L'Environnement de Développement Dynamique (EDD) pour le prototypage rapide d'interfaces graphiques

André Jodoin [1,2] et Michel Desmarais [1]

1. École Polytechnique de Montréal
Département Génie Informatique
C.P. 6079 succ. Centre-ville
Montréal, Qc. Canada H2C 3A7

2. Agence Spatiale Canadienne
Centre spatial John H. Chapman
6767, route de l'Aéroport
Saint-Hubert, Qc. Canada J3Y 8Y9

RESUME

Dans cet article, nous décrivons l'EDD, un environnement de développement très flexible pour le prototypage rapide des interfaces graphiques. L'EDD a été conçu et développé à l'Agence Spatiale Canadienne (ASC) dans le but de permettre aux utilisateurs experts de rapidement développer des prototypes d'interfaces humain-machine (IHM) pour le contrôle de systèmes robotiques. Il s'inspire des architectures distribuées et des langages de description d'interface pour offrir un environnement dans lequel les objets graphiques (*widgets*) peuvent être intégrés par la manipulation directe tout en étant « dynamiques », dans le sens où ils s'exécutent dans ce même environnement. L'EDD permet au concepteur de faire rapidement l'essai du comportement de l'interface et offre une très grande flexibilité pour intégrer de nouveaux objets graphiques, tout en minimisant le besoin de codage. Les environnements de développement tels que Netbeans, .Net et Eclipse requièrent que la compilation et l'exécution de l'IHM soient faites de façon consécutive et séparée de l'activité de développement l'IHM. L'EDD se distingue en offrant à l'utilisateur la possibilité de tester l'IHM au même moment qu'il la développe sans avoir à relancer la plateforme.

MOTS CLÉS: environnements de développement, langages de description d'interface, XML, UIML, composants, environnements distribués, programmation visuelle.

ABSTRACT

We present an interactive development environment (IDE) designed and implemented at the Canadian Space Agency (CSA) to assist expert users in rapidly developing graphical user interface prototypes for the control of robotic systems. The Dynamic Development Environment (DDE) offers a highly flexible tool for interface design and development. The environment allows the direct manipulation and integration of running widgets to rapidly build a live interface. New widgets can be readily integrated and their behaviour tested within this environment, skipping code compilation and freeing the interface designer from most of the programming effort. Other IDE`s such as Netbeans, .Net and Eclipse require separate compilation and execution steps. The DDE offers a new possibility by allowing the user to test the interface while it is being built without the need to launch the platform again.

GENERAL TERMS: DESIGN, LANGUAGES, EXPERIMENTATION, HUMAN FACTORS.

CATEGORIES AND SUBJECT DESCRIPTORS: D.2.2 [Software Engineering]: Design Tools and Techniques - software libraries; user interfaces.

KEYWORDS: IDE, XML, UIML, interface description languages, distributed environments, visual programming.

INTRODUCTION

Le prototypage des IHMs permet aux utilisateurs de bien faire connaitre leurs besoins aux développeurs et de s'assurer que ceux-ci obtiendront une IHM qui répond à leurs exigences. Il est donc nécessaire d'avoir accès à des outils performants permettant le prototypage des IHMs. Des outils qui sont les plus efficaces, simples et complets possible. La problématique reliée au prototypage rapide des IHMs existe depuis longtemps. On a qu'à penser au système *Druid* [4] présenté au début des années 90 pour le prototypage rapide d'IHMs.

Nous présentons une architecture innovatrice et flexible pour le développement d'interfaces graphiques que nous nommerons l'environnement dynamique de développement (EDD). L'EDD permet d'intégrer les objets graphiques (widgets) par la manipulation directe et de les rendre «dynamiques», dans le sens où ils s'exécutent dans ce même environnement. Une brève version précédente de ce système a été présentée dans [2] et nous décrivons ici plus en détail l'architecture et les derniers développements. D'une perspective d'utilisation, l'EDD se distingue des autres environnements de développement tels que Netbeans, .Net et Eclipse avec son « Visual Editor » en offrant la possibilité à l'utilisateur de tester l'IHM à mesure qu'il la construit sans nécessiter d'étapes sup-

IHM 2009, 13-16 Octobre 2009, Grenoble, France

plémentaires de compilation et d'exécution. Il permet au concepteur de faire rapidement l'essai du comportement de l'interface et offre une très grande flexibilité pour intégrer de nouveaux objets graphiques, tout en minimisant le besoin de codage.

L'environnement est implanté en Java et intégré au cadre d'Eclipse. Il repose sur l'usage de la réflexivité de Java et sur les principes des langages de déclaration d'interfaces graphiques telles qu'UIML [1].

Il a été conçu et développé à l'Agence Spatiale Canadienne (ASC) dans le but d'offrir aux utilisateurs experts un système leur permettant de rapidement prototyper des IHMs pour le contrôle de systèmes robotiques. L'ASC développe de nombreux projets en robotique et avait besoin d'un environnement de développement simple et efficace qui permettrait aux utilisateurs de rapidement développer des prototypes d'IHMs au tout début des projets et avant même que les systèmes à contrôler ne soient complètement définis. Une caractéristique absente dans les environnements disponibles était celle de pouvoir faire la conception complète d'une IHM de contrôle d'un système connecté par une interface de programmation (API), sans pour autant que le système soit disponible au moment de la conception.

Lors de sa conception, il est rapidement devenu évident qu'un tel environnement pourrait servir à développer des IHMs pour plusieurs autres types de systèmes. L'EDD a notamment été utilisée pour développer des IHMs pour le contrôle de robots (CART et NORCAT) et pour développer l'IHM d'une application automatisée de test d'IHM qui simule les interactions de l'utilisateur (MARVIN).

CARACTÉRISTIQUES PRINCIPALES

Les environnements de développement logiciel disponibles sur le marché offrent des outils d'assistance au développement d'IHMs. Ces outils permettent la manipulation directe des objets graphiques qu'ils rendent accessibles à l'aide d'une palette ou autres mécanismes. Entre autres, ils permettent aussi de gérer la mise en forme des écrans et génèrent le squelette du code source des méthodes de traitement des évènements devant être complétées par le programmeur. Si ces assistants sont forts utiles, le développement des IHMs requiert encore beaucoup de travail et de connaissances, notamment parce que les objets graphiques sont reliés au noyau fonctionnel par la programmation à même les méthodes d'écoute d'évènements de ces mêmes objets. Cette façon de faire rend ardue l'utilisation d'une même IHM pour contrôler divers procédés car elle ne propose pas de mécanisme ou d'architecture voué à la définition d'IHMs dont les systèmes à contrôler ont une nature polymorphique, un environnement qui ne tient compte que de l'interface et non pas de ses réalisations. Ceci constitue une caractéristique que l'EDD possède et qui le distingue des autres environnements. Il est, bien sûr, possible de coder l'aspect polymorphique du contrôle exercé par l'IHM. Mais le fait qu'elle soit offerte par l'EDD rend ce type de construction beaucoup plus intuitif.
Enfin, l'approche par la programmation des méthodes d'écoute des événements nécessite beaucoup de connaissances en programmation et les comportements ainsi définis ne sont pas d'emblée réutilisables. Les comportements définis dans l'EDD sont des objets qui peuvent être connectés à tout événement très simplement en les associant graphiquement.

Le fait de déplacer plusieurs aspects de la construction d'IHM de la programmation vers la manipulation directe permet aux utilisateurs experts de participer à la conception et au développement des IHMs [3]. Les utilisateurs, par le prototypage rapide, peuvent ajuster finement l'IHM aux tâches qu'ils doivent réaliser.

La communication entre l'utilisateur, les constituants graphiques et les constituants logiques s'établit à partir de levé d'événements qui provoquent l'activation des comportements. Ce mode de communication est particulièrement compatible avec le développement d'IHM par la manipulation directe [3]. En effet, il est facile d'établir la relation entre les actions de l'utilisateur et la réaction du système. De plus, les comportements définis sont réutilisables et peuvent être associés à plusieurs événements.
L'utilisateur n'a qu'à définir les comportements de l'IHM en reliant graphiquement les événements levés par les composants graphiques aux diverses fonctionnalités du noyau fonctionnel. L'EDD offre des assistants qui utilisent la réflexivité de Java pour proposer à l'utilisateur les méthodes de l'interface du noyau fonctionnel qu'il peut brancher aux événements levés par les composants graphiques.

L'aspect dynamique des composants et des comportements constitue la distinction principale de l'EDD par rapport aux autres environnements de développement d'IHM. Les environnements tels que Microsoft Visual Studio et .Net, ou encore NetBeans de Sun Micro Systems ou Eclipse.org proposent une approche en trois étapes qui consistent en la programmation, la compilation et l'exécution. L'approche proposée ici est différence : toute modification apportée à l'IHM est applicable immédiatement et peut-être testée directement dans l'environnement sans la nécessité de relancer l'EDD. En effet l'EDD est à la fois l'éditeur d'IHM visuel, l'environnement de développement Java et l'IHM elle-même. Autant les modifications effectuées par manipulation directe à partir de l'éditeur d'IHM visuel que les modifications apportées au code source Java contribuant au contrôle de l'IHM sont appliquées de façon dynamique et aussitôt sauvegardées.

De plus, l'architecture de cette plateforme est basée sur le patron modèle-vue-contrôleur (MVC). Dans le contexte de l'EDD, l'avantage principal de cette approche réside dans la séparation entre les vues, les données, et le modèle d'interaction entre les deux. En effet, cette séparation facilite l'entretien et l'amélioration de l'EDD sans pour autant invalider les vues déjà construites par l'utilisateur, en plus de favoriser leur réutilisabilité.

COUPLAGE DE L'IHM AU NOYAU FONCTIONNEL

L'un des choix de conception de l'EDD consiste à augmenter l'outillage de la plate-forme Éclipse tout en profitant des nombreux avantages de son architecture. Les caractéristiques ajoutées offrent des fonctionnalités supplémentaires qui simplifient la conception, permettent l'application immédiate des modifications apportées par l'utilisateur aux vues et aux comportements de l'IHM, et permettent de relier l'IHM à différents noyaux fonctionnels possédant une interface commune.

En effet, en substituant le noyau fonctionnel par une souche (un remplacement postiche, ou « dummy » en anglais) dont l'interface de programmation sera la même que celle du vrai noyau fonctionnel, il est possible de concevoir l'IHM sans que le noyau fonctionnel ne soit disponible. La souche peut être remplacée par le noyau fonctionnel lorsqu'il est disponible. De la même façon, il est possible de contrôler au choix des noyaux fonctionnels dont les interfaces de programmation sont identiques à partir d'une même vue. Par exemple, un système robotique pourrait être contrôlé par la même IHM qu'une simulation de ce même système.

Une ou plusieurs interfaces Java serviront à représenter le noyau fonctionnel. Ces interfaces Java constituent un modèle du noyau fonctionnel dont l'utilisateur fera usage pour concevoir l'interface graphique en s'y référant constamment. C'est-à-dire que les architectures de l'IHM et du noyau fonctionnel resteront compatibles tout au long du processus de conception [6]. Et comme il ne s'agit que d'un modèle du noyau fonctionnel, sa forme concrète peut être polymorphe et se manifester à travers diverses formes qui seront compatibles avec le modèle.

Dans ce sens, L'EDD introduit les concepts de systèmes, de cibles et de contextes d'exécution. Le système représente une interface définissant une API dont le but est d'exposer les méthodes auxquelles sont branchées les objets graphiques. La cible représente une implémentation proposée d'un système. Enfin le contexte définit à quelle cible est associé un système. Le concepteur ou l'utilisateur peut, durant l'exécution, sélectionner un différent contexte que celui utilisé par l'IHM pour changer le procédé qu'il contrôle. Cette dernière caractéristique a pour but de promouvoir la réutilisation des interfaces graphiques et permet de relier l'IHM à l'un ou à l'autre des noyaux fonctionnels représentés par les cibles en sélectionnant le contexte correspondant, et ce, pendant l'exécution de l'IHM.

Concrètement, tel qu'illustré à la figure 1, le système est représenté par une interface Java, la cible une classe Java qui réalise cette interface, et le contexte, la relation entre le système et la cible, c'est-à-dire entre une interface Java et une classe qui la réalise.

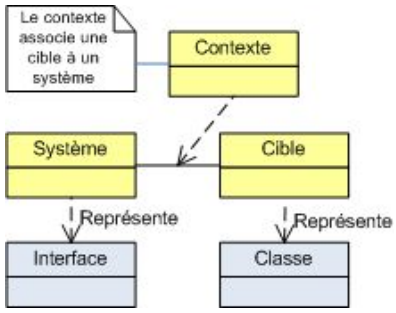

Figure 1 : L'association tertiaire système-cible-contexte

La figure 2 présente un exemple de l'utilisation des contextes d'exécution. Ici, le système Robot définit l'interface permettant le contrôle d'un robot. Ce système est associé à une interface Java IRobot qui en définit l'API. Deux réalisations de cette interface Java sont présentées, les classes Robot et Simulateur. Ces deux classes sont associées aux cibles correspondantes Robot et Simulateur. Les deux classes sont des réalisations de l'interface Robot. L'une communique avec un robot et en permet le contrôle, l'autre substitue le robot par un simulateur. En choisissant le contexte Robot, l'utilisateur configure l'IHM pour qu'elle s'interface directement avec le robot, tandis qu'en choisissant le contexte Simulateur, l'utilisateur fait en sorte que l'IHM contrôle un simulateur du robot.

L'EDD permet de changer le contexte d'exécution à tout moment de façon dynamique. Comme les comportements sont construits à partir de l'API du système (l'interface Java) et non celle des cibles (réalisation de l'interface), il est possible d'aiguiller les appels de méthodes vers les classes sélectionnées par le contexte.

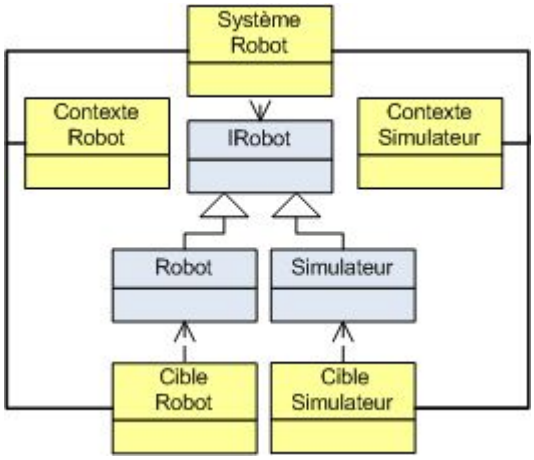

Figure 2 : Exemple de la relation système-cible-contexte

L'EDD : UNE EXTENSION DE LA PLATEFORME ECLIPSE

Tel qu'illustré à la figure 3, L'EDD est constituée de plugiciels qui complémentent l'atelier de développement Java d'Eclipse en offrant divers points d'extension. Entre autres, l'EDD offre un assistant de création de projet, un éditeur central de configuration de l'IHM, un éditeur graphique pour la création des pages par manipulation direct, une palette de composants graphiques, des éditeurs de propriétés, des vues de survol (arbre de la page), des valideurs, des actions pour la fonction couper/coller et pour la création des comportements et un point d'extension permettant aux plugiciels externes de fournir de nouveaux composants graphiques.

Les projets de l'EDD sont assemblés par l'assistant de création de projet qui crée les répertoires et les fichiers qui constituent le squelette du projet. Ce répertoire se situe dans l'espace de travail de la plateforme. Il est à la fois un plugiciel Eclipse, un projet de type « Rich Client Platform » (RCP), un projet Java et un projet EDD.

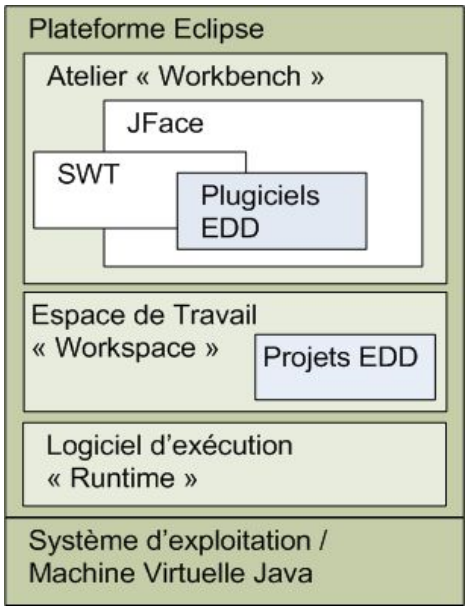

Figure 3 : L'EDD comme extension de l'environnement de développement Java d'Eclipse

Le projet créé par l'EDD comporte aussi un fichier de produit « .product ». Ce fichier permet à l'utilisateur, à travers l'assistant de déploiement de produit Eclipse, de déployer l'IHM en un produit complètement autonome dans lequel tous les plugiciels requis à son exécution sont inclus. Dans ce contexte, l'IHM ne s'exécute plus dans un environnement de développement, mais plutôt dans un environnement d'exécution dénudé des outils de l'atelier Eclipse et de ceux de l'EDD. Seuls les logiciels d'exécution permettant aux pages construites à l'aide de l'EDD de s'afficher y sont présents. Ce mode de déploiement n'est utile que lorsque la construction de l'IHM est complétée et que le produit est prêt à être utilisé aux étapes de production ou d'opération, selon l'objectif visé.

Les pages (ou vues) créées par l'utilisateur sont conservées en mémoire par l'EDD dans des objets faisant partie d'un modèle de type « Eclipse Modeling Framework » (EMF). Le modèle EMF est à son tour encodé en format XML lors de la sauvegarde sur fichier dans le répertoire du projet. Les fichiers, lors de l'ouverture d'un projet EDD, sont décodés de leur format XML à partir duquel est reconstruit le modèle EMF. Le rendu graphique de l'IHM résulte de l'interprétation du modèle par l'EDD et des comportements d'affichage de chacun des composants graphiques du modèle. Cette approche offre une flexibilité additionnelle en permettant une interprétation adaptée à l'environnement cible.

Il s'agit d'une approche qui s'apparente à celle de l'ingénierie de l'IHM dirigée par les modèles [5] qui permet de déployer les IHMs dans des espaces interactifs hétérogènes et dynamiques. L'utilisation du modèle permet à l'EDD de dynamiquement interpréter les compor-

tements et la mise en page de façon à permettre leur utilisation immédiate par l'utilisateur. De plus, et bien que cela n'ait pas été réalisé dans l'EDD, cette approche rend possible l'interprétation du modèle de façon à satisfaire les besoins de plateformes hétérogènes. On pourrait par exemple déployer l'IHM sur un portable ou sur une page web.

Cette approche est différente de celles employées par d'autres extensions Eclipse comme le «Visual Editor » dont le produit est du code Java et SWT par opposition au modèle d'IHM généré par l'EDD qui peut être interprété selon les besoins de l'environnement d'exécution.

LES COMPOSANTS GRAPHIQUES ACTIFS

Une des caractéristiques importantes de l'EDD est la possibilité qu'il offre au développeur de fournir un objet graphique dynamique rendu disponible à travers un point d'extension Eclipse défini par L'EDD. Lorsque le nouveau composant est détecté, il est ajouté à la palette d'objets graphiques et son interface programmatique devient publique et visible directement au concepteur de l'IHM. Celui-ci peut alors prendre connaissance des événements qu'il peut gérer et le connecter par manipulation directe, par la saisie de propriétés ou par l'appel des méthodes du composant à d'autres objets dans l'IHM en développement. Il est aussi possible de basculer l'environnement dans le mode exécution et valider d'emblée le comportement du nouveau composant graphique.

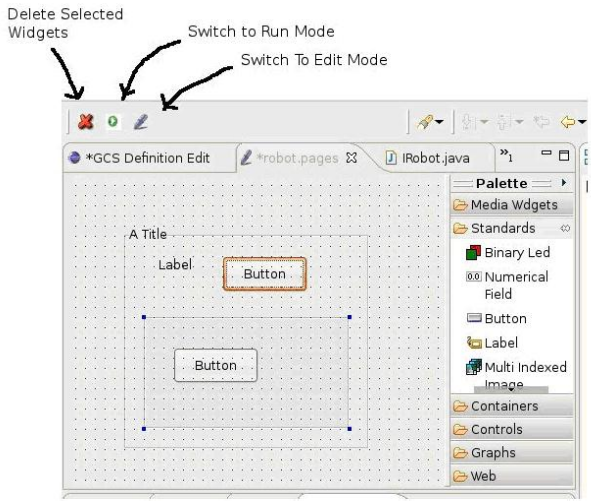

Figure 4 : La palette des composants graphiques actifs et les modes de l'éditeur visuel

L'éditeur de pages est présenté à la figure 4. On peut y apercevoir le canevas sur lequel sont déposés et manipulés les composants graphiques, un grillage offrant un guide pour l'ajustement de leurs positions et la configuration de leurs dimensions, quelques boutons permettant d'effacer les composants sélectionnés et de basculer l'éditeur du mode édition au mode exécution, et la palette de composants graphiques.

LA PROGRAMMATION VISUELLE

L'éditeur graphique de pages est accompagné d'une vue de survol du modèle de la page. On y retrouve les composants graphiques et les événements qu'ils lèvent. En cliquant sur le bouton droit de la souris lorsqu'elle est localisée sur un événement, l'utilisateur active l'affichage d'un menu contextuel dans lequel sont offerts plusieurs choix associés à des actions pouvant être rattachées à l'événement. Par exemple, l'appel d'une méthode d'un système. Dans ce cas, l'utilisateur choisi un système, et l'EDD, grâce à la réflexivité de Java, procède à l'inspection de l'interface Java pour révéler ses méthodes et leurs signatures. Une fois la méthode sélectionnée, ses paramètres sont ajoutés à la vue de survol et pour chacun, une action devra être sélectionnée de façon à lui assigner une valeur. L'utilisateur pourra alors choisir entre la valeur d'une propriété, une constante, une conversion numérique, ou encore l'appel d'une méthode d'un système. À noter que lorsqu'un composant graphique offre une ou des interfaces Java le représentant à l'EDD, l'EDD le considère aussi comme un système duquel les méthodes peuvent être rattachées aux comportements définis par l'utilisateur.

Les comportements ainsi construits par l'utilisateur sont continuellement validés par l'EDD qui s'assure que tous les paramètres sont assignés et que les valeurs assignées sont compatibles avec les types attendus par la méthode appelée. Lorsqu'une erreur est décelée, l'EDD marque la méthode d'un icône symbolisant une erreur. De cette façon l'utilisateur sait lorsqu'il a complété l'assemblage du comportement correctement.

La technique d'assemblage proposée par l'EDD est plutôt laborieuse. Il serait utile, comme développement futur, de concevoir une vue graphique dans laquelle les événements, les propriétés, les méthodes et leurs paramètres peuvent facilement être exposés de façon à pouvoir les connecter en associant les points d'un graphique à l'aide de la souris.

LA PROGRAMMATION JAVA DYNAMIQUE

Un autre objectif visé consiste à permettre la programmation dynamique d'interfaces et de classes Java qui soient automatiquement détectées, compilées et chargées par la machine virtuelle Java durant l'assemblage sans nul besoin de redémarrer l'EDD. De cette façon, il devient possible de faire usage de ces interfaces et classes Java immédiatement en les reliant à l'interface graphique. Cette fonctionnalité offre au concepteur la possibilité de développer un système complexe tout en lui

permettant de le brancher à l'interface graphique sans devoir relancer la plate-forme.

À cette fin, L'EDD est munie d'un chargeur de classe dynamique qui permet le chargement des classes directement à partir des fichiers (fichiers dont le nom se termine par l'extension « .class ») situés dans le répertoire du projet EDD. Il permet aussi le chargement des classes accessibles par le chemin de classes défini par le projet lui-même. Cette caractéristique n'est pas normalement offerte par la plateforme Eclipse qui requiert que la plateforme soit lancée avec un nouveau chemin de classes défini par la configuration du projet.

Le chargeur de classe dynamique hérite directement du chargeur de classe du plugiciel le plus dépendant. De cette façon, il lui est possible de localiser toute classe référée par le code de l'utilisateur ou par l'EDD lui-même.

L'INTERFACE DE PROGRAMMATION D'APPLICATION (API).

Pour offrir à l'utilisateur encore plus de flexibilité, L'EDD offre une API permettant de faire appel à ses fonctionnalités par la programmation. Par exemple, le code de l'utilisateur peut recevoir en référence un composant graphique et en modifier les propriétés. Ou encore, il peut simuler le lancement d'un événement de ce composant. Ceci pourrait permettre, par exemple, de développer une série de tests automatisés dans lesquels l'utilisateur n'a pas à intervenir. Cette capacité permet aussi d'interagir avec les systèmes ou de contrôler les états de l'EDD.

En plus de l'API offerte par l'EDD, toutes ses classes sont accessibles grâce à l'architecture de plugiciels d'Eclipse et à sa gestion des dépendances. Or, l'utilisateur est en parfait contrôle de l'EDD. Cela, bien qu'avantageux, comporte des risques. L'utilisateur pourrait créer des états qui pourraient causer des erreurs d'exécution.

APPLICATIONS INDUSTRIELLES

L'EDD a à ce jour permis de développer plusieurs IHMs à l'Agence Spatiale Canadienne. Entre autres, une IHM pour le robot CART. Un robot à deux bras possédant chacun six degrés de liberté qui est utilisé dans plusieurs projets de recherche.

L'EDD a aussi été utilisée pour la conception d'un prototype d'IHM de contrôle d'une perceuse destinée à la planète Mars dans le projet NORCAT.
L'EDD n'a pas servi qu'à développer des IHMs pour les stations de contrôle, il a, entre autres, servi au développement d'une IHM pour une application de tests automatisés d'IHM dans lequel le noyau fonctionnel simule les interactions de l'utilisateur dans le projet MARVIN.

Bien que l'expérience ait été constructive, certains problèmes ont été rencontrés par les développeurs qui ont utilisé l'EDD. Notamment, en l'absence de toute contrainte au niveau de la méthodologie de design, trop de fonctionnalité et d'intelligence ont été placés dans les interactions entre l'IHM et le noyau fonctionnel de MARVIN. La conséquence directe fut que les développeurs ont éprouvé des difficultés à localiser la source des erreurs rencontrés.

Plusieurs leçons ont été tirées de cette expérience. Par exemple, qu'il est préférable de limiter au minimum les interactions entre l'IHM et le noyau fonctionnel. Ensuite, qu'il faut s'assurer que l'API du noyau fonctionnel soit complète, simple et qu'elle favorise une intégration rapide de l'IHM et du noyau fonctionnel en offrant des méthodes dont les arguments nécessitent peu de traitement. Dans ce sens, plusieurs méthodes du noyau fonctionnel de MARVIN nécessitent que des objets soient construits et passés en paramètres aux méthodes. Il aurait été préférable que ces arguments soient de types simples et que le travail de construction des objets complexes soit effectué à même le noyau fonctionnel. Enfin, comme l'objectif de l'EDD est d'offrir un environnement de prototypage rapide, le noyau fonctionnel devrait être complètement testé et validé avant son intégration avec l'IHM.

CONCLUSION

Cet article décrit l'Environnement Dynamique de Développement (EDD) dans lequel l'utilisateur peut rapidement prototyper et tester des IHMs de façon dynamique sans devoir relancer la plateforme. Nous y avons décrit ses caractéristiques principales, son architecture, et nous avons discuté de son utilisation dans certains projets à l'Agence Spatiale Canadienne (ASC). Bien que l'EDD offre un environnement de prototypage flexible et riche en fonctionnalités, nous croyons que sa conception pourrait être poussée davantage pour offrir des techniques d'assemblage des comportements plus intuitives et plus intelligentes. Des techniques qui permettraient à l'utilisateur de visualiser de façon plus directe la relation entre ses tâches et l'IHM.

REMERCIEMENTS

Nous remercions Éric Dupuis, Martin Picard et Serge Ferland pour avoir encouragé le développement et l'utilisation de l'EDD dans le cadre de leurs projets.

BIBLIOGRAPHIE

1. Ali M.F., Perez-Quinones M.A. et Abrams M., Building Multi-Platform User Interfaces With UIML, in: A. Seffah & H. Javahery (eds.) Multiple User Interfaces: Engineering and Application Framework. John Wiley and Sons, 2003.

2. Jodoin, A., L'Archevêque, R., Allard, P., Desmarais, M., and Granger, L. 2006. Un environnement de développement dynamique qui intègre des composants graphiques actifs. In *Proceedings of the 18th international Conferenceof the Association Francophone D'interaction Homme-Machine* (Montreal, Canada, April 18 - 21, 2006). IHM '06, vol. 133. ACM, New York, NY, 293-294.

3. Myers B., Hudson S. E., Pausch R., Past, present, and future of user interface software tools, ACM Transactions on Computer-Human Interactions (TOCHI), ACM, 2000.

4. Singh G., Kok C. H., Hgan T. Y., Druid: a system for demonstrational rapid user interface development, Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology table, Snowbird, Utah, United States, 1991.

5. Sottet, J.-S., Calvary G, Favre J.-M., Ingénierie de l'Interaction Homme-Machine Dirigée par les Modèles, IDM'05, Premières Journées sur l'Ingénierie Dirigée par les Modèles, 2005.

6. Sukaviriya, P., Foley, D. J., Griffith, T.,A Second Generation User Interface Design Environment: The Model and The Runtime Architecture,
Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems, 1993.