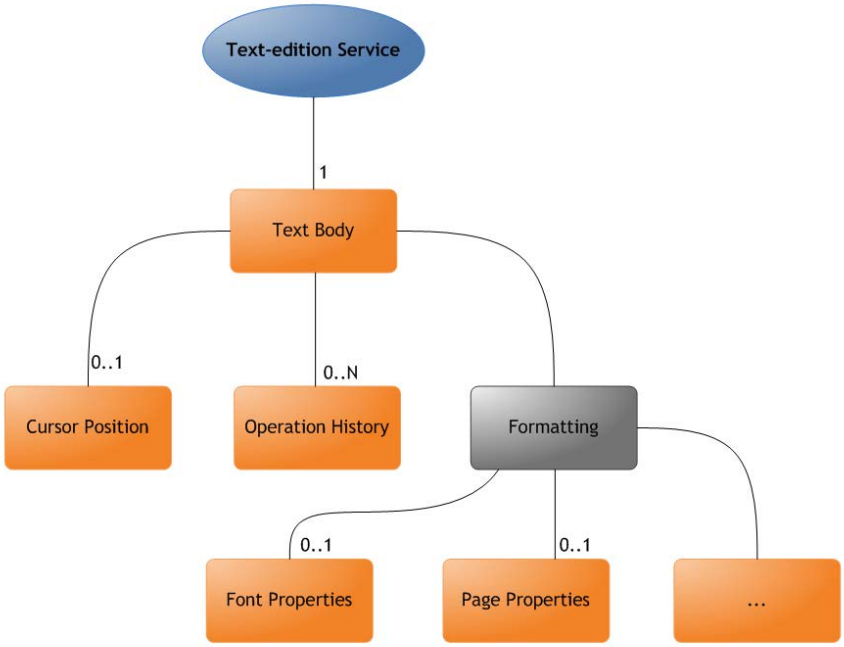

FIGURE 6.4 – Contexte d'un service d'édition de texte.

de ses ressources. En effet, pendant la fourniture d'un service le contexte est utilisé par la logique en lecture mais aussi en écriture. Les ressources étant potentiellement modifiées en continu, il peut arriver que des données soient périmées ou dans un état temporaire lors du transfert du contexte.

Dans le premier cas, le contexte est transféré alors qu'une action vient de modifier seulement une partie des ressources. Par exemple, si un transfert intervient pendant l'édition d'un texte et que la dernière action de l'utilisateur n'a pas encore été enregistrée par l'historique d'édition, les deux ressources ne seront pas cohérentes. Le contexte ainsi transféré entraînera des comportements non désirés, ici la touche d'annulation ne corrigeant pas la dernière action réalisée (mais probablement la précédente).

Dans le second cas, une ressource est en cours de modification lorsque le contexte est transféré. La donnée n'étant pas prête, deux solutions se présentent. Soit l'action en cours est annulée et la ressource (voire tout le contexte) est restaurée à sa valeur précédente avant que le contexte soit transféré ; cette approche nécessite que l'état du contexte soit sauvegardé avant d'effectuer l'action (pour

permettre l'annulation). Soit l'action poursuit jusqu'à son terme avant le transfert du contexte ; la durée de l'opération n'étant pas nécessairement connue a priori, le transfert peut être longuement retardé.

Cette nouvelle approche du concept de service offre une vision plus intuitive et de nouvelles perspectives. Il devient possible de réfléchir à un modèle de continuité tout en faisant abstraction de l'environnement hétérogène des services et essayer de dessiner des mécanismes de mobilité génériques.

6.3 Un modèle unifié de mobilité

D'après l'étude des solutions de mobilité présentées dans l'état de l'art et grâce à l'expérience acquise en développant des prototypes de continuité dans divers environnements, nous avons identifié les forces et les faiblesses de chacune des approches afin de dessiner les contours d'un modèle générique et complet de mobilité de service. Nous avons ainsi dégagé cinq problématiques correspondant à cinq mécanismes clés qui constituent les différentes étapes successives nécessaires à une solution globale de continuité de service. Les voici.

- **Découverte**. Identification des différents acteurs : utilisateurs, terminaux, services, etc.
- **Déclenchement**. Désignation et orchestration du transfert.
- **Capture**. Collecte des différentes informations, préparation des données au transfert.
- **Transfert**. Mobilité effective du contexte et des informations de contrôle.
- **Reprise**. Continuation du service, adaptation à l'environnement.

En nous basant sur le concept de service tel que défini dans la section précédente, nous allons détailler chacun de ces mécanismes clés. Mais avant de penser à mettre en œuvre le moindre mécanisme de mobilité, il est nécessaire de connaître l'environnement. Les solutions que nous avons successivement étudiées étaient basées sur un environnement « intelligent » avec des structures spécifiques qui offraient un certain nombre de fonctions de base : traçage, identification, déclenchement, etc. Dans notre approche générique, il n'est plus possible d'imposer des contraintes d'infrastructure qui limitent son application ; les mécanismes doivent

être indépendants et autonomes. Cependant l'environnement existe, il nous a donc fallu choisir un cadre le plus neutre possible qui fasse abstraction de toute fonction extérieure.

Cet environnement basique sera constitué uniquement de terminaux connectés entre eux de manière ad-hoc via leurs interfaces réseaux. Nous appellerons cet ensemble de terminaux connectés appartenant à un utilisateur et formant un réseau *overlay* (ou surcouche), un « environnement personnel de services » (ou *Personal Service Environment*, PSE). Le PSE est en quelque sorte matérialisé par la sphère électronique de l'utilisateur que nous avions mentionnée en introduction de ce mémoire. Les mécanismes génériques que nous allons définir s'appliquent dans le cadre d'un PSE pour un utilisateur donné, cf. Figure 6.5.

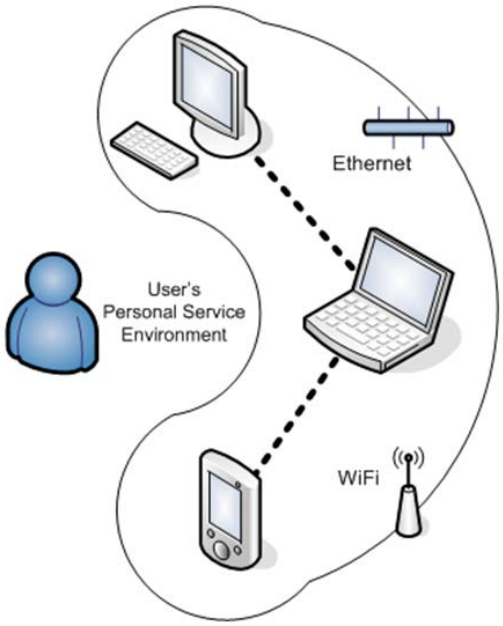

FIGURE 6.5 – Un environnement personnel de service (PSE).

6.3.1 Découverte

Avant de penser à mettre en œuvre le moindre mécanisme de mobilité, il est nécessaire d'identifier les acteurs en présence : terminaux et services. L'environnement étant centré sur l'utilisateur, le PSE doit être construit de manière ad-hoc

autour de lui en gérant les terminaux qui s'y ajoutent et s'en retirent. Chaque composant doit être identifié et authentifié afin d'assurer la protection des services de l'utilisateur.

Les services instanciés par les terminaux du PSE doivent également être connus par chaque interface, ainsi le PSE doit se comporter tel un système complet disposant d'autant d'interfaces qu'il est constitué de terminaux. Les interfaces doivent permettre à l'utilisateur d'accéder de manière transparente à ses services, chaque terminal doit donc avoir une vision globale du système.

La collecte d'un maximum d'informations sur ces éléments découverts est primordiale pour les étapes suivantes. Les caractéristiques détaillées des interfaces et des services permettront de guider les mécanismes de déclenchement, de transfert et d'adaptation.

6.3.2 Déclenchement

La seconde phase est le déclenchement du transfert. Comme nous avons vu dans les approches existantes et dans les prototypes implémentés, le déclenchement est un mécanisme particulièrement important et complexe. Il peut s'effectuer de manière manuelle, les éléments découverts à l'étape précédente seront alors désignés par l'utilisateur à partir d'une interface du PSE afin de mettre en œuvre la suite du transfert. Les modes *push* et *pull* vus précédemment (cf. 2.1.3) doivent être applicables, permettant à l'utilisateur de transférer un service précis vers l'interface locale ou un vers terminal distant.

Un mode automatique est également possible mais uniquement basé sur les informations disponibles au niveau de l'interface, il n'existe aucun composant d'infrastructure qui pourrait induire le déclenchement tel que le *Presence Server* dans l'IMS. Le détail des informations collectées dans la phase de découverte révèlent ici toute leur importance, une connaissance précise de chaque interface du PSE permettra des déclenchements efficaces tenant compte de l'environnement : faible batterie, meilleur terminal disponible, etc.

6.3.3 Capture

Une fois qu'un service a été choisi et que son transfert a été requis, le PSE via le terminal correspondant doit identifier l'ensemble des ressources du contexte

en question et le capturer dans un état stable. Cette capture peut se visualiser comme un « cliché » du contexte (*service snapshot*) juste avant son transfert effectif (à l'image du *bookmark média* présenté dans le chapitre 5). Ce cliché qui fige le contexte permet à l'utilisateur de retrouver des points de référence une fois le transfert réalisé, offrant à l'utilisateur une expérience similaire du service sur l'interface d'origine comme sur celle de destination.

Le terminal destination ne peut accepter qu'un contexte stable, la capture des différentes ressources doit donc être réalisée de manière efficace, cohérente et consistante (cf. 6.2.2) ce qui est parfois difficile considérant l'hétérogénéité des ressources.

6.3.4 Transfert

Vient alors le mécanisme de transfert qui consiste à envoyer l'ensemble des données nécessaires à la reprise du service d'une interface du PSE à une autre conformément à la désignation réalisée pendant la phase de déclenchement. La Figure 6.6 illustre le principe de transfert basé sur la définition de service haut niveau que nous avons proposée en début de ce chapitre. La logique de service est dissociée de son implémentation, seul le contexte est déplacé de l'application origine vers l'application destination qui peut être différente mais implémente nécessairement la même logique.

Le contexte à transférer pouvant être de taille importante ou comporter des ressources de type flux continu de données, des mécanismes spécifiques et optimisés doivent être mis en œuvre pour masquer la complexité et l'hétérogénéité des ressources. La conception même du contexte décrite en section 6.2.2 apporte déjà quelques pistes que nous verrons en détails dans le chapitre suivant 7 qui précise leur implémentation.

6.3.5 Reprise

La reprise du service est certainement la phase la plus critique mais également la plus importante du mécanisme de transfert. Ce mécanisme est généralement négligé par les solutions que nous avons étudiées pour la simple raison que la continuité contextuelle est peu adressée et que les environnements d'origine et de

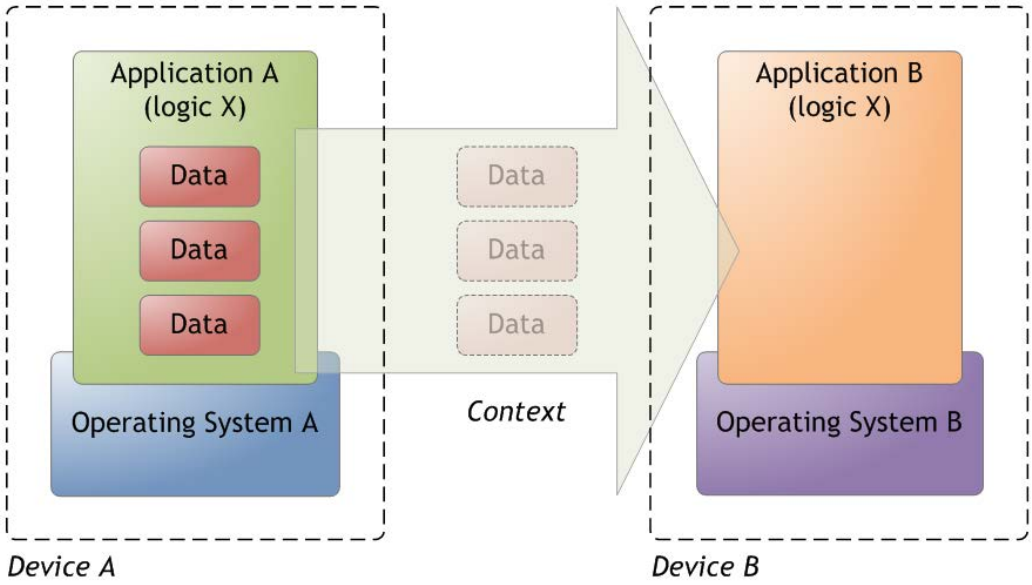

FIGURE 6.6 – Principe de transfert.

destination sont généralement identiques. Avec l'objectif d'une expérience utilisateur optimale, la continuité contextuelle est pour nous une priorité, de plus le PSE est intrinsèquement hétérogène et ce à tous les niveaux. Cette phase étant la dernière dans le chaîne de fonctions du processus de mobilité, celle qui a un impact direct sur l'utilisateur.

Alors en quoi consiste la reprise du service (ou *service resume*) ? Tout d'abord, le contexte qui a été transféré lors de la phase précédente doit être transmis à l'application qui gère la logique en question. La logique connue depuis la phase découverte ayant permis l'éligibilité et la désignation de cette interface comme destinataire du transfert. L'application destination est identifiée et automatiquement exécutée afin de continuer le service en mobilité.

Les mécanismes qui entrent en jeu ici assurent l'adaptation du service aux propriétés de la nouvelle interface, en d'autres termes ils permettent à la nouvelle application d'exploiter le contexte transféré et de restituer à l'utilisateur une expérience comparable à celle offerte par l'interface d'origine. L'efficacité de cette étape dépend directement de la présentation du contexte : granularité et organisation des ressources, cf. 6.2.2.

6.4 Conclusion

Ce premier travail nous confère une vision à la fois suffisamment abstraite et pragmatique de la notion de *Service*. Il nous a permis d'appréhender avec précision les relations qui existent entre l'utilisateur, ses terminaux, ses applications et ses services. Nous avons analysé de manière conceptuelle et abstraite, indépendamment de leur type, les notions de logique et de ressource afin d'identifier un certain nombre de propriétés qui se révèlent essentielles dans la gestion de la mobilité. Nous avons présenté les fonctions (haut-niveau) clés d'un modèle de continuité générique adapté à un environnement hétérogène.

Nous avons également introduit une vision neutre de l'environnement, le PSE, qui présente les services de l'utilisateur au centre d'un système accessible via ses terminaux qui ne sont finalement que des interfaces. On y retrouve l'idée de « sphère électronique » qui entoure l'utilisateur (cf. *Introduction générale*). L'objectif de la continuité dans ce cadre est de permettre à l'utilisateur d'accéder librement et de manière transparente à ses services mais de manière adaptée à l'interface choisie. L'adaptation est essentielle, elle offre des points de référence à l'utilisateur dans sa relation avec le service mais surtout elle exploite l'hétérogénéité de l'environnement afin de justifier la mobilité de service.

Le travail de définition et de positionnement des concepts présentés dans ce chapitre nous a permis de prendre de la distance par rapport à l'état de l'art qui n'apportait pas de solution satisfaisante tout en tirant des enseignements des mécanismes existants. Le modèle générique proposé ici est ambitieux et n'est pas sans poser de nombreux problèmes, nous avons donc entrepris une implémentation et un prototypage afin d'évaluer sa faisabilité et d'identifier les défis techniques qui se présentent. Ce sera l'objet des deux prochains chapitres.