Approche Logique de la Personnalisation dans les Environnements Informatiques pour l'Apprentissage Humain

Fabrice Popineau

université PARIS-SACLAY

Approche Logique de la Personnalisation dans les Environnements Informatiques pour l'Apprentissage Humain

Logical Approach of Personalization in Technology Enhanced Learning

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, ED Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique, Référent : CentraleSupélec

Thèse préparée au Laboratoire interdisciplinaire des sciences du numérique (Université Paris-Saclay, CNRS) sous la direction de Nicole BIDOIT, Professeure des Universités.

Thèse soutenue à Paris-Saclay, le 12 janvier 2023, par

Fabrice POPINEAU

THESE DE DOCTORAT

NNT : 2023UPASG001

Composition du jury

Anne VILNAT Professeur des Universités, Université Paris-Saclay	Présidente
Zahia GUESSOUM Maître de Conférences HDR, Université de Reims	Rapporteur
Sébastien GEORGE Professeur des Universités, Le Mans Université	Rapporteur
Bruno DEFUDE Professeur des Universités, Télécom-SudParis	Rapporteur
Marie-Hélène ABEL Professeur des Universités, Université de Technologie de Compiègne	Examinateur
Nicolas SABOURET Professeur des Universités, Université Paris-Saclay	Examinateur

Table des matières

Liste des figures

Remerciements

Je remercie tout d'abord Madame Nicole Bidoit qui a accepté de m'accompagner dans ce parcours de thèse de doctorat en VAE. Ses conseils m'ont été précieux, que ce soit pour l'orientation, la rédaction de ce mémoire ou la présentation des travaux.

Je tiens également à remercier le laboratoire LRI, devenu LISN, que j'ai rejoint de même que mes collègues de Supélec après la fusion avec l'École Centrale Paris. J'ai une pensée particulière pour Monsieur Yannis Manoussakis, directeur du LRI, disparu trop tôt, qui nous accueillis chaleureusement et qui a permis notre intégration dans le laboratoire.

Mes remerciements s'adressent également à mon employeur, CentraleSupélec, et en particulier à Monsieur Olivier Gicquel qui a soutenu et encouragé ma démarche de VAE.

Madame Yolaine Bourda occupe une place particulière dans ce mémoire. Elle m'a accepté dans son département Informatique de Supélec à une époque charnière de ma carrière. Elle m'a également accordé sa confiance pour co-encadrer avec elle plusieurs thèses de doctorat. Je lui adresse ma plus vive gratitude.

Je ne peux pas oublier les étudiants dont les travaux de thèse m'ont conduit jusqu'ici : Cédric, Georges, Youssef, Hiba pour les travaux présentés et ceux qui ont suivi : Jill-Jênn, Benoît qui ont tous soutenu brillamment. Les échanges que nous avons eu ont été très enrichissants et m'ont permis de progresser dans mes recherches. Merci à eux également.

Enfin, je dédie ce mémoire à ma famille, à mes parents, à mes amis et à tous ceux qui m'ont soutenu dans cette démarche.

Avant-propos

Nous présentons ici des travaux scientifiques dans le cadre d'une thèse de doctorat en *Validation des Acquis de l'Expérience* (VAE). Ces travaux ont été entrepris dans une carrière qui s'est déroulée majoritairement à Supélec, devenue CentraleSupélec depuis sa fusion avec l'École Centrale Paris ainsi qu'au LRI (Laboratoire de Recherche en Informatique, UMR8372) devenu LISN (Laboratoire Interdisciplinaire des Sciences du Numérique, UMR9015).

Notre production scientifique académique a commencé tardivement dans cette carrière. En tant qu'école d'ingénieurs, Supélec était essentiellement tournée vers l'industrie et ses partenaires industriels. La production scientifique académique n'est devenue un objectif pour Supélec que lors d'un mouvement qui s'est opéré au milieu des années 2000.

Les travaux présentés sont sélectionnés pour constituer un choix thématiquement cohérent dans une période qui s'étend jusqu'en 2017. Ce choix consiste en une approche classique de l'intelligence artificielle, fondée essentiellement sur la logique, appliquée à la conception d'environnements informatiques pour l'apprentissage humain (EIAH) et plus spécifiquement à l'adaptation de ceux ci à l'apprenant. Nous réservons la suite de nos travaux pour soumettre éventuellement et ultérieurement un dossier d'habilitation à diriger des recherches.

1 - Introduction

1.1 . Environnements Informatiques pour l'Apprentissage Humain

La diffusion de la connaissance et probablement la cognition humaine par la même occasion ont connu plusieurs révolutions technologiques : l'écriture, l'imprimerie, la diffusion par radio et télévision. L'informatique est la dernière révolution technologique spectaculaire dans le traitement de l'information, autant par la puissance de calcul qu'elle met à disposition que par le fait qu'elle permet de relier les humains sur toute la planète. Dans le domaine de la diffusion de la connaissance, elle a ouvert un nouveau champ de par l'interaction qu'elle permet avec un apprenant, que ce soit en présence ou à distance.

Depuis plusieurs décennies, l'ordinateur a été envisagé comme un medium pour délivrer des connaissances à un apprenant. On trouve dans [46], [26] des recensements historiques, prenant en compte l'interaction avec l'ordinateur, en présence ou à distance au moyen des premiers espaces de discussion [1]. Lorsque des leçons sont diffusées par un canal unique tel que la télévision, tous les spectateurs reçoivent le même contenu. L'ordinateur permet l'interaction : l'apprenant peut répondre à des questions, choisir de passer à la leçon suivante, ou toute autre option proposée. Par le fait que l'apprenant est identifié, la machine peut également décider du contenu le plus approprié à délivrer à tout moment et des modalités pour le faire. Les possibilités sont colossales.

Le terme EIAH pour *Environnement Informatique pour l'Apprentissage Humain* a été forgé à la fin des années 1990. L'équivalent anglo-saxon est TEL pour *Technology Enhanced Learning*. La couverture du champ scientifique des EIAH est donnée dans [41] :

> Le champ scientifique des EIAH correspond aux travaux focalisés sur les environnements informatiques dont la finalité explicite est de susciter et d'accompagner l'apprentissage humain, c'est-à-dire la construction de connaissances chez un apprenant. Ce type d'environnement mobilise des agents humains (élève, enseignant, tuteur) et artificiels (agents informatiques, qui peuvent eux aussi tenir différents rôles) et leur offre des situations d'interaction, localement ou à travers les réseaux informatiques, ainsi que des conditions d'accès à des ressources formatives (humaines et/ou médiatisées), ici encore locales ou distribuées. Il comprend les questions scientifiques et technologiques soulevées par la conception, la réalisation et l'évaluation de ces environnements, ainsi que la compréhension de leurs impacts sur la connaissance, la personne et la société.

L'espoir porté par les EIAH réside dans le fait d'enseigner le mieux possible au plus grand nombre, en particulier par une adaptation du rythme et de la progression à chaque apprenant [53], [37]. L'ère du « one size fits all » en éducation serait enfin révolue grâce à l'outil informatique, soit par un support aux enseignants pour la gestion individualisée des apprenants, soit par la construction d'EIAH entièrement automatisés.

Il a été démontré que de tels systèmes sont utiles sur le plan éducatif dans divers domaines, comme l'apprentissage des mathématiques et sont déjà utilisés par un très grand nombre d'étudiants chaque année. Le potentiel de ces systèmes pour révolutionner l'éducation, en offrant un retour d'information à tout moment et en tout lieu, est jugé majeur [47].

1.2 . Problématiques étudiées

Le cadre des EIAH est vaste et ouvre la porte à de très nombreuses questions de recherche. Les travaux présentés ici sont orientés vers le problème de la personnalisation des parcours

1. *babillards électroniques* en France.

dans un EIAH du point de vue de l'adaptation du contenu.

Les techniques de personnalisation sont multiples : un EIAH et un système de recommandation proposent tous les deux des contenus personnalisés, mais peuvent nécessiter de procéder de façon différente, parce que les situations et les objectifs sont différents. La technique de recommandation collaborative s'applique assez simplement pour l'achat de produits en ligne, biens de consommation, musique, films, etc. Les recommandations sont faiblement couplées et ne sont pas obligatoirement suivies d'actes d'achat. Si elles ne sont pas diversifiées, elles tendent à conforter, voir à enfermer, les utilisateurs dans leurs choix. La construction d'un parcours personnalisé pour un apprenant doit répondre à un certain nombre de contraintes : notions intermédiaires obligatoires, prérequis entre les notions, niveau des ressources utilisées, ...La construction du parcours personnalisé dans un EIAH oblige l'étudiant à suivre la ressource proposée. Il faut donc être certain que cette ressource soit adaptée. Pour cette raison, on peut avoir envie de se tourner vers une approche « expert » de la construction de ce parcours à l'aide de règles ou d'un formalisme explicite. [2]

Il n'existe pas de technique générique d'expression de la personnalisation, comme nous avons pu le montrer dans [15]. Dans le cadre des EIAH, nous pouvons toutefois espérer définir une technique générique d'expression d'un parcours personnalisé pour l'apprenant.

Parmi toutes les options possibles offertes par le cadre de recherche des EIAH, nous sommes parti d'une vision qui trouve son origine dans la représentation des documents électroniques et de leur structuration quand elle est appliquée aux manuels scolaires.

La notion de document structuré a été popularisée avec l'émergence de standards tels que SGML [25]. Le balisage permet une séparation de la forme et du fond, ainsi qu'un étiquetage sémantique des éléments constitutifs du document. Cette structuration des documents s'est développée plus avant avec l'introduction de XML mais aussi des langages de composition comme LaTeX.

Parallèlement, nombre de manuels scolaires, surtout en sciences, sont architecturés selon un plan qui permet plusieurs parcours de lecture selon le temps disponible et les objectifs visés par le lecteur ou l'enseignant (figure 1.1).

En combinant ces deux observations, on en déduit qu'il est possible de construire un premier système personnalisé sur la base :

— d'un découpage structurel d'un document en contenus unitaires : chapitres, sections ou plus généralement grains pédagogiques,

— d'une planification personnalisée du parcours.

À l'image de ce qui s'est pratiqué dans le cadre des systèmes experts, les systèmes d'EIAH ont implémenté ce type de personnalisation le plus souvent à l'aide de règles, sans capacité de contrôle sur le résultat obtenu. Est-ce que l'apprenant s'est bien vu offrir tous les contenus requis par son objectif d'apprentissage ? Est-il arrivé à l'objectif fixé ? Le système a-t-il toujours présenté les prérequis nécessaires avant chaque unité ? Ces systèmes de règles s'apparentent plus à un langage de programmation qu'à un système de spécification logique qui permettrait de prouver des propriétés du système réalisé. L'écriture d'un système d'EIAH nécessite une programmation ad-hoc avec pour conséquence une extensibilité et une réutilisabilité faibles.

Nous avons vu ici l'opportunité d'introduire une modélisation sous-jacente et un contrôle de la personnalisation à l'aide d'outils de logique formelle permettant une meilleure garantie du résultat obtenu.

Plus spécifiquement, le découpage d'un document tel qu'un manuel scolaire selon sa structure résulte en un graphe de contenus reliés par une relation de prérequis. Le découpage automatique d'un polycopié de cours à partir du source LaTeX et l'introduction de différents parcours de lecture (figure 1.2) ont été réalisés lors du stage de Claudiu Dinca [1] à Supélec. La conversion LaTeX vers XML a été effectuée grâce à TeX4ht et la présentation à l'apprenant

2. Les techniques de recommandation classiques sont appliquées dans le cadre des MOOCs par exemple, mais pour recommander un MOOC dans son ensemble, comme un produit.

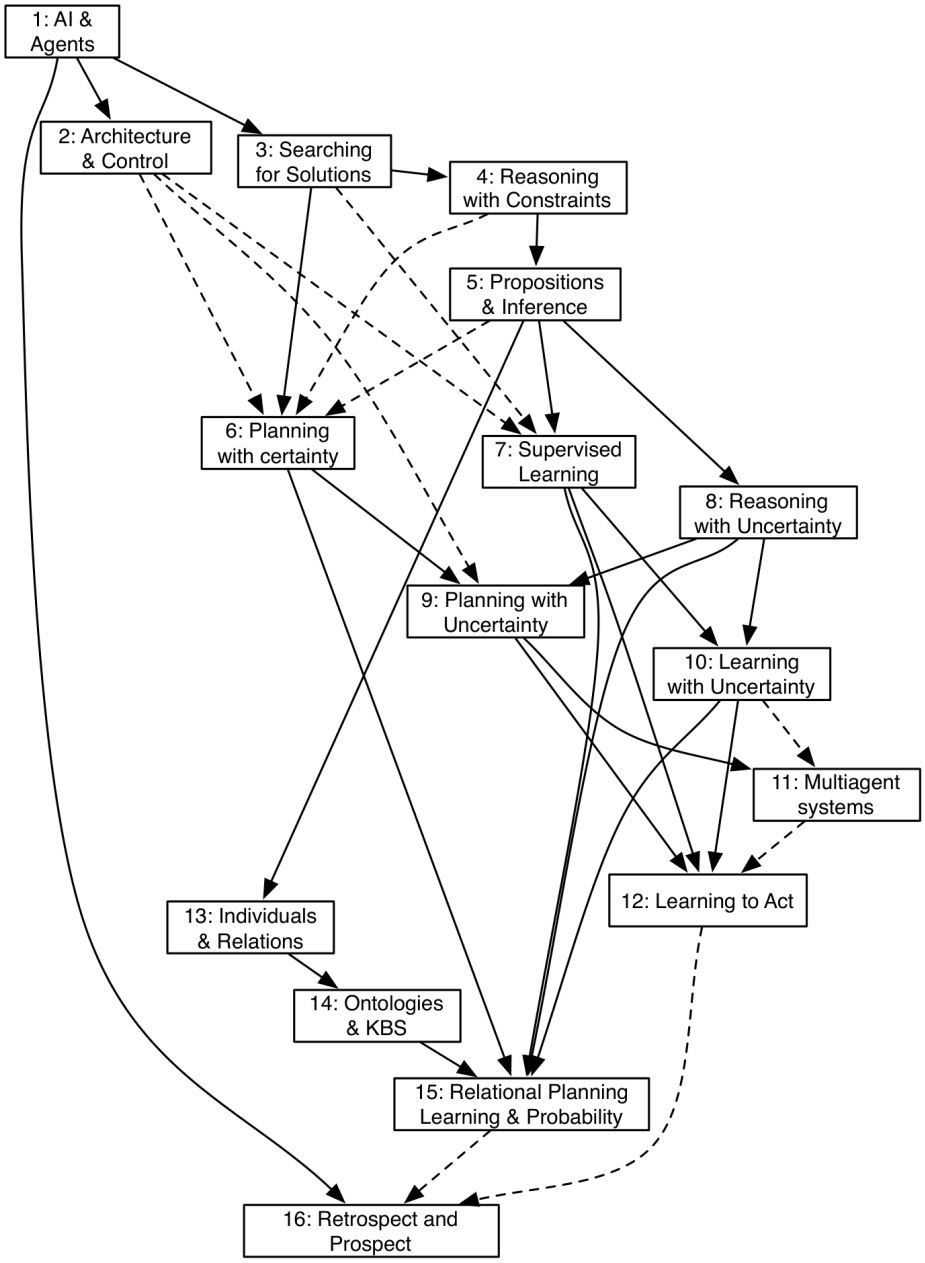


Figure 1.1 – Exemple de relation de dépendance entre les chapitres de l'ouvrage [55].

était prise en charge par le serveur Web CL-HTTP qui permettait déjà à cette époque de servir dynamiquement un contenu [3].

Prétendre traiter les EIAH en partant à chaque fois d'un seul manuel serait évidemment très restrictif. Cependant il est facile d'étendre cette vision à celle d'un graphe de grains pédagogiques, issus potentiellement de différentes sources et non d'un seul document, reliés par une relation de prérequis, mais éventuellement aussi par d'autres relations. Chacun de ces grains est décrit par un ensemble de métadonnées qui en formalisent le contenu. Par exemple, une définition d'un concept sera décrite par le terme défini, mais également par les autres concepts immédiatement nécessaires à la compréhension de cette définition. De la même façon la démonstration d'un théorème fera référence au grain pédagogique du théorème lui-même, ainsi qu'aux concepts nécessaires à la compréhension de la démonstration. On peut également étendre cette formalisation à des exercices. Nous ne nous attacherons pas

3. Et ce bien avant le Web 2.0 et Javascript ! CL-HTTP était un serveur Web écrit en Common Lisp. Il a été utilisé en particulier par Peter Brusilovsky pour ELM-ART [31] et comme serveur Web de la Maison Blanche.

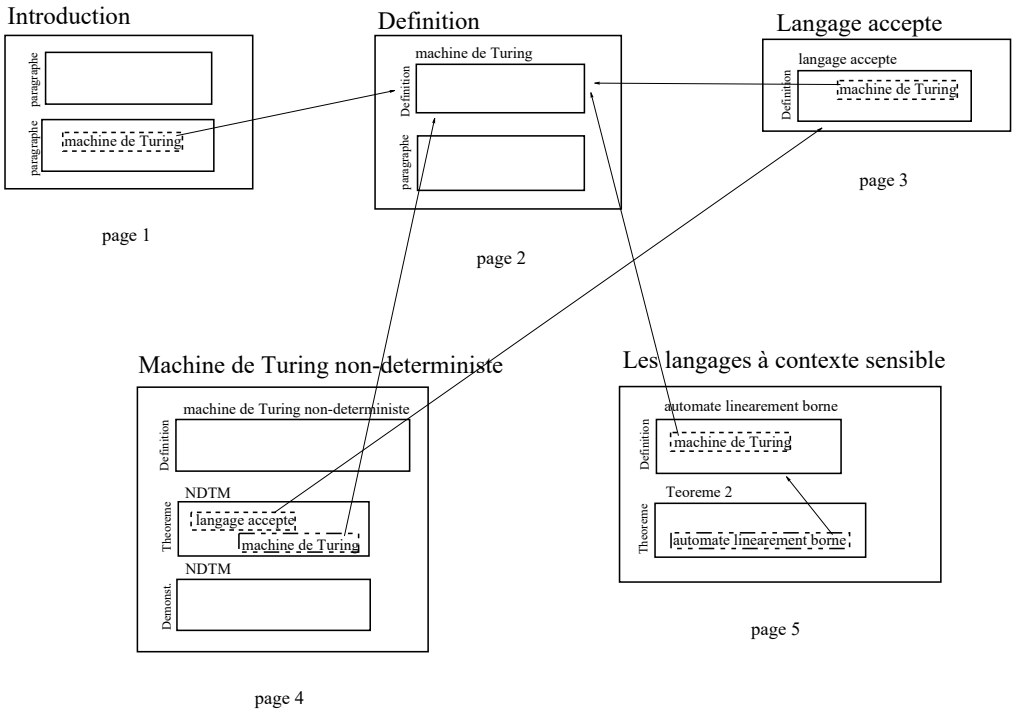


Figure 1.2 – Extraction automatique de contenu à partir d'un polycopié de cours.

ici à définir la représentation des connaissances elle-même : c'est un travail à part entière. Les EIAH ont besoin d'interopérabilité pour assurer une pérennité aux contenus produits. C'est entre autres l'objet du sous-comité ISO/IEC JTC1/SC36 de définir des standards de description pour ces grains pédagogiques. Nous supposerons seulement la disponibilité d'un ensemble de grains pédagogiques avec leurs métadonnées et les relations qui les lient.

La caractéristique importante de cette formalisation est de reposer sur un graphe (potentiellement multi-graphe) sous-jacent. Pour la suite, notre intérêt se portera sur la partie raisonnement attachée à un EIAH : quel est le contenu à fournir à l'apprenant ? Quelle action entreprendre pour lui assurer la meilleure expérience d'apprentissage ?

La relation la plus importante entre les grains pédagogiques est la relation de prérequis. La construction d'un parcours à travers un graphe de prérequis peut être étayée par une approche logique telle que celle utilisée dans les problèmes de planification. Ici, le parcours devient la solution à une requête logique : comment amener l'apprenant d'un point d'entrée à un point de sortie, en tenant éventuellement compte de contraintes ?

Au chapitre 2, nous proposons une synthèse des travaux publiés [19]–[24]. Nous montrerons comment formaliser la planification de parcours dans le cadre du calcul des situations [49]. Nous nous focaliserons sur les aspects de raisonnement, ou de calcul des contenus les plus adaptés à un apprenant. Associé à la programmation logique, nous verrons qu'il constitue une base solide pour les EIAH.

Grâce au Web2.0, les « cours en ligne ouverts et massifs » ou MOOCs (pour *Massive Online Open Courses*, voir section 4.1) sont devenus un phénomène mondial à part entière. Pour aller plus loin dans la construction d'une expérience pédagogique personnalisée, nous avons souhaité nous rapprocher de l'approche classique en intelligence artificielle de l'agent rationnel (figure 1.3) telle qu'elle est décrite dans [65] [4]. Il s'agit donc de considérer que c'est un agent qui délivre les contenus ou qui pilote le MOOC. Cette approche a pour particularité essentielle de tracer une ligne de séparation claire entre l'agent et son environnement et de nécessiter de formaliser les échanges à l'aide d'actions et de captation d'événements ou *sensing*. L'objet est d'implémenter la fonction agent qui détermine les actions entreprises par

4. Il s'agit volontairement d'une édition antérieure à l'édition courante de 2022.

l'agent en fonction de ce qu'il perçoit avec ses capteurs. C'est la raison qui nous a conduit à nous intéresser :

— à la formalisation de l'interaction avec une application web d'une part,

— à la personnalisation et à la personnification d'agents logiques d'autre part.

Au chapitre 3, nous proposons une synthèse des travaux publiés [12]–[15]. Nous préciserons les notions d'adaptation et nous orienterons les travaux vers les agents virtuels interactifs qui offrent un cadre général pour l'adaptation de contenu, avec la possibilité de donner une personnalité à un tel agent. Nous montrerons comment certains aspects de personnalisation et de personnification d'un agent peuvent être implémentés par transformation de programmes logiques.

Enfin, la mise en musique ne serait pas complète sans un regard sur des aspects complémentaires :

— de l'intégration d'un agent virtuel interactif dans un MOOC,

— de recommandation qui peuvent également être vus comme une capacité supplémentaire d'un agent, et donc un moyen d'action vis à vis de l'apprenant.

Au chapitre 4, nous présenterons une mise en oeuvre dans un cadre réel qui est celui de la plateforme OpenEdX et nous nous intéresserons également à la recommandation personnalisée de contenus pédagogiques dans le cadre d'un MOOC, qui est une forme d'adaptation de contenu. Ce chapitre sera constitué d'une synthèse des publication [8]–[11].

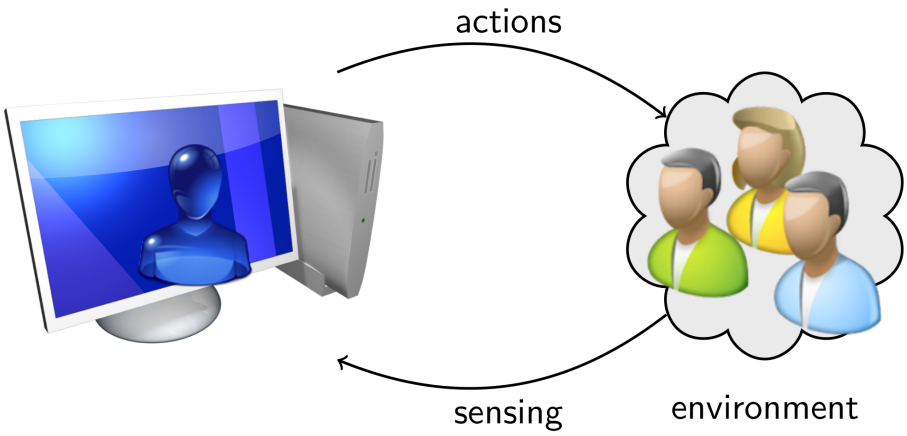


Figure 1.3 – Architecture IA reposant sur un agent.

En résumé, notre contribution est essentiellement la vision d'un système EIAH piloté par un agent :

— fondé sur une approche logique,

— doté d'un comportement rationnel,

— qui s'adapte à des profils différents (personnalisation de l'expérience pédagogique),

— qui peut prendre différentes postures (personnification de l'agent),

— que l'on peut mettre en oeuvre dans le cadre d'une plateforme MOOC

  — pour servir des contenus adaptés sur la base d'un objectif,

  — pour recommander des contenus,

  — pour entretenir l'engagement.

Bien sûr ces travaux se sont déroulés sur un temps assez long et l'objectif est très ambitieux. Des jalons ont été posés, mais il reste bien des espaces à combler. Un chapitre de conclusion 5 évoquera quelques pistes et quelques-unes des grandes difficultés.

1.3 . Productions et encadrements

Les travaux présentés ont été accomplis lors de l'encadrement de stages de master et du co-encadrement de plusieurs thèses. Ils ont donné lieu à plusieurs publications dont la liste exhaustive est donnée ci-dessous et sera reprise par chapitre. Des articles choisis et représentatifs sont reproduits en annexe B.

– Encadrements de stages de Master ou de niveau équivalent :

[1] C. Dinca, « Construction d'un Site Web Adaptatif », Universitatea Politehnica Bucuresti, Stage de fin d'études, sept. 2000.

[2] D. Arfire, "Graphical Knowledge Representation", Universitatea Politehnica Bucuresti, Stage de fin d'études, Jun. 2006.

[3] H. Hawes, « Personnalisation et Personnification des Tuteurs Intelligents », Université de la Manouba, École Nationale des Sciences de l'Informatique, Stage de fin d'études, sept. 2015.

– Co-encadrements de thèses :

[4] C. Jacquiot, « Modélisation logique et générique des systèmes d'hypermédias adaptatifs », Thèse de doctorat, Paris 11, 1er jan. 2006, Prix de Thèse de la Fondation Jean-Luc Lagardère.

[5] G. Dubus, « Transformation de programmes logiques : application à la personnalisation et à la personnification d'agents. », Thèse de doctorat, Supélec, 4 sept. 2014.

[6] H. Hajri, « Personnalisation des MOOC par la réutilisation de Ressources Éducatives Libres », Thèse de doctorat, Université Paris-Saclay (ComUE), 8 juin 2018.

– Publications dans des conférences avec comité de lecture :

[7] Y. Bourda et al., « Tuteurs intelligents : boucler la boucle », in *PFIA 2018 - Journée « IA pour l'éducation »*, juill. 2018, page 4.

[8] H. Hajri, Y. Bourda, and F. Popineau, "A System to Recommend Open Educational Resources during an Online Course:" in *Proceedings of the 10th International Conference on Computer Supported Education*, Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, Mar. 2018, pages 99–109. doi: 10/gmfn5w.

[9] H. Hajri, Y. Bourda, and F. Popineau, "Personalized Recommendation of Open Educational Resources in MOOCs", in *Computer Supported Education - 10th International Conference, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Revised Selected Papers*, series Communications in Computer and Information Science, B. M. McLaren, R. Reilly, S. Zvacek, and J. Uhomoibhi, Eds., volume 1022, Cham: Springer International Publishing, 2018, pages 166–190. doi: 10.1007/978-3-030-21151-6_9.

[10] H. Hajri, Y. Bourda, and F. Popineau, "MORS: A System for Recommending OERs in a MOOC", in *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, Jul. 2017, pages 50–52. doi: 10/ghfbvb.

[11] H. Hajri, Y. Bourda, and F. Popineau, "Querying Repositories of OER Descriptions: The Challenge of Educational Metadata Schemas Diversity", in *Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015*, series Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015, volume LNCS, Toledo, Spain: Springer, Sep. 2015, pages 582–586. doi: 10/gmfpf6.

[12] F. Popineau, G. Dubus, and Y. Bourda, "Personalization and personification: A constructive approach based on parametric agents", in *Intelligent Virtual Agents - 14th International Conference, IVA 2014*, T. W. Bickmore, S. Marsella, and C. L. Sidner, Eds., series Lecture Notes in Computer Science, volume 8637, Boston, MA, USA: Springer, Aug. 2014, pages 339–344. doi: 10.1007/978-3-319-09767-1_44.

[13] G. Dubus, F. Popineau, Y. Bourda, and J.-P. Sansonnet, "Parametric Reasoning Agents Extend the Control over Standard Behaviors", in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 2, Atlanta, GA, USA: IEEE, Nov. 2013, pages 163–170. DOI: 10.1109/WI-IAT.2013.105.

[14] G. Dubus, F. Popineau, and Y. Bourda, "A Formal Approach to Personalization", in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, Boca Raton, FL, USA: IEEE Computer Society, Nov. 2011, pages 233–238. DOI: 10/fzhff2.

[15] G. Dubus, F. Popineau, and Y. Bourda, "Situation calculus and personalized web systems", in *2011 11th International Conference on Intelligent Systems Design and Applications*, Córdoba, Spain: IEEE, Nov. 2011, pages 569–574. DOI: 10/fzrtsq.

[16] N. Zemirline, Y. Bourda, C. Reynaud, and F. Popineau, "MESAM: A Protégé Plug-in for the Specialization of Models", in *11th International Protégé Conference*, Amsterdam, Netherlands, Jun. 2009, 3 pages.

[17] N. Zemirline, Y. Bourda, C. Reynaud, and F. Popineau, "Assisting in Reuse of Adaptive Hypermedia Creator's Models", in *Adaptive Hypermedia and Adaptive Web-Based Systems*, W. Nejdl, J. Kay, P. Pu, and E. Herder, Eds., series Lecture Notes in Computer Science, volume 5149, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pages 357–360. DOI: 10.1007/978-3-540-70987-9_53.

[18] N. Zemirline, C. Reynaud, Y. Bourda, and F. Popineau, "A pattern and rule-based approach for reusing adaptive hypermedia creator's models", in *Proceedings of 16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns*, A. Gangemi and J. Euzenat, Eds., series Lecture Notes in Computer Science, volume 5268, Acitrezza, Catania, Italie: Springer-Verlag, 2008, pages 17–31. DOI: 10.1007/978-3-540-87696-0_5.

[19] C. Jacquiot, Y. Bourda, F. Popineau, A. Delteil, and C. Reynaud, "GLAM: A generic layered adaptation model for adaptive hypermedia systems", in *Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference, AH 2006, Dublin, Ireland, June 21-23, 2006, Proceedings*, H. A. V. Wade and B. Smyth, Eds., series LNCS, volume 4018, Springer-Verlag, Jun. 2006, pages 131–140.

[20] C. Jacquiot, Y. Bourda, and F. Popineau, "GEAHS: A Generic Educational Adaptive Hypermedia System", presented at the EdMedia + Innovate Learning, Association for the Advancement of Computing in Education (AACE), 2004, pages 571–578.

[21] C. Jacquiot, Y. Bourda, and F. Popineau, "GEAHS: A Generic Educational Adaptive Hypermedia System Based on Situation Calculus", in *Adaptive Hypermedia and Adaptive Web-Based Systems, Third International Conference, AH 2004, Eindhoven, the Netherlands, August 23-26, 2004, Proceedings*, P. D. Bra and W. Nejdl, Eds., series Lecture Notes in Computer Science, volume 3137, Springer, 2004, pages 413–416. DOI: 10.1007/978-3-540-27780-4_63.

[22] C. Jacquiot, Y. Bourda, and F. Popineau, "Reusability in GEAHS", in *Engineering Advanced Web Applications: Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering (ICWE 2004), Munich, Germany, 28-30 July, 2004*, M. Matera and S. Comai, Eds., Munich, Germany: Rinton Press, 2004, pages 199–209.

[23] C. Jacquiot, Y. Bourda, and F. Popineau, "SEWAGO : A generic adaptive educational hypermedia system.", in *HYPERTEXT '03: Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia*, Nottingham, UK, Aug. 2003.

[24] F. Popineau, Y. Bourda, and B.-L. Doan, "Generating adaptive hypermedia with Golog and conceptual graphs", in *Proceedings 3rd IEEE International Conference on Advanced Technologies*, Athens, Greece: IEEE Computer Society, Jul. 2003, pages 463–466. DOI: 10/cq9bcv.

2 - Planification des apprentissages

2.1 . Hypermédias adaptatifs

Parmi tous les EIAH envisageables, ceux construits sur une interface Web ont été particulièrement populaires. L'un des tous premiers EIAH de ce type a été ELM-ART [31] dédié à l'apprentissage de la programmation en Lisp. Avec son interface innovante, ELM-ART transforme l'EIAH en un manuel hypertexte intelligent. ELM-ART « connaît » le contenu qu'il présente à ses utilisateurs. Chaque exemple présenté devient une « expérience vivante » : les exemples sont générés et exécutés dynamiquement. Dans le contexte des EIAH, nous utiliserons indifféremment les termes « utilisateur » et « apprenant » par la suite.

Peter Brusilovsky [30] définit les *hypermédias adaptatifs* comme se situant à la croisée de l'hypermédia et de la modélisation de l'utilisateur. Une des limites des applications hypermédias « statiques » traditionnelles est qu'elles fournissent les mêmes pages avec le même contenu et le même ensemble de liens à tous les utilisateurs. On trouve ici les prémisses de la personnalisation que nous définirons plus précisément en section 3.2. Si la population d'utilisateurs est relativement diverse, un système traditionnel souffrira d'une incapacité à satisfaire tout le monde, par la nécessité de présenter trop d'information dont une grande partie ne sera pas adaptée. Le « one size fits all » trouve ici ses limites. Un système hypermédia éducatif traditionnel présentera la même explication statique et suggérera la même page suivante ou le même ensemble de pages suivantes à des étudiants ayant des profils très différents, dont les objectifs pédagogiques et les connaissances sur le sujet sont très différents.

Nous nous placerons pour la suite de ce chapitre dans le cadre des hypermédias adaptatifs ou HA, en cherchant à fournir des solutions à l'adaptation de parcours : la présentation d'un contenu personnalisé et adapté à chaque utilisateur. Il est à noter ici qu'une distinction a prévalu entre adaptation de contenu, adaptation de navigation et adaptation de présentation. Traditionnellement, l'adaptation de contenu recouvre le fait qu'un contenu est composé de plusieurs parties qui peuvent être affichées ou non. Il en est de même des liens de navigation pour l'adaptation de navigation. La présentation peut faire l'objet d'une attention particulière pour des besoins tels que l'accessibilité. Dans le cadre qui nous occupe, nous considèrerons que chaque page Web permet un certain nombre d'actions de navigation : passage à une étape suivante à l'intérieur de l'EIAH, ouverture d'un lien externe, retour arrière, soumission de formulaire ... nous préférerons parler d'adaptation de parcours que de considérer une séparation stricte entre contenu et navigation. L'adaptation de présentation pourra être prise en charge dans le profil de l'apprenant de façon homogène, comme un élément de contexte général.

Comme annoncé en introduction, nous souhaitons montrer l'intérêt d'une solution fondée sur la programmation logique et une approche IA traditionnelle sous la forme d'un agent rationnel.

Les environnements existants à l'époque des travaux présentés ici reposent sur des systèmes à base de règles procédurales [72], [33], [29]. Ils sont écrits de façon ad-hoc, ce qui rend difficile ou impossible de prouver une prise en charge correcte des besoins de chaque apprenant. Un système d'adaptation du parcours écrit de façon ad-hoc possède deux inconvénients majeurs :

1. il n'est pas réutilisable ;
2. il est difficile à faire évoluer ;
3. il est difficile ou impossible de prouver les propriétés d'un tel système.

Pour remédier aux inconvénients de l'écriture de règles ad-hoc, nous proposons de construire un système d'HA reposant sur une base purement logique et plus précisément sur le calcul des situations. En effet, l'adaptation de contenu est à rapprocher de la planification

classique et une modélisation très formelle de ce problème en est le calcul des situations [48]. Dans ce cadre formel, l'expression de l'adaptation pourra se faire de façon totalement abstraite : la définition du domaine lui-même n'interviendra pas. Il en résultera la réutilisabilité, ainsi que la capacité de contrôle sur le résultat obtenu.

2.2 . Agents, calcul des situations et Golog

Le calcul des situations a été introduit par Reiter [59] sur la base d'une proposition antérieure de McCarthy et Hayes [51].

Ce calcul repose sur trois composantes :

1. les fluents qui décrivent les propriétés du monde,
2. les actions qui permettent d'agir sur une situation pour en produire une nouvelle,
3. les situations elles-mêmes, avec une situation particulière S_0 qui constitue la situation initiale.

La valeur d'un fluent dépend de la situation particulière observée. Les fluents peuvent être relationnels dans le cas de prédicats, ou bien fonctionnels dans le cas de fonction. Un fluent relationnel aura une valeur booléenne alors qu'un fluent fonctionnel aura une valeur appartenant à un domaine.

Supposons que l'on dispose de ressources pédagogiques $R = \{r_1, \dots, r_n\}$. Chacune de ces ressources délivre une composante de connaissance (CC, ou « knowledge component », KC en anglais). Informellement, nous supposons qu'après avoir travaillé la ressource, nous avons acquis la compétence élémentaire que la ressource délivre. En suivant [44] :

> « Nous définissons une composante de connaissance (CC) comme étant une unité acquise d'une fonction cognitive ou une structure qui peut être inférée de la performance d'un apprenant sur un ensemble de tâches liées entre elles. [...] En pratique, nous utilisons la notion de composante de connaissance pour généraliser des termes qui décrivent des éléments de cognition ou de connaissance [...] mais également des termes de tous les jours comme concept, principe, fait ou compétence [...]. »

Nos ressources délivrent donc des composantes de connaissance $C = \{c_1, \dots, c_p\}$. Chaque ressource pédagogique délivre une seule CC [1] et une même CC peut-être délivrée par plusieurs ressources pédagogiques. Un apprenant commencera dans une situation initiale S_0 : il ne maitrisera aucune des composantes de connaissances délivrées par l'HA. Un fluent relationnel $\text{seen}(c, s)$ indiquera que la CC c aura été vue dans la situation s. Un fluent fonctionnel $\text{KC}(r, s) = c$ donnera la CC c délivrée par la ressource r. Cette valeur sera de fait indépendante de la situation s. Un fluent fonctionnel $\text{prereq}(c, s) = \{c_{c_1}, \dots, c_{c_n}\}$ donnera la liste $\{c_{c_1}, \dots, c_{c_n}\}$ des CC requises pour aborder la CC c. Un fluent fonctionnel $\text{level}(c, s) = l$ donnera le niveau $l \in \{\text{beginner}, \text{normal}, \text{advanced}\}$ de l'étudiant sur la CC c dans l'état s. L'apprenant terminera dans une situation finale s_f où il aura vu un certain nombre de composantes de connaissances : $\{c \in C \mid seen(c, s_f)\}$. Nous chercherons à faire en sorte qu'un certain sous-ensemble de CC cibles de C aient été vues.

Les actions sont régies par un prédicat et une fonction spécifiques :

Poss(a, s) est un prédicat qui définit quand l'action a est *possible* dans la situation s,

do(a, s) est une fonction qui désigne le résultat de l'exécution de l'action a dans la situation s.

En continuant notre exemple, nous pouvons considérer une action $\text{show}(r)$ qui consiste à présenter la ressource r. La définition de $\text{show}(r)$ passerait par :

$$\text{Poss}(\text{show}(r), s) \equiv \forall c \in \text{prereq}(\text{KC}(r, s))\ \text{seen}(c, s)$$

1. Ce n'est bien sûr pas le cas en général, mais cette restriction est prise ici pour des raisons de simplicité.

autrement dit : il faut que toutes les CC requises pour aborder r aient été vues pour présenter r.

Une situation s est définie par une « chronique », c'est-à-dire l'historique de l'application successive d'actions élémentaires à une situation de départ S_0. Ainsi la situation $s = \text{do}(\text{show}(r_3), \text{do}(\text{show}(r_2), \text{do}(\text{show}(r_1), S_0)))$ désignera la situation où l'on aura présenté successivement les ressources r_1, puis r_2, puis r_3 à partir de la situation S_0. On pourra bien sur abréger les notations des situations en utilisant $s_1 = \text{do}(\text{show}(r_1), S_0)$, $s_2 = \text{do}(\text{show}(r_2), s_1)$, etc.

À ce stade, il manque encore les règles qui permettent de définir le calcul proprement dit – les règles d'enchaînement – des situations. Reiter a calqué l'axiomatisation du calcul des situations sur l'arithmétique de Peano pour les entiers naturels. Les axiomes de base du calcul des situations sont au nombre de quatre :

$$\text{do}(a_1, s_1) = \text{do}(a_2, s_2) \Rightarrow a_1 = a_2 \wedge s_1 = s_2 \tag{2.1}$$

$$\forall P\ P(S_0) \Rightarrow (\forall a, s\ P(s) \wedge P(\text{do}(a, s))) \Rightarrow \forall s\ P(s) \tag{2.2}$$

$$\neg s \sqsubset S_0 \tag{2.3}$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s' \tag{2.4}$$

Le premier axiome (2.1) est un axiome des noms uniques pour les situations.

Le second axiome (2.2) est un axiome du second ordre analogue à l'axiome d'induction de l'arithmétique de Peano. Il permet de propager les propriétés (fluents) au travers des actions.

La relation $\sqsubset$ est une relation d'ordre sur les situations : $s \sqsubset s'$ signifie que la chronique qui définit la situation s' est un surensemble de la chronique qui définit la situation s, la situation s' peut être obtenue en appliquant une séquence d'actions à partir de la situation s. La notation $\sqsubseteq$ est une abréviation : $s \sqsubseteq s' \equiv s = s' \vee s \sqsubset s'$.

Le troisième axiome (2.3) indique que S_0 est bien la situation initiale et le quatrième axiome (2.4) formalise la relation d'ordre sur les situations.

Ces quatre axiomes sont indépendants du domaine. Ils forment une base pour une théorie du calcul des situations et seront dénotés par Σ.

Le *frame-problem*, ou problème de la persistance, a longtemps été un obstacle à toute entreprise IA (symbolique) [42], [57]. Ce problème provient du fait que si l'on se contente de stipuler les effets des actions, il devient difficile de prédire la valeur d'un fluent après une longue série d'actions. Beaucoup de fluents ne changent pas et chaque action ne change la valeur que de peu de fluents. La question qui se pose est donc : comment déterminer rapidement ce qui persiste au travers d'une action ? Comment éviter de calculer l'ensemble de l'état pour connaître la valeur d'un fluent dans cet état ? Reiter résout ce problème à l'aide d'axiomes de l'état-successeur [59]. Le principe consiste à introduire, pour chaque fluent, un axiome qui énumère les actions qui changent la valeur de ce fluent. Pour un fluent relationnel F, l'axiome sera de la forme :

$$Poss(a, s) \Rightarrow \left[F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s))\right] \tag{2.5}$$

où $\gamma_F^+(\vec{x}, a, s)$ (respectivement $\gamma_F^-(\vec{x}, a, s)$) dénote le fait que l'action fait passer la valeur du fluent F à vrai (respectivement : faux). Ces axiomes garantissent que les fluents qui ne sont pas affectés par une action ne changent pas de valeur pendant cette action.

Par exemple, si l'on considère le fluent $\text{seen}(c, s)$ pour indiquer que la KC c a été vue dans l'état s par l'apprenant et l'action $\text{show}(r)$ qui présente la ressource r à l'apprenant, alors un axiome successeur d'état pour seen sera :

$$\text{seen}(c, \text{do}(a, s)) \equiv \exists r\ (a = \text{show}(r) \wedge \text{KC}(r) = c) \vee (\text{seen}(c, s)),$$

autrement dit : la KC c aura été vue dans l'état résultat de l'action a à partir de la situation s si et seulement si a est $\text{show}(r)$ et r illustre la KC c, ou alors la KC c avait déjà été vue dans la situation s.

Formellement, une théorie du calcul des situations sur un domaine $\mathscr{D}$ sera définie sur la base de Σ par l'adjonction d'ensembles d'axiomes supplémentaires spécifiant le domaine et le comportement des actions et comportera donc :

— Σ, les axiomes de base du calcul des situations indépendants du domaine ;

— $\mathscr{D}_{SS}$, les axiomes successeurs d'état, un par fluent F du domaine ;

— $\mathscr{D}_P$ les axiomes de préconditions, un par action a du domaine ; ils servent à définir $\text{Poss}(a, s)$;

— $\mathscr{D}_{S_0}$, les axiomes définissant la situation initiale $\mathscr{D}_{S_0}$

— $\mathscr{D}_{una}$, les axiomes des noms uniques pour les actions ;

— $\mathscr{D}_{dca}$, les axiomes de clôture du domaine pour les actions .

On trouvera une axiomatisation complète du calcul des situations dans [60], [49].

Ce calcul des situations a été opérationnalisé en un langage Golog par l'équipe de robotique cognitive de l'Université de Toronto [50]. Golog fournit une implémentation du calcul des situations qui permet d'assembler des actions primitives, définies dans le calcul des situations, en actions complexes qui constituent collectivement un programme δ. Golog est naturellement implémenté en Prolog. Les programmes Golog sont définis de façon inductive à partir des constructions suivantes (ici, δ, δ_1 et δ_2 sont eux-mêmes des programmes Golog) :

— a — action primitive : une action du calcul des situations.

— $\phi?$ — test : lors d'un choix non déterministe, seules les branches dans lesquelles ϕ est vraie peuvent être exécutées.

— $\delta_1; \delta_2$ — séquence : δ_1 est exécuté, puis δ_2 est exécuté.

— $\delta_1 | \delta_2$ — choix non-déterministe de programme : soit δ_1, soit δ_2 est exécuté.

— $\pi(x)\delta(x)$ — choix non-déterministe de paramètres : $\delta(x)$ est exécuté pour une certaine valeur de x, où x est une variable libre dans $\delta(x)$.

— **if** ϕ **then** δ_1 **else** δ_2 — conditionnelle : si ϕ est vraie, δ_1 est exécuté, sinon δ_2 est exécuté.

— **while** ϕ **do** δ — boucle tant que : δ est exécuté tant que ϕ est vraie.

— **proc** $P(\vec{v})$ δ **endProc** — procédure : définit une procédure P qui peut être appelée ensuite dans le programme.

Ces constructions peuvent être utilisées pour écrire un programme dans le langage de la théorie du domaine. Étant donné une théorie du domaine $\mathscr{D}$ et un programme Golog δ, l'exécution du programme consiste à trouver une séquence d'actions $a_0, \ldots, a_{n-1}$ telle que :

$$\mathscr{D} \models \text{Do}(\delta, S_0, \text{do}(a_{n-1}, \text{do}(\ldots, \text{do}(a_0, S_0) \ldots)))$$

Si l'on abrège la situation $\text{do}(a_{n-1}, \text{do}(\ldots, \text{do}(a_0, S_0) \ldots))$ par s_n, alors l'expression $\text{Do}(\delta, S_0, s_n)$ dénote le fait que le programme Golog δ commençant dans la situation S_0 terminera licitement dans la situation s_n .

2.3 . Définition du domaine et profil de l'apprenant

Reprenons notre exemple introduit à la section précédente. Nous considérons disposer d'un certain nombre de ressources ou grains pédagogiques. Ces ressources sont reliées implicitement par une relation de dépendance : une ressource r introduit une KC c qui appartiendra aux KC prérequises pour accéder à une ressource r'.

La première idée consiste à marier une représentation symbolique des connaissances avec le formalisme de la logique du premier ordre pour que cette représentation soit manipulable par le mécanisme de calcul du parcours, lui-même également défini en logique des prédicats.

Nous avons deux modèles à définir :

1. le modèle du domaine, constitué de fluents qui sont essentiellement invariables en fonction de la situation, parce que nous considérons que les ressources pédagogiques n'évoluent pas pendant que l'apprenant utilise l'HA ;

2. le modèle ou profil de l'apprenant, également constitué de fluents, mais dont la valeur évolue avec les situations.

Les modèles de l'apprenant et du domaine sont décrits indépendamment, chacun par un ensemble de fluents qu'il faut pouvoir représenter. Le modèle du domaine comprendra entre autres la relation de prérequis entre CC qui est constante – indépendante de la situation, ou bien encore le niveau de difficulté d'une ressource qui est également constant. À l'inverse, le profil de l'apprenant sera constitué de fluents comme seen(c, s) qui dépendent bien évidemment de la situation.

Une première approche a consisté à utiliser les graphes conceptuels de [68]. Les graphes conceptuels sont des multi-graphes étiquetés : les nœuds de concept sont connectés par des nœuds de relation. Des contextes peuvent être introduits, par exemple pour fournir une négation imbriquée, et des liens co-référents peuvent être ajoutés à un graphe pour exprimer les variables et leur portée. Dans les graphes conceptuels simples, chaque étiquette de nœud est composée d'un type et d'un référent générique/individuel. Ce formalisme possède la même expressivité que la logique des prédicats du premier ordre [70], mais présente une plus grande lisibilité pour notre besoin. Des travaux ont été menés lors d'un stage de niveau master par Doru Arfire [2] pour construire un outil de composition qui permette de définir graphiquement un graphe conceptuel dédié à notre problème.

Toute la dynamique du système repose sur l'évolution du profil de l'apprenant au travers des actions. Le simple fait de considérer un profil utilisateur nous amène dans le champ de la personnalisation. Nous verrons plus loin 3.2 la nécessité de définir plus précisément la notion de personnalisation.

Rétrospectivement, nous pouvons dire que nous avons implémenté une base de raisonnement ontologique dans le formalisme des graphes conceptuels. Des formalismes alternatifs auraient pu être utilisés mais aucun d'entre eux n'avait un statut de standard [2]. L'intérêt du formalisme des graphes conceptuels a été sa simplicité de manipulation en Prolog, ainsi que sa capacité à représenter également des règles de raisonnement [3].

2.4 . Démonstrateur

Nous avons réalisé un système entièrement fondé sur une représentation des connaissances à base de graphe conceptuel avec un pilotage par le calcul des situations. L'architecture de ce système est reproduite en figure 2.1.

2. Aujourd'hui, OWL est un standard promu par le W3C.
3. Caractéristique des langages homoïconiques, où il y a équivalence entre code et données.

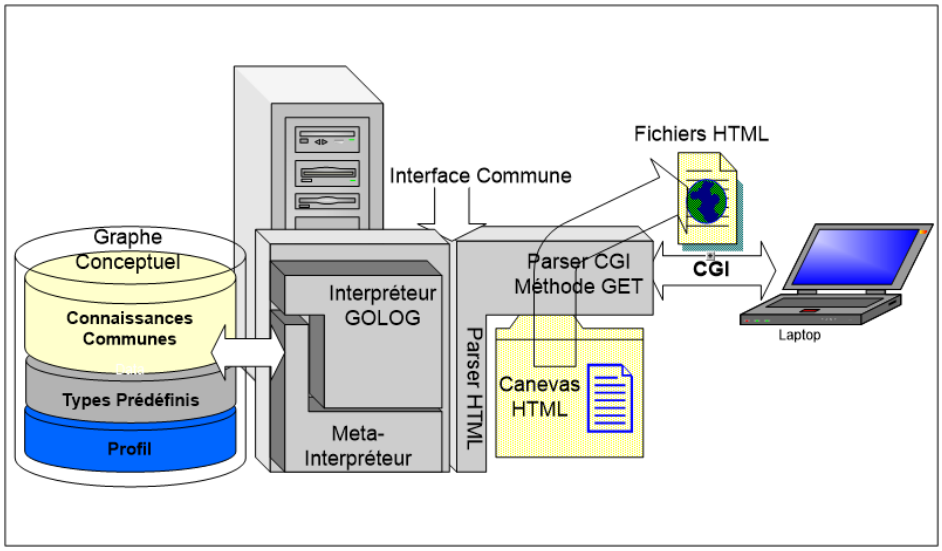


Figure 2.1 – Architecture d'un système à base de graphes conceptuels et de programmation Golog pour un hypermédia adaptatif.

Ce système a permis de programmer très aisément deux applications :

1. un hypermédia adaptatif de type « polycopié », avec un parcours personnalisé,
2. un jeu pour apprendre à reconnaître des animaux, destiné à des enfants (figure 2.2).

Il est à noter que lors de cette réalisation, aucune des facilités disponibles aujourd'hui pour composer des sites web dynamiques n'étaient disponibles. La seule interface disponible était l'interface CGI. La programmation logique a toutefois permis d'intercéder facilement sur la production des pages web.

Figure 2.2 – Jeu pour apprendre à reconnaître des animaux, destiné à des enfants.

Le graphe conceptuel contient des liens selon le formalisme suivant :

```
[[concept1],[concept2]]--->(etiquette)--->[concept3].
[premisse]--->(implique)--->[conclusion].
```

Des éléments classiques sont prédéfinis :

— des types : `user`, `clink`, `bool`, `int`, ...

— des opérations : plus, moins, mult, div, max, min, ...

L'application est réalisée selon un mode qui serait qualifié aujourd'hui de « Single Page Application » ou « Application Mono-Page »[4]. L'application est basée sur un patron de page HTML et un métainterpréteur unique la pilote. Le comportement spécifique de l'application est

4. https://developer.mozilla.org/fr/docs/Glossary/SPA

défini dans le graphe conceptuel lui-même et exécuté par le métainterpréteur. Les définitions dans le graphe conceptuel prennent la forme :

```
%
% Les procédures et les fonctions :
%
proc(Nom,Prog) :- holds([Nom]--->(proc)--->[Prog],[true]).
func(F,Prog) :- F =.. [Fonct|LVars],
            holds([[Fonct]|LVars]--->(func)--->[Prog],[true]).
%
% Les types prédéfinis, ainsi que leurs opérations :
%
[X]--->(isa)--->[user] :- currentUser(X).
[[_|_]--->(_)--->[_|_]]--->(isa)--->[clink].
[X]--->(isa)--->[int] :- integer(X).
[[X],[Y]]--->(plus)--->[Res] :- integer(X), integer(Y), Res is X + Y.
...
```

La programmation logique nous a permis de mettre en œuvre ce type d'application dynamique avec une facilité déconcertante. La formalisation complète de l'application au niveau logique reste séparée du substrat qui en permet l'exécution.

L'avantage par rapport aux systèmes construit sur des règles procédurales est confirmé. Golog se révèle être un outil adéquat pour calculer la planification des parcours de nos apprenants et surtout permettre une définition claire de ceux-ci, qui ne soit pas entachée par des considérations de plus bas niveau.

2.5 . Conclusion

Cette expérimentation a révélé assez rapidement le besoin sinon l'envie d'étendre les opérateurs disponibles et la gestion des informations de profil de l'utilisateur : le jeu illustré ci-dessus nécessite la mémorisation de l'avancement de l'apprenant – au minimum l'historique des questions présentées et de ses réponses. Nous avons donc expérimenté et étendu le mécanisme de base avec plusieurs capacités :

— la capacité d'interagir avec le graphe conceptuel pour y stocker un profil utilisateur,

— la capacité à appliquer des règles de type « ontologique » utilisant les relations *isa* du graphe conceptuel,

— l'introduction de nouveaux opérateurs tels que `desirable(a, s)` indiquant que l'action a est souhaitable pour l'apprenant courant dans la situation s.

L'approche d'un EIAH fondé sur le calcul des situations a donné lieu aux publications [23], [24]. L'utilisation du calcul des situations et l'approche logique du calcul du parcours sont à la base de la thèse développée par Cédric Jacquiot [4]. Cette thèse développe plus avant un modèle générique pour la conception d'hypermédias adaptatifs. Le modèle développé dans cette thèse permet de créer des hypermédias adaptatifs dans des domaines très variés, de manière essentiellement déclarative. Il généralise des notions retrouvées dans de nombreux modèles existants comme AHAM [33].

Dans un travail ultérieur mené par Najet Zemirline lors de sa thèse, la réutilisation des composants de l'hypermédia adaptatif a été analysée plus avant. Une modélisation ontologique à l'aide de l'environnement Protégé a été définie et un plugin spécifique pour Protégé a été développé. Ce travail a donné lieu aux publications [16]–[18] (reprises ci-dessous).

Le chapitre suivant va analyser plus précisément la notion de personnalisation, celle de personnification d'un agent virtuel et les façons possibles de les mettre en œuvre dans le cadre choisi.

Production scientifique et encadrements

Encadrement de stages de Master :

— C. Dinca, « Construction d'un Site Web Adaptatif », Universitatea Politehnica Bucuresti, Stage de fin d'études, sept. 2000

— D. Arfire, "Graphical Knowledge Representation", Universitatea Politehnica Bucuresti, Stage de fin d'études, Jun. 2006

Co-encadrement de thèse :

— C. Jacquiot, « Modélisation logique et générique des systèmes d'hypermédias adaptatifs », Thèse de doctorat, Paris 11, 1er jan. 2006, Prix de Thèse de la Fondation Jean-Luc Lagardère

Logiciels :

— GLAM - système de conception modulaire et générique pour des EIAH.

Publications :

— C. Jacquiot, Y. Bourda, and F. Popineau, "GEAHS: A Generic Educational Adaptive Hypermedia System", presented at the EdMedia + Innovate Learning, Association for the Advancement of Computing in Education (AACE), 2004, pages 571–578

— C. Jacquiot, Y. Bourda, and F. Popineau, "GEAHS: A Generic Educational Adaptive Hypermedia System Based on Situation Calculus", in *Adaptive Hypermedia and Adaptive Web-Based Systems, Third International Conference, AH 2004, Eindhoven, the Netherlands, August 23-26, 2004, Proceedings*, P. D. Bra and W. Nejdl, Eds., series Lecture Notes in Computer Science, volume 3137, Springer, 2004, pages 413–416. DOI: 10.1007/978-3-540-27780-4_63 (disponible en annexe)

— C. Jacquiot, Y. Bourda, and F. Popineau, "Reusability in GEAHS", in *Engineering Advanced Web Applications: Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering (ICWE 2004), Munich, Germany, 28-30 July, 2004*, M. Matera and S. Comai, Eds., Munich, Germany: Rinton Press, 2004, pages 199–209

— C. Jacquiot, Y. Bourda, F. Popineau, A. Delteil, and C. Reynaud, "GLAM: A generic layered adaptation model for adaptive hypermedia systems", in *Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference, AH 2006, Dublin, Ireland, June 21-23, 2006, Proceedings*, H. A. V. Wade and B. Smyth, Eds., series LNCS, volume 4018, Springer-Verlag, Jun. 2006, pages 131–140 (disponible en annexe)

— N. Zemirline, Y. Bourda, C. Reynaud, and F. Popineau, "Assisting in Reuse of Adaptive Hypermedia Creator's Models", in *Adaptive Hypermedia and Adaptive Web-Based Systems*, W. Nejdl, J. Kay, P. Pu, and E. Herder, Eds., series Lecture Notes in Computer Science, volume 5149, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pages 357–360. DOI: 10.1007/978-3-540-70987-9_53

— N. Zemirline, Y. Bourda, C. Reynaud, and F. Popineau, "MESAM: A Protégé Plug-in for the Specialization of Models", in *11th International Protégé Conference*, Amsterdam, Netherlands, Jun. 2009, 3 pages

— N. Zemirline, C. Reynaud, Y. Bourda, and F. Popineau, "A pattern and rule-based approach for reusing adaptive hypermedia creator's models", in *Proceedings of 16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns*, A. Gangemi and J. Euzenat, Eds., series Lecture Notes in Computer Science, volume 5268, Acitrezza, Catania, Italie: Springer-Verlag, 2008, pages 17–31. DOI: 10.1007/978-3-540-87696-0_5

3 - Agents personnalisés et personnifiés

3.1 . Agents virtuels intelligents

Nous avons vu comment délivrer un contenu personnalisé, adapté à un apprenant. Toutefois, notre modélisation n'est pas aussi élaborée que nous pourrions le souhaiter :

— l'EIAH n'est pas exactement modélisé sous la forme d'un agent : en particulier, il n'y a pas de *sensing* explicite des actions de l'apprenant ;

— la personnalisation reste faible : le programme exécuté est le même pour tous les apprenants ; les variations dépendent uniquement de l'historique des interactions.

Pour aller plus loin dans l'expérience de l'apprenant et dans la modélisation de l'adaptation dans l'EIAH, nous proposons de faire reposer l'EIAH sur un agent virtuel interactif ou IVA (*Interactive Virtual Agent*). L'expression de l'interaction de l'apprenant avec un agent capable de *sensing* est plus naturelle et dynamique que l'interaction vue au chapitre précédent, dans la mesure où l'agent peut percevoir différents fluents à chaque action de l'apprenant et adapter son comportement.

Nous souhaitons développer l'idée que la personnalisation du comportement de l'agent n'est qu'un cas particulier de l'altération du comportement d'un agent ou d'un système informatique en fonction du contexte. La question se pose également d'envisager ces altérations du comportement de façon générique et d'unifier les techniques de personnalisation ainsi que d'autres altérations du comportement connexes, celles consistant à donner une personnalité ou un caractère à l'agent et que nous nommerons ici personnification. La figure 3.1 illustre ces deux concepts de personnalisation – prise en compte du profil utilisateur – versus personnification – prise en compte d'un profil de personnalité agent.

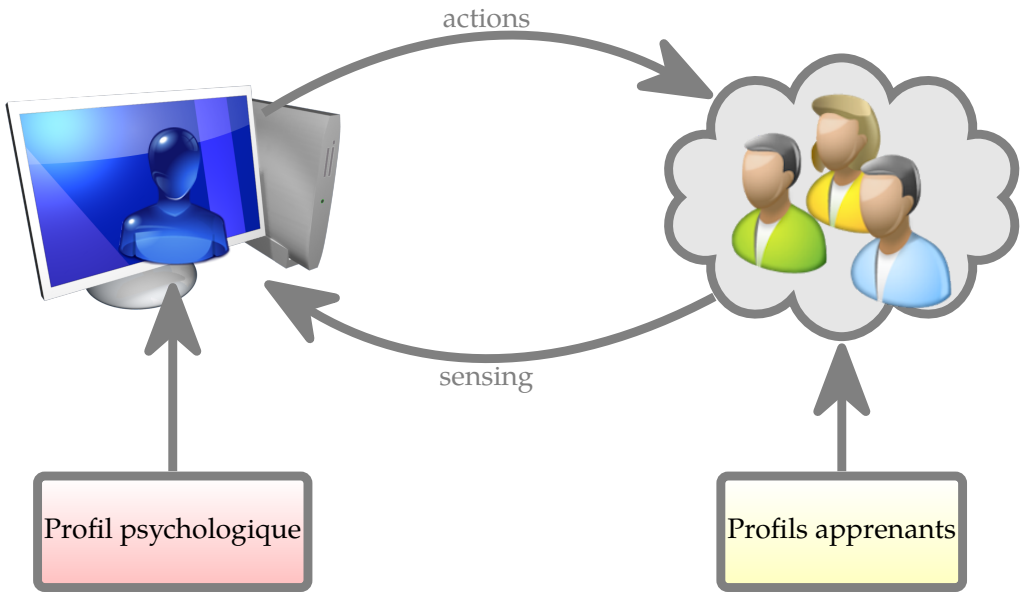


Figure 3.1 – L'apprenant comme l'agent peuvent être dotés d'un profil. La prise en compte du profil de l'apprenant par l'agent mène à un comportement personnalisé. L'attribution d'un profil agent à l'agent lui-même mène à un comportement personnifié.

Les exemples pris dans cette section et dans l'annexe A pour illustrer le comportement de l'agent pourront sortir du cadre des EIAH. Il existe des exemples canoniques et historiques dans ce domaine auxquels il est plus simple de se rattacher.

Dans le chapitre précédent, nous avons vu que notre agent était limité : les actions entreprises par l'apprenant ne sont pas du ressort de l'agent. L'agent peut donc planifier, mais il ne sait pas ce que l'apprenant va décider.

Le problème auquel nous nous heurtons ici est un problème de modélisation. L'agent que l'on cherche à modéliser et pour lequel on souhaite établir un comportement est en *interaction* avec l'apprenant. Nous avons donc besoin que l'agent puisse intégrer l'information en provenance des actions de l'apprenant.

Les applications en robotique cognitive de Golog ont le même type de demande : le robot a besoin d'acquérir de l'information de son environnement. Golog a été étendu pour prendre en compte cet aspect au travers du *sensing* : l'action qui demande à mesurer la valeur d'un fluent. Ce modèle classique d'agent en interaction avec son environnement, utilisé largement en robotique cognitive, s'applique également à l'interaction entre l'utilisateur et une application Web, en particulier si cette application est un EIAH.

L'objectif fixé ici sera de montrer dans quelle mesure il est possible de définir des variations du comportement d'un agent qui suivent un profil. Le point de départ est le programme définissant le comportement d'un agent purement rationnel dans le cas où les profils sont vides : ce comportement est qualifié de neutre. Nous montrerons comment il est possible de définir un mécanisme de transformation d'un programme logique neutre qui permette de s'adapter à un utilisateur – cas de la personnalisation – ou bien de donner une personnalité à un agent – cas de la personnification.

En section 3.2, nous clarifierons le concept de personnalisation par rapport à la notion de profil de l'apprenant.

En section 3.3, nous introduirons le concept de profil de personnalité pour un agent qui reste rationnel, compte tenu de sa personnalité. Cette personnification induit des contraintes sur son comportement et le pilotage par planification garantit un comportement rationnel sous ces contraintes.

En section 3.4, nous présenterons les raisons motivant le passage de Golog à Indigolog. Nous introduirons un modèle générique permettant d'interagir avec un IVA au travers d'une application Web.

En section 3.5, nous montrerons comment il est possible d'altérer les choix d'un agent en introduisant des préférences dans la sémantique de Indigolog.

En section 3.6, nous définirons les opérateurs de transformation d'un programme Indigolog qui permettront de modifier la façon dont les choix sont effectués dans le programme.

En section 3.7, nous introduirons la notion d'affinité entre des actions auxquelles on attribue certaines caractéristiques et les caractéristiques du profil.

En section 3.8, nous assemblerons les éléments précédents pour définir un processus de transformation qui, à partir d'un programme neutre, permettra automatiquement de créer un programme Indigolog qui tienne compte d'un profil d'apprenant ou d'un profil de personnalité. Cette transformation sera automatique ou semi-automatique. Le programme résultant sera une restriction formelle du programme régissant le comportement neutre. En particulier, l'objectif à atteindre. Le programme transformé autorisera des variations par rapport au programme neutre : la prise en compte d'un profil utilisateur ou de personnalité conduira à des choix d'actions différents pour atteindre l'objectif.

En section 3.9, nous présenterons deux exemples courts de cette technique. Des exemples plus approfondis sont fournis en annexe A.

Enfin, la section 3.10 conclura sur ce processus de transformation et son utilisation.

3.2 . Personnalisation faible et forte

Il est possible de distinguer deux types de personnalisation. Dans le cadre des hypermédias adaptatifs présentés dans le chapitre précédent, il existe un enchaînement des contenus qui est particulier à chaque apprenant : chacun va choisir un lien suivant selon son appétence – sauf dans le cas où l'HA ne donnerait aucun choix.

Peut-on pour autant parler de personnalisation dans ce cas ? Il n'y a rien de vraiment caractéristique de l'utilisateur qui soit mis en jeu à ce niveau.

La personnalisation débute lorsqu'on prend explicitement en compte des éléments personnels de l'apprenant : des informations qui constituent son profil.

On peut donc établir une hiérarchie dans le niveau de personnalisation d'une application et poser les définitions suivantes.

Nous appellerons système *non personnalisé* un système dont le comportement ne varie pas en fonction de l'utilisateur avec lequel il interagit.

Nous qualifierons de *faiblement personnalisé* un système qui fait usage d'informations sur l'utilisateur ou de l'historique de ses interactions de manière ad hoc, sans les organiser ni en extraire un profil.

Enfin, nous qualifierons de *fortement personnalisé* un système qui utilise explicitement les informations sur l'utilisateur dont il dispose, affinées en un modèle, pour adapter le service qu'il fournit.

Nous nous intéresserons dans la suite à la personnalisation forte et nous supposerons disposer d'un profil de l'utilisateur. La figure 3.2 illustre le cas où l'on dispose d'un programme agent initial neutre (sans profil). Sous l'hypothèse de deux profils d'utilisateurs différents, le programme neutre est transformé en des programmes personnalisés.

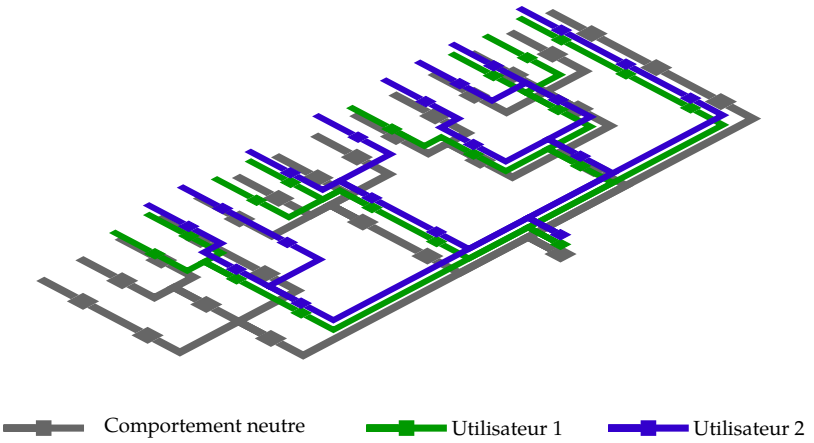


Figure 3.2 – Illustration d'un programme neutre et d'exécutions différentes sous contrainte de profils utilisateurs différents.

3.3 . Profils de personnalité

La personnalisation fait référence à l'adaptation du comportement de l'agent à des utilisateurs qui possèdent des profils différents. Il existe une autre dimension de l'adaptation du comportement d'un agent : celle qui consiste à donner une personnalité à l'agent, et à le faire interagir selon différentes modalités avec son environnement. Certaines actions peuvent être vues à haut niveau d'abstraction comme réalisant un but – une condition nécessaire à l'achèvement d'un plan. La mise en oeuvre de ces actions peut se dérouler à un niveau concret de différentes façons.

Pour illustrer ce concept, prenons le cas d'un robot que nous appellerons Robbie et qui doit pénétrer dans une salle où se trouve sa batterie de secours. Cette salle est fermée à clé. Ce robot peut avoir plusieurs manières de réaliser le but « entrer dans la salle » : soit il peut aller chercher la clé, soit il peut crocheter la serrure, soit il peut défoncer la porte – sous les hypothèses que ces actions sont réalisables. Chacune de ces manières correspond à un trait de personnalité différent.

Les techniques classiques de personnification des agents sont très différentes des techniques de personnalisation. Cela est principalement dû au fait que, dans les domaines qui s'intéressent à la personnification, les agents sont basés sur des architectures précises et la personnification est intrinsèquement liée à ces architectures. La famille d'architectures *Belief* -

Desire - Intention (BDI) [58] est couramment utilisée, de même que l'architecture cognitive SOAR [45]. Les traits de personnalité sont utilisés pour choisir entre les différents buts possibles d'un agent BDI (les traits influencent les désirs). Une fois le but choisi, les actions sont planifiées et exécutées et les traits psychologiques n'ont plus d'influence.

La taxonomie TFS pour *Traits - Facets - Schemes* a été introduite par [27] pour permettre une mise en relation des actions avec des aspects de personnalité. Cette taxonomie trouve son origine dans le *Five Factor Model* [40] et les versions enrichies qui en ont découlé comme la taxonomie NEO PI-R [32] sur laquelle nous allons nous appuyer.

Le niveau des *Traits* est divisé en 5 classes : {*Openness*, *Conscientiousness*, *Extroversion*, *Agreeableness*, *Neuroticism*}, comme illustré sur la figure 3.3.

Chaque *Trait* est divisé en 6 *Facets* bipolaires, ce qui donne 30 *Facets*. Les *facets* sont désignées par $F_{\text{facetname}}$ et les deux pôles respectivement par $+F_{\text{facetname}}$ et $-F_{\text{facetname}}$. On trouvera par exemple la *Facet* F_{fantasy} avec un pôle positif et un pôle négatif.

Chaque *Facet* est divisée en 1 à 4 *Schemes*. Au contraire des *facets*, les pôles des *schemes* sont dénotés par un nom et son antonyme comme +HARDWORKER et −LAZY ou +IDEALISTIC et −PRACTICAL.

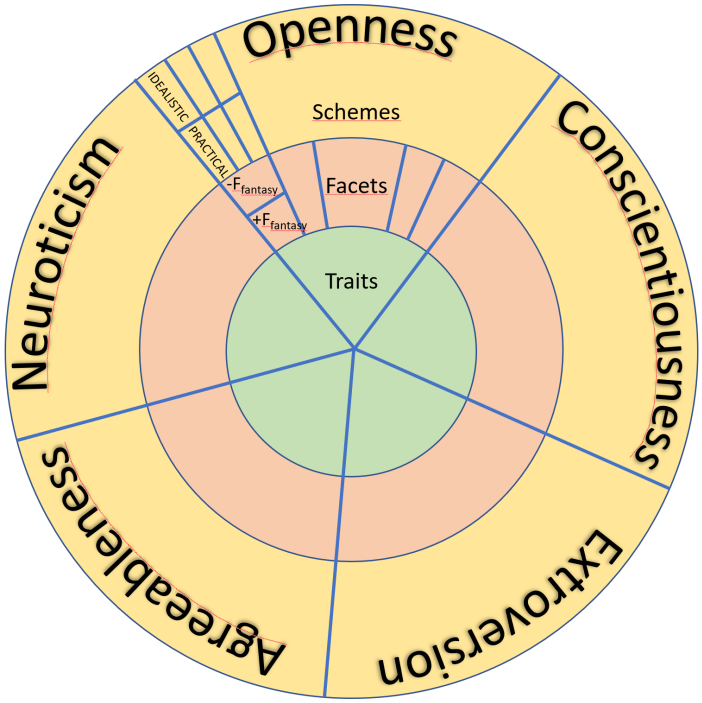


Figure 3.3 – Architecture de la taxonomie *Traits - Facets - Schemes*

Un profil de personnalité pour un agent est défini par un ensemble de *schemes* actifs pour cette personnalité. Pour simplifier, lorsque tous les schemes d'une *facet* sont activés, on peut abréger leur liste en celle de la *facet* (positive ou négative). Il en est de même au niveau des traits.

Voici 3 exemples pour un robot Robbie :

1. Robbie est paresseux : $P_{\text{paresseux}}(\text{Robbie}) = \{-\text{LAZY}\}$. Un seul *scheme* est activé : -LAZY. Le signe moins rappelle seulement la négativité de ce *scheme*. Le *scheme* activé est positionné ainsi : $-\text{LAZY} \in F_{\text{AchievementStriving}} \in -T_{\text{Conscientiousness}}$.
2. Robbie est un loubard : $P_{\text{mauvaisGarçon}}(\text{Robbie}) = \{-F_{\text{Dutyfulness}}, -\text{LAZY}, +\text{AMBITIOUS}, +\text{DOMINEERING}, -\text{DISHONEST}, -\text{HARMFUL}\}$. Ici, la *facet* $-F_{\text{Dutyfulness}}$ abrège ses 4 *schemes* qui sont −UNFAITHFUL, −IMMORAL, −LAWLESS, −PREJUDICED.
3. Robbie est désagréable : $P_{\text{désagréable}}(\text{Robbie}) = \{-T_{\text{Agreeableness}}\}$. Le trait $-T_{\text{Agreeableness}}$ est activé, donc tous ses *schemes* négatifs sont activés.

Cette taxonomie possède deux intérêts qui se conjuguent :

1. elle fournit des axes selon lesquels positionner toute personnalité,
2. elle permet de relier des actions à des *schemes*.

À l'aide de cette taxonomie, nous pourrons donc choisir un profil de personnalité et associer des actions spécifiques aux *schemes* composant cette personnalité. Ces actions spécifiques viendront remplacer des actions génériques dans un squelette de plan neutre (sans profil). Le plan exécuté sera différent en fonction du profil de personnalité de l'agent et le but sera toujours garanti.

Le profil de personnalité peut être appliqué aussi bien à l'agent – cas de la personnification – qu'à l'utilisateur – cas de la personnalisation. Nous verrons que le même principe de transformation des programmes s'applique dans les deux cas.

3.4 . Une théorie des applications Web

Comme nous l'avons fait remarquer précédemment, notre modélisation n'est pas réellement celle d'un agent parce que l'agent est capable de planification mais pas de *sensing* : il ne peut « mesurer » les fluents dont la valeur a changé à la suite d'une action de l'utilisateur. Bien que le calcul de situations a souvent été proposé pour piloter un agent dans le cadre d'une application Web, les détails de la mise en oeuvre n'ont jamais été abordés. Dans [15], nous avons proposé une solution pour cette mise en oeuvre.

Pour répondre à la nécessité de prendre en compte la réponse de l'utilisateur, nous utilisons des théories d'actions gardées [35]. Dans une théorie d'action gardée, il existe un nombre fini de fonctions de *sensing* (ou fonctions de capteur), qui sont des fonctions unaires dont l'argument est une situation et qui modélisent un capteur disponible pour l'agent. Par exemple une fonction de *sensing thermometer*(s) retournera la température mesurée par le thermomètre dans la situation s. La garde est utilisée pour indiquer quand l'axiome est applicable. Une théorie de l'action gardée est identique à une théorie de l'action simple, avec la différence que les axiomes successeur d'état sont remplacés par deux ensembles d'axiomes : un équivalent gardé des axiomes successeur d'état et un ensemble d'axiomes gardés de résultat de *sensing* :

— Les axiomes successeur d'état gardés, de la forme $\alpha(\vec{x}, a, s) \Rightarrow [F(\vec{x}, (a, s)) \equiv \gamma(\vec{x}, a, s)]$ où α est une formule contenant des fluents, F est un fluent relationnel et γ est une formule contenant des fluents. Ces axiomes permettent la régression, c'est-à-dire de déduire la valeur d'un fluent dans une situation à partir de la valeur dans les situations précédentes.

— Des axiomes gardés de résultat du *sensing*, de la forme $\beta(\vec{x}, s) \Rightarrow [F(\vec{x}, s) \equiv \rho(\vec{x}, s)]$ où β est une formule contenant des fluents de capteur, F est un fluent relationnel et ρ est une formule de capteur sans fluent. Ces axiomes permettent de détecter la valeur d'un fluent dans une certaine situation, en utilisant des fonctions de détection.

Les gardes α et β sont utilisées pour préciser quand les formules de régression et de détection sont utilisables. Cela nous permet d'avoir des fluents qui peuvent être détectés dans certaines situations, et qui peuvent être modifiés par des actions dans d'autres situations. IndiGolog [34], [36] est une extension de Golog dans laquelle les actions primitives sont des actions d'une théorie d'action gardée. Il introduit également une nouvelle construction du langage, l'opérateur d'exploration $\Sigma(\delta)$ qui effectue une exploration en avant sur le programme δ pour s'assurer que les choix non-déterministes sont résolus de manière à garantir son bon achèvement. En dehors du bloc à explorer, les choix non déterministes sont résolus de manière arbitraire. Par conséquent, le calcul est réduit car aucune exploration n'est effectuée en dehors du bloc examiné.

Pour mettre en oeuvre Indigolog dans le cadre d'une architecture d'application Web, nous avons introduit un proxy nécessaire à l'interaction entre l'agent et l'utilisateur. Le comportement de ce proxy a été formalisé en calcul situationnel (voir [5], section 3.2).

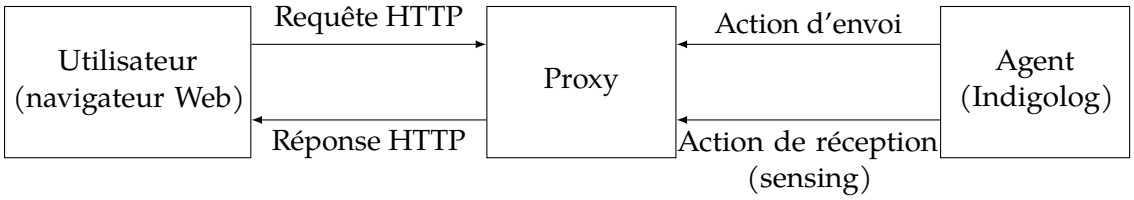


Figure 3.4 – Échanges entre le navigateur Web et l'agent à travers le proxy.

Ce proxy correspond également à une partie du rôle du méta-interpréteur de la figure 2.1. Il est à noter que les flèches issues de l'agent sont correctement orientées : c'est bien l'agent qui entreprend l'action de *sensing*.

3.5 . Altération des choix d'un agent

Indigolog dispose d'un opérateur de choix non-déterministe de programmes noté |. Le programme $\delta_1 \mid \delta_2$ exprime que soit δ_1 soit δ_2 doit être exécuté. Il n'est cependant pas possible d'exprimer une préférence sur cette exécution. Si δ_1 et δ_2 sont possibles, rien ne spécifie lequel des deux sera exécuté et le choix sera fait de manière arbitraire.

Nous introduisons un opérateur de choix de programmes avec préférence, noté $\rangle$. Le programme $\delta_1 \rangle \delta_2$ signifie « essayer δ_1, sinon δ_2 » : l'un des deux programmes est exécuté, avec une préférence pour δ_1 si δ_1 et δ_2 sont tous les deux possibles. Le tableau 3.1 détaille tous les cas possibles. Il est à noter que, comme le fait remarquer [34], l'implémentation de Indigolog évalue les branches de gauche à droite, donc l'implémentation de l'opérateur | a le même comportement que celle de l'opérateur $\rangle$. Cependant, il est important de distinguer les cas où le choix est indifférent et les cas où l'ordre d'évaluation des options a une importance. Ce fait pourra tout de même être utilisé pour simplifier l'implémentation en gardant l'implémentation de l'opérateur | plutôt que de prendre la sémantique formelle de l'opérateur $\rangle$ qui est beaucoup plus complexe.

δ_1 possible ?	δ_2 possible ?	exécution de $\delta_1 \mid \delta_2$	exécution de $\delta_1 \rangle \delta_2$
oui	oui	δ_1 ou δ_2	δ_1
oui	non	δ_1	δ_1
non	oui	δ_2	δ_2
non	non	impossible	impossible

Table 3.1 – Exécutions des opérateurs de choix

Cette construction est similaire à la construction $\mathbf{lex}(\delta_1, \delta_2)$ définie dans [39], mais son implémentation est différente, puisque les formalismes sont différents. L'idée est également similaire à la construction $\triangleright$ de CANPlan[67], à ceci près que dans $\delta_1 \rangle \delta_2$, aucune partie de δ_1 n'est exécutée si δ_1 n'est pas entièrement exécutable.

La sémantique formelle de l'opérateur est donnée dans les équations (3.1) et (3.2).

$$\begin{aligned} Final(\delta_1 \rangle \delta_2, s) \equiv\ & Final(\delta_1, s) \\ & \vee \neg \exists s''.Do(\delta_1, s, s'') \wedge Final(\delta_2, s) \end{aligned} \tag{3.1}$$

$$\begin{aligned} Trans(\delta_1 \rangle \delta_2, s, \delta', s') \equiv\ & \\ & \exists s''.Do(\delta_1, s, s'') \wedge Trans(\delta_1, s, \delta', s') \\ & \vee \neg \exists s''.Do(\delta_1, s, s'') \wedge Trans(\delta_2, s, \delta', s') \end{aligned} \tag{3.2}$$

Rappelons ici que $Do(\delta, s, s')$, rencontré précédemment page 16, signifie que l'exécution complète du programme δ dans la situation s est possible et se termine légitimement dans la situation s'. Le terme $Final(\delta, s)$ signifie que le programme δ peut être considéré comme terminé dans la situation s. La définition de $Final(\delta_1 \rangle \delta_2, s)$ donnée en (3.1) déclare que le programme $\delta_1 \rangle \delta_2$ est terminé si δ_1 est terminé, ou si δ_1 n'est pas exécutable et que δ_2 est terminé. Le terme $Trans(\delta, s, \delta', s')$ signifie que l'exécution d'une étape élémentaire de δ dans s résulte dans la situation s', avec le programme δ' restant à être exécuté. La définition de $Trans(\delta_1 \rangle \delta_2, s, \delta', s')$ donnée en (3.2) déclare que si δ_1 est intégralement exécutable dans s, une étape de δ_1 sera exécutée, sinon, c'est une étape de δ_2 qui le sera. Ces deux définitions peuvent se lire « exécuter $\delta_1 \rangle \delta_2$ dans la situation s est équivalent à exécuter δ_1 si δ_1 peut être entièrement exécuté dans s, sinon, à exécuter δ_2 ».

3.6 . Les transformations de programmes agent

Nous allons introduire ici les opérateurs de transformation – ou plus simplement transformations – de programmes Indigolog qui ont été mis au point pour modifier la manière dont les choix sont effectués dans le programme.

Ces transformations sont de la forme $a \rightarrow b$; l'application d'une transformation consiste à remplacer un programme de la forme a par la forme b équivalente. Cette transformation peut se faire sur le programme lui-même, ou sur une partie du programme. Ainsi, une transformation de la forme $a \rightarrow b$ permet de transformer le programme $\delta_1; (a \mid \delta_2)$ en $\delta_1; (b \mid \delta_2)$.

On notera tout d'abord que pour modifier les choix effectués par le programme, il faut que le programme comporte des opérateurs de choix non-déterministe. Si ce n'est pas le cas, le programme ne possède qu'une exécution possible et ne peut être modifié sans introduire de nouveaux éléments, ce qui risquerait de le rendre non conforme au programme original. Il est toutefois possible sous certaines conditions d'introduire des choix dans un programme qui n'en possède pas en passant par la notion de famille d'actions comme nous allons le voir.

Nous allons considérer au total 7 transformations. Les 5 premières concernent la structure du programme :

T1 $\delta_1 \mid \delta_2 \rightarrow \delta_i \rangle \delta_j \qquad i, j \in \{1, 2\}, i \neq j$

T2 $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?; \delta(x) \rangle \pi x.\delta(x)$

T3 $\delta_1 \mid \delta_2 \rightarrow \delta_i$ avec $i \in \{1, 2\}$

T4 $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?; \delta(x)$

T5 $\delta \rightarrow$ **if** ϕ **then** δ **else** δ

Les transformations T1 et T2 sont des transformations simples qui expriment des préférences. Si un programme est exécutable, le résultat de sa transformation par T1 ou T2 sera également exécutable (voir [5]).

La transformation T1 consiste à considérer que si les exécutions de δ_1 et δ_2 sont des alternatives possibles équivalentes, on peut considérer que δ_1 doit être préférée à δ_2. Cette transformation T1 a pour unique effet de forcer un ordre d'évaluation entre les deux branches du programme et ne change pas le résultat. Si δ_1 n'est pas possible, l'exécution se rabattra sur δ_2.

La transformation T2 fonctionne de manière analogue, mais pour les arguments d'un programme. Elle change l'ordre d'évaluation des arguments lors d'un choix non-déterministe d'arguments, de manière à ce que les arguments x vérifiant $\phi(x)$ soient évalués en premier.

Les transformations T3 et T4 sont des transformations destructrices : elles restreignent les choix possibles au lieu de simplement les orienter. Elles sont analogues aux transformations T1 et T2 respectivement, mais interdisent certaines options en perdant l'option non préférée.

La transformation T3 retire purement et simplement l'une des options du programme, rendant obligatoire l'exécution de celle que l'on conserve. De manière identique, la transfor-

mation T4 permet de sélectionner les arguments x tels que $\phi(x)$, plutôt que de prendre des x quelconques.

La transformation T5 est une transformation purement technique d'expansion des programmes. Si l'on souhaite par exemple transformer un programme $\delta_1 \mid \delta_2$ de manière à ce que δ_1 soit préféré à δ_2 seulement quand une formule ϕ est vraie, la forme originelle du programme peut ne pas le permettre. La transformation T5 servira à introduire la formule ϕ explicitement, ce qui permettra par la suite d'exprimer notre préférence à l'aide des transformations T1-T4.

Il est à noter que les transformations T1-T5 ne font que préciser les choix lors de l'exécution du programme. Les exécutions du programme transformé sont des exécutions valides du programme original et donc des restrictions de ce programme original, en un sens formel qui correspond au schéma de la figure 3.2 et qui a été défini formellement dans [5].

Les transformations précédentes ciblent la structure du programme. Nous souhaitons introduire des variantes au niveau des actions : dans la pratique, il existe souvent plusieurs manières d'atteindre un objectif élémentaire. Pour notre robot Robbie, plusieurs actions peuvent aboutir à ouvrir une porte : soit la crocheter, soit trouver la clé adéquate, soit défoncer la porte, etc.

Nous introduisons la notion de famille d'actions pour désigner des actions qui modifient les même fluents. Le prédicat *family* est défini par le concepteur de l'application. La formule $family(F)$ est vraie si l'ensemble F est une famille d'actions, c'est-à-dire un ensemble d'actions modifiant les mêmes fluents dans les mêmes états. Une action donnée ne peut faire partie que d'une seule famille.

Dans notre exemple, les trois actions *openDoorWithKey*, *lockpickDoor* et *breakDoorOpen* modifient le même fluent : $\neg\text{opened}(door, s)$ devient $\text{opened}(door, s')$ en supposant que l'exécution de l'action fasse passer de l'état s à l'état s'. Les préconditions sont différentes pour les 3 actions : *openDoorWithKey* va exiger de disposer de la clé adéquate, *lockpickDoor* va exiger de disposer d'un rossignol et *breakDoorOpen* n'aura pas de précondition spécifique.

Avec cette définition d'une famille d'actions, nous pouvons introduire la transformation T6 qui va permettre de choisir une action quelconque parmi une famille :

T6 Si $F = \{a_1, \ldots, a_n\}$ est une famille d'actions,

$$\forall a_i \in F \qquad a_i \rightarrow a_1 \mid \ldots \mid a_n.$$

Le mécanisme précédent se généralise aux procédures par la définition de famille de procédures et la définition de la transformation T7 autorisant le choix indéterminé de toute procédure dans la même famille.

Le prédicat *familyP* est défini par le concepteur de l'application. La formule $familyP(F)$ est vraie si F est une famille de procédures, c'est-à-dire un ensemble de procédures effectuant la même opération.

T7 Si $F = \{P_1, \ldots, P_n\}$ est une famille de procédures,

$$\forall P_i \in F \qquad P_i \rightarrow P_1 \mid \ldots \mid P_n.$$

Les transformations T6 et T7 sont conçues pour être utilisées conjointement avec les transformations T1 à T4. Reprenons l'exemple de l'agent robot qui doit ouvrir une porte et pour lequel on a défini les actions *openDoorWithKey*, *lockpickDoor* et *breakDoorOpen* rassemblées en une même famille. La transformation T6 suivie de T1 permet de transformer n'importe laquelle de ces actions en :

$$openDoorWithKey \rangle lockpickDoor \rangle breakDoorOpen.$$

On obtient donc un comportement pragmatique où l'agent peut recourir à des comportements inhabituels par rapport à sa personnalité.

Si les opérations *openDoorWithKey*, *lockpickDoor* et *breakDoorOpen* ne sont plus des actions mais des procédures rassemblées dans une famille :

proc *openDoorWithKey*
take(key); move(key, lock); turn(key);
putAway(key); push(door)
endProc

proc *breakDoorOpen*
take(ram);
while ¬destroyed(door) **do**
smack(door); push(door)
endProc

on peut alors utiliser la transformation T7 en lieu et place de la transformation T6.

3.7 . Affinités d'actions

Pour appliquer les transformations précédentes à un programme, nous avons besoin d'exprimer des préférences envers une action ou une procédure selon un profil.

Pour modifier automatiquement un programme, nous avons besoin de savoir si une action (*resp* un programme) est préférée à une autre action (*resp* un autre programme) au regard d'un profil donné. Pour cela, il faut que les actions portent des informations supplémentaires, en dehors du calcul des situations, informations qui puissent être utilisées pour savoir si ces actions sont désirables, neutres ou déplaisantes, par rapport à un profil donné. Ceci est mis en œuvre en attachant des attributs aux actions, similaires à ceux du profil.

Tout d'abord, le concepteur de l'application doit définir ce qui peut modifier le comportement, à savoir un ensemble d'attributs qui sont liés aux actions, et qui sont utilisés pour décider si une action doit être favorisée ou évitée. La nature des attributs dépend du type de modification apportée à l'agent.

Un *attribut* est un atome. L'ensemble des attributs sera désigné par A et sera fourni par le concepteur de l'application.

Le comportement d'un agent est modifié en fournissant à l'agent un point de vue positif ou négatif sur les attributs. Un profil englobe tous les points de vue d'un agent donné. Pour chaque attribut, l'agent peut être fermement opposé, légèrement opposé, neutre, légèrement favorable ou très favorable. Ces valeurs sont respectivement représentées par les valeurs entières -2, -1, 0, 1 et 2 [28]. Nous considérons que les valeurs 1 et -1 indiquent une préférence et ne devraient pas modifier beaucoup le programme, mais les valeurs de 2 et -2 indiquent une exigence, et peuvent conduire à des programmes invalides si elles amènent l'agent à poser son veto sur une partie critique du programme. La valeur 0 dénote l'ambivalence ou la neutralité. Le concepteur de l'application doit garder cela à l'esprit lorsqu'il définit un profil.

Un *profil* est une fonction associant à chaque attribut de l'ensemble A une valeur de l'ensemble $\{-2, -1, 0, 1, 2\}$.

Dans le cas d'un agent personnalisé, les attributs font référence à des choses qui peuvent plaire ou non à l'utilisateur, et le profil indique ce qui plaît et déplaît à un utilisateur spécifique. Dans le cas d'un agent avec personnalité, les attributs font référence aux connotations des actions que l'agent veut favoriser ou éviter. Un tel profil peut être considéré comme un profil psychologique de l'agent puisqu'il indique quel type de comportement il aime ou n'aime pas.

Pour modifier automatiquement un programme, nous avons besoin d'un moyen de savoir si une action (resp. un programme) est préférée à une autre action (resp. un autre programme) par rapport à un profil donné. Cela nécessite que les actions portent des informations supplémentaires, en dehors du calcul des situations, informations qui puissent être utilisées pour savoir si elles sont désirables, neutres ou indésirables, par rapport à un profil donné. Ceci est mis en œuvre en attachant des attributs aux actions, similaires à ceux du profil. Par exemple, dans le cas de Robbie, avec seulement quelques traits, nous pouvons avoir $A = \{\text{lazy}, \text{brutal}, \text{cautious}\}$. Un profil particulier sera défini par $p(\text{lazy}) = 2, p(\text{brutal}) = -1, p(\text{cautious}) = 0$.

La fonction $\mathit{attributes}(a)$ associe un ensemble d'attributs à chaque action a. Cette fonction est fournie par le concepteur de l'application.

En associant une action et un profil, on peut calculer l'affinité entre cette action et ce profil : une valeur numérique indiquant si l'action convient à ce profil.

La fonction d'affinité aff_p associe une valeur numérique aux actions en fonction d'un profil et est définie par :

$$aff_p(a) = \sum_{at \in attributes(a)} p(at).$$

Une action a_1 est donc préférée à une action a_2 sous un profil donné p si et seulement si $aff_p(a_1) > aff_p(a_2)$. Pour transformer des programmes entiers, nous avons besoin de connaître l'affinité du programme entier. Ceci est possible en étendant la définition de aff_p aux programmes par une définition inductive. L'exécution du programme n'étant pas connue à la compilation (moment où la transformation est exécutée), il faut faire des approximations : les choix non-déterministes seront supposés aléatoires, et l'affinité égale à la valeur espérée.

La fonction aff_p est étendue aux programmes par construction inductive sur l'ensemble des opérateurs Indigolog.

$$\begin{aligned}
aff_p(\phi?) &= 0 \\
aff_p(\delta_1;\delta_2) &= aff_p(\delta_1) + aff_p(\delta_2) \\
aff_p(\delta_1|\delta_2) &= \frac{aff_p(\delta_1) + aff_p(\delta_2)}{2} \\
aff_p(\delta_1\rangle\delta_2) &= aff_p(\delta_1) \\
aff_p(\pi x.\delta) &= aff_p(\delta) \\
aff_p(\delta^*) &= aff_p(\delta) \\
aff_p(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2) &= \frac{aff_p(\delta_1) + aff_p(\delta_2)}{2} \\
aff_p(\textbf{while } \phi \textbf{ do } \delta) &= aff_p(\delta) \\
aff_p(\Sigma(\delta)) &= aff_p(\delta)
\end{aligned}$$

Les actions avec des arguments qui modélisent des verbes avec des objets doivent être définies différemment, parce que les attributs peuvent dépendre des arguments (la signification d'une action dépend des objets sur lesquels elle est appliquée). Par exemple, la même action *read* peut être *hard* quand on lit un ouvrage universitaire ou *easy* quand on lit le journal.

La fonction $attributes_a$ prend une action avec un argument $a(x)$ et donne une liste de paires $< attribut, predicat >$. Pour chacune de ces paires, $a(x)$ aura l'attribut quand x satisfera au prédicat correspondant. Cette fonction est fournie par le concepteur de l'application.

$$attributes_a(a(x)) = \{< att_1, \phi_1 >, ..., < att_n, \phi_n >\}$$

3.8 . Le processus PAGE de transformation automatique

Le processus de modification d'un programme δ se déroule en trois étapes qui sont exécutées dans l'ordre suivant :

1. transformation des actions ou procédures qui font partie d'une famille,
2. transformation de choix non déterministes de programmes,
3. transformation des choix non déterministes d'arguments.

Ce processus utilise l'affinité des actions et des programmes pour déterminer comment appliquer les transformations. Par conséquent, en plus de la définition habituelle de l'application Indigolog, le concepteur de l'application doit fournir les définitions supplémentaires suivantes :

— *attributes*, liste des attributs pour chaque action ou procédure ;

— $attributes_a$, liste des attributs en fonction des arguments ;

— *family* et $family_P$, les familles d'actions et de procédures ;

— p, profil sous forme d'une liste de valeurs pour les différents attributs.

Transformation des actions ou procédures : le programme est parcouru récursivement pour trouver les actions a_0 ou procédures p_0 et leur famille F. Chaque a_0 (resp. p_0) est remplacée par la transformation T6 (resp. T7) en un choix non-déterministe. Les actions (resp. procédures) avec une affinité négative sont écartées : nous n'introduisons les choix que pour une affinité positive. L'action (resp. procédure) d'origine est gardée seulement si son affinité est positive ou si c'est la seule option. Les actions sont groupées par affinité. Un choix non-déterministe est appliqué dans un groupe d'actions d'affinité équivalente, et un choix préférentiel est appliqué entre les groupes par la transformation T1. Les choix avec l'affinité la plus élevée seront considérés en premier. Le programme résultant peut s'écrire :

$$\rangle|_{a_i \in F | aff(a_i)=2}\ a_i\ \rangle|_{a_i \in F | aff(a_i)=1}\ a_i\ \rangle a_0$$

Les actions d'affinité 2 subissent un choix non-déterministe et elles sont préférées aux actions de d'affinité 1, elles-mêmes en choix non-déterministe dans leur groupe. Enfin, l'action d'origine est considérée en dernier recours. Cette formule est généralisable à un plus grand nombre de degrés d'affinité.

Transformation des choix non-déterministes de programmes : le programme est parcouru récursivement en mode *bottom-up* à la recherche de choix non-déterministes de programmes. Pour chaque forme $\delta_1 \mid \delta_2$ identifiée, les affinités de δ_1 et δ_2 sont comparées. La branche avec l'affinité la plus élevée est préférée en utilisant la transformation T1. Si la différence d'affinité est supérieure à une valeur de seuil fixée ici à 2, la branche avec l'affinité la plus faible est écartée en utilisant la transformation T3 au lieu de T1. Si les affinités sont identiques, le choix non-déterministe est conservé.

Transformation des choix non-déterministes d'arguments : le programme est parcouru récursivement à la recherche de choix non-déterministes d'arguments. Pour chaque forme $\pi x.\delta(x)$ identifiée :

— $\delta(x)$ est parcouru à la recherche des actions a_i qui ont x comme argument ;

— pour chaque action $a_i(x)$, la liste de prédicats $P = attributes_a(a_i)$ associés à cette action est extraite. Pour chacun des prédicats, l'attribut associé peut être vu par le profil comme positif, négatif ou neutre. Nous pouvons donc vouloir introduire le prédicat, sa négation ou ne pas l'introduire, ce qui donne $3^{|P|}$ combinaisons à étudier ;

— Pour chaque combinaison c , nous calculons l'affinité de δ en donnant à chaque action l'affinité :

$$aff_p(a(x)) = \Sigma_{<attribute,\phi>} \begin{cases} p(attribute) & \text{si } \phi \in c \\ p(attribute) & \text{si } \neg\phi \in c \\ 0 & \text{sinon.} \end{cases}$$

— Nous retenons la combinaison c_k qui maximise la valeur d'affinité de δ et nous appliquons la transformation T2 pour remplacer $\pi x.\delta(x)$ par $\pi x.\psi(x)?;\delta(x)\rangle\pi x.\delta(x)$ où $\psi(x)$ est la conjonction des prédicats de c_k. Ceci garantit que les arguments qui satisfont $\psi(x)$ seront considérés en premier lors du choix. Si l'affinité déterminée pour c_k est supérieure à un seuil (fixé ici à 2), la transformation T4 est utilisée au lieu de la transformation T2 pour rendre la condition $\psi(x)$ obligatoire. En cas d'égalité d'affinité, la combinaison comportant le plus petit nombre de prédicats est choisie.

Les trois étapes précédentes sont exécutées dans cet ordre pour un profil donné p et constituent un mécanisme d'altération automatique des programmes Indigolog.

Avec un profil standard où tous les attributs sont à 0, toutes les affinités sont à 0 et le programme est laissé inchangé.

Au final, le processus de transformation peut-être résumé par la figure 3.5. Le même processus de transformation s'applique avec un profil de personnalité de l'agent.

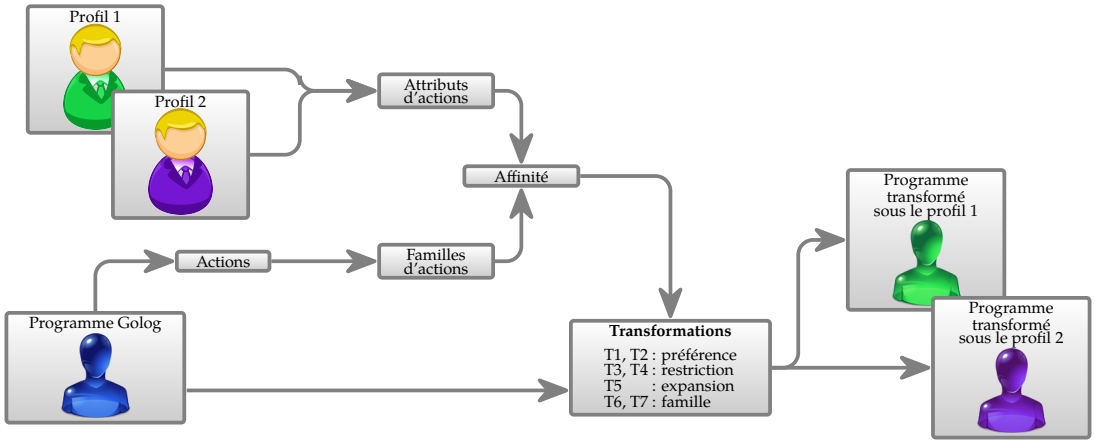


FIGURE 3.5 – Illustration du processus de transformation des programmes Golog sous l'influence de profils d'utilisateurs.

3.9 . Exemples

Nous présentons ici deux exemples très simples de l'application des transformations à un programme original en fonction de profils différents. Des exemples plus élaborés sont proposés en annexe A.

3.9.1 . Exemple de personnalisation

Soit un programme original pour un système de recommandation d'articles d'information :

$$\begin{array}{l}\textbf{while } \mathit{continue}\\ \quad \textbf{do } \pi a.(\mathit{article}(a)?; \mathrm{recommend}(a))\end{array}$$

Ce programme exécute une boucle qui propose indifféremment des articles.

Définissons les attributs suivants :

$$\begin{aligned}\mathit{attributes}_a(\mathit{recommend}(a)) = [&\langle \mathit{dealsWith}(a, \mathit{sport}), \mathit{sport}\rangle,\\ &\langle \mathit{dealsWith}(a, \mathit{economy}), \mathit{economy}\rangle]\end{aligned}$$

Considérons maintenant le profil p_1 :
$\mathrm{User}_1 : p_1 = \{\mathrm{economy} = 0, \mathrm{sport} = -2\}$
la transformation T4 est utilisée et le programme transformé est :

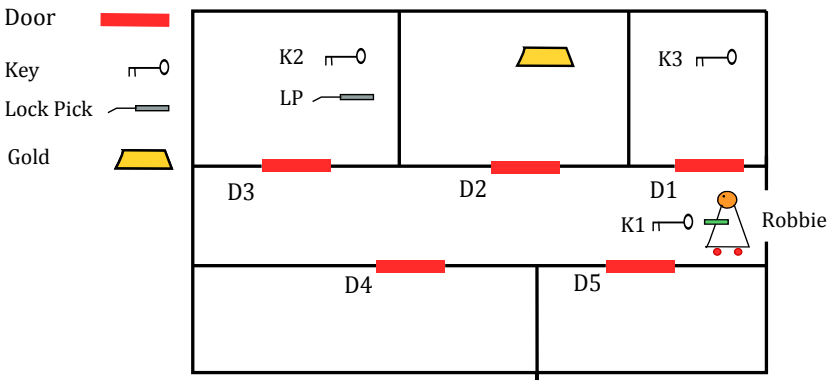
$$\begin{array}{l}\textbf{while } \mathit{continue}\\ \quad \textbf{do } \pi a.(\neg \mathit{dealsWith}(\mathit{sport}, a)?; \mathit{article}(a)?;\\ \qquad \mathrm{recommend}(a))\end{array}$$

Avec le nouveau profil p_2 : $\mathrm{User}_2 : p_2 = \{\mathrm{economy} = -1, \mathrm{sport} = 1\}$
la transformation T2 est utilisée et le programme transformé est :

$$\begin{array}{l}\textbf{while } \mathit{continue}\\ \quad \textbf{do } \pi a.(\mathit{dealsWith}(\mathit{sport}, a)?; \mathrm{recommend}(a))\\ \qquad \rangle \pi a.(\neg \mathit{dealsWith}(\mathit{economy}, a)?; \mathrm{recommend}(a))\\ \qquad \rangle \pi a.(\mathit{article}(a)?; \mathrm{recommend}(a))\end{array}$$

3.9.2 . Exemple de personnification

Soit le cas de Robbie le robot qui doit récupérer l'or derrière la porte verrouillée D2 :



Le programme original est :

proc loot(r)
 while $\exists i.position(i) = r$;
  do $\pi i.(position(i) = r)?$;
   take(i)
endProc

proc main
 while $\neg position(gold) = inventory$;
  do $\Sigma(\pi r.closed(r)?$;
   openWithKey(r);loot(r))
endProc

Le robot parcourt les salles tant qu'il n'a pas trouvé l'or ; si une salle est fermée, il l'ouvre avec une clé. Chaque fois qu'il pénètre dans une salle, il prend les objets disponibles (procédure *loot*). Le plan neutre est : openWithKey(D1), take(K3), openWithKey(D3), take(K2), take(lockpick), openWithKey(D2), take(gold).

Considérons des caractéristiques possibles de profil :

— outlaw : facette de Dutifulness du trait Conscientiousness, couvre les notions de /loyal/-ness, /virtuous/-ness, /lawabiding/-ness et /truthful/-ness ;

— brutal, est associé avec trois facettes : le pôle négatif de la facette Tender-mindedness du trait Agreeableness et les facettes Angry-Hostility et Impulsiveness du trait Neuroticism ;

— practical.

Pour utiliser le processus de transformation, nous définissons une famille autour de l'action « ouvrir une porte avec la clé » : *openDoorWithKey*, *breakDoorOpen* et *lockpickDoor*. Il faut aussi donner des attributs aux actions, en fait les traits psychologiques qui favorisent ces actions. L'attribut brutal est associé à *breakDoorOpen*, l'attribut outlaw à *lockpickDoor* et l'attribut practical à *openDoorWithKey*.

Soit le profil de personnalité p_1 d'un agent qui est hors la loi, déteste la brutalité et n'est pas spécialement pragmatique : $p_1(\text{brutal}) = -2$, $p_1(\text{outlaw}) = 1$, $p_1(\text{practical}) = 0$.

Le programme transformé devient :

proc main $- p_1$
 while $\neg position(gold) = inventory$;
  do $\Sigma(\pi r.closed(r)?$;
   lockpick(r)$\rangle$openWithKey(r);
   loot(r))
endProc

et le plan associé :

openWithKey(D1), take(K3), openWithKey(D3), take(K2), take(lockpick), lockpick(D2), take(gold).

Soit un second profil de personnalité p_2 d'un agent qui est brutal, qui n'est pas spécialement hors la loi, mais qui est très pragmatique : $p_2(\text{brutal}) = 1, p_1(\text{outlaw}) = 0, p_1(\text{practical}) = 2$.

le programme transformé devient :

$$
\begin{array}{l}
\textbf{proc}\ \text{main} - p_2 \\
\quad \textbf{while}\ \neg \mathit{position}(\mathit{gold}) = \mathit{inventory}; \\
\quad\quad \textbf{do}\ \Sigma(\pi r.\mathit{closed}(r)?; \\
\qquad\qquad \text{openWithKey}(r) \rangle \text{break}(r); \\
\qquad\qquad \text{loot}(r)) \\
\quad \textbf{endProc}
\end{array}
$$

et le plan associé :

openDoorWithKey(D1), take(K3), break(D2), take(gold).

L'annexe A décrit in-extenso deux exemples d'application de ce processus de transformation :

— le premier dans le cadre de la personnalisation d'un EIAH A.1,

— le second dans le cadre de la personnification d'un agent pour un jeu sérieux A.2.

3.10 . Conclusion

Ce chapitre introduit et illustre l'utilisation de notre formalisme et de notre processus de transformation pour créer des applications personnalisées. Les exemples décrits en annexe montrent que notre formalisme agent d'application web permet l'expression de scénarios réalistes sous forme d'agent Golog. Notre processus de transformation permet l'ajout de comportements personnalisés dans ces applications et répond aux objectifs que nous nous étions fixés. Dans le cas de l'hypermédia adaptatif, les transformations permettent aussi bien d'introduire de l'adaptation de parcours (l'utilisateur parcourt les concepts différemment en fonction de son but) que de contenu (différentes ressources sont proposées).

Contrairement à [69] qui prend en compte des valeurs continues pour les traits, nous avons discrétisé les valeurs des traits. Ceci n'est pas une limitation forte du modèle. Si l'on veut vraiment des valeurs continues, l'extension du modèle est possible en assignant des valeurs réelles aux traits du profil.

Avec ces deux exemples, nous avons montré l'intérêt de travailler par transformation des programmes : la sécurité et la prouvabilité liées à l'utilisation d'un langage formel pour définir le comportement de l'agent sont conservées. L'expression de la personnalisation et de la personnification sont simplifiées par rapport à une approche plus classique.

Production scientifique et encadrements

Co-encadrement de thèse :

— G. Dubus, « Transformation de programmes logiques : application à la personnalisation et à la personnification d'agents. », Thèse de doctorat, Supélec, 4 sept. 2014

Logiciels :

— WAIG+PAGE : implémentation d'applications Web à l'aide d'agents Golog et transformation automatique pour altérer le comportement des agents en fonction d'un profil utilisateur ou agent.

Publications :

— G. Dubus, F. Popineau, and Y. Bourda, "Situation calculus and personalized web systems", in *2011 11th International Conference on Intelligent Systems Design and Applications*, Córdoba, Spain: IEEE, Nov. 2011, pages 569–574. DOI: 10/fzrtsq

— G. Dubus, F. Popineau, and Y. Bourda, “A Formal Approach to Personalization”, in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, Boca Raton, FL, USA: IEEE Computer Society, Nov. 2011, pages 233–238. DOI: 10/fzhff2 (disponible en annexe)

— G. Dubus, F. Popineau, Y. Bourda, and J.-P. Sansonnet, “Parametric Reasoning Agents Extend the Control over Standard Behaviors”, in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 2, Atlanta, GA, USA: IEEE, Nov. 2013, pages 163–170. DOI: 10.1109/WI-IAT.2013.105 (disponible en annexe)

— F. Popineau, G. Dubus, and Y. Bourda, “Personalization and personification: A constructive approach based on parametric agents”, in *Intelligent Virtual Agents - 14th International Conference, IVA 2014*, T. W. Bickmore, S. Marsella, and C. L. Sidner, Eds., series Lecture Notes in Computer Science, volume 8637, Boston, MA, USA: Springer, Aug. 2014, pages 339–344. DOI: 10.1007/978-3-319-09767-1_44

4 - Recommandation de ressources pédagogiques

4.1 . Cours massifs en ligne

Avec la démocratisation des accès à internet, la publication de ressources éducatives libres (REL) a pris on essor [38]. Ce terme de ressource éducative libre désigne tous types de matériels pédagogiques partagés gratuitement sur internet sous des licences ouvertes ou Open License. Ces REL se présentent sous la forme d'un cours complet, d'un module, d'un objet d'apprentissage, d'une vidéo, d'un quiz et bien d'autres façons permettant de proposer de la connaissance.

Parallèlement, les pages Web sont passées d'un statut statique ou quasiment à un statut dynamique grâce au Web2.0 [54]. Bien sûr, les EIAH ont été une cible pour les nouvelles possibilités offertes par le Web2.0 [63].

C'est ainsi qu'en 2008 on a pu voir naître les *Massive Open Online Courses* ou MOOCs, résultant de la conjonction d'un mouvement entre la pédagogie ouverte, la mise à disposition d'un plus grand nombre de REL, et la technologie nouvelle des applications Web. Les MOOCs sont un cas particulier d'EIAH et sont rapidement devenus très populaires.

4.2 . Intégration d'un agent virtuel interactif

Pour concrétiser les études menées aux chapitres 2 et 3, nous avons entrepris d'intégrer un agent virtuel interactif (IVA) minimal à l'une des plateformes MOOC les plus utilisées : OpenEdX [56].

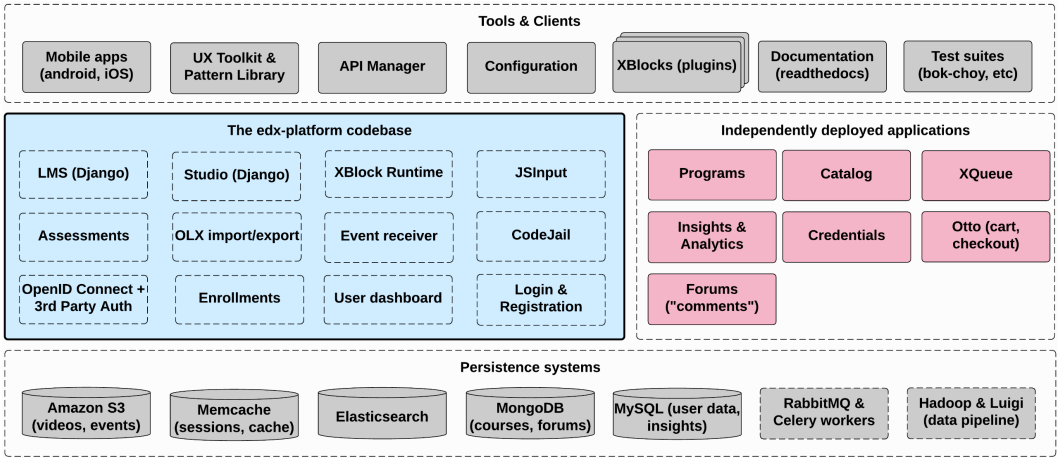


Figure 4.1 – Architecture de la plateforme OpenEdX.

Deux options étaient offertes pour construire ce démonstrateur :

1. insérer l'IVA côté serveur, ce qui peut sembler le plus naturel, puisqu'il s'agit d'ajouter l'agent à l'application OpenEdX elle-même,
2. insérer l'IVA côté client, en passant par un proxy qui permet de modifier la page web.

La première option peut sembler la plus à même de prendre en charge la mise en œuvre d'un IVA. Il serait en effet intéressant d'avoir accès à toutes les données et actions d'un utilisateur pour pouvoir agir au mieux sur ce qui lui sera proposé. Il s'avère que le code de la plateforme OpenEdX représenté à la figure 4.1 est très complexe : il repose sur plusieurs bases de données et différents modules de taille imposante. De plus, ce code étant ouvert, il est sujet à de nombreuses modifications. Insérer une API (*Application Programming Interface*) pour gérer un agent dans ce contexte est très périlleux et toute initiative n'aurait pu fonctionner

que sur une version figée de OpenEdX. Les possibilités d'extension de la plateforme sont rassemblées dans la documentation en ligne [1] et la table ci-dessous :

	Custom JavaScript Applications*	LTI	External Graders	XBlocks	Platform Customization
Development Cost	Low	Low	Medium	Medium	High
Language	JavaScript	Any	Any	Python	Python
Development Environment Needed	No	No	Yes	Yes	Yes
Self-hosting Needed	No	Yes	Yes	No	No
Need edX Involvement	No	No	Yes	Yes	Yes
Clean UI Integration	Yes	No (see LTI)	Yes	Yes	Yes
Mobile enabled	Possibly	Possibly	Yes	Yes	Yes
Server Side Grading	Possibly (See JavaScript)	Yes	Yes	Yes	Yes
Usage Data	No (See JavaScript)	No	Limited	Yes	Yes
Provision in Studio	No	No	No	Yes	No
Privacy Loss Compared to Hosting Open edX	No	Possibly	Possibly	No	No

La plateforme OpenEdX offre un mécanisme d'extension au travers des XBlocks qui permettent de définir des blocs personnalisés dans la page. Toutefois, cette solution ne nous convient pas ici, parce qu'il s'agit de composants de cours, alors que nous avons besoin de construire une expérience personnalisée au niveau du comportement de la plateforme. En revanche, les XBlocks seront utiles pour le travail présenté en section 4.3.

La seconde option présente l'avantage de ne pas dépendre du code de la plateforme elle-même, mais uniquement du fait de pouvoir instrumenter la page web délivrée. Bien sûr, il reste une dépendance au format de cette page, mais elle est beaucoup plus légère. Le principal avantage de cette solution est de pouvoir déployer cet IVA sur un cours existant et de façon locale sur le poste client de l'utilisateur.

L'architecture mise en œuvre est donnée à la figure 4.2.

1. https://edx.readthedocs.io/projects/edx-developer-guide/en/latest/extending_platform/index.html

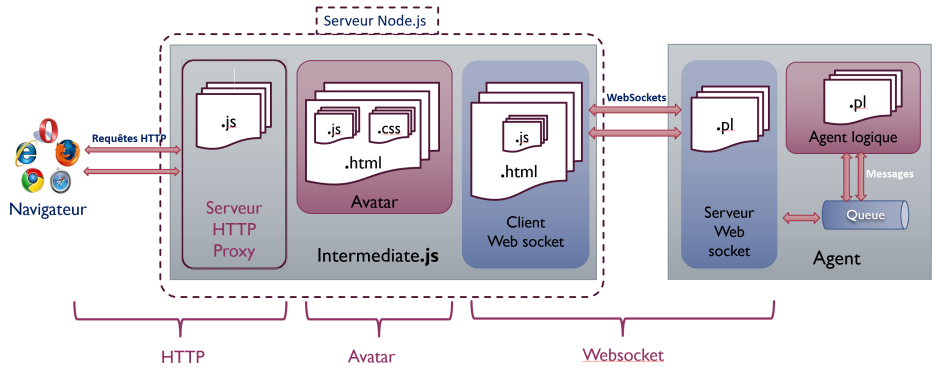


Figure 4.2 – Architecture pour l'intégration d'un IVA dans la plateforme OpenEdX.

Reste à choisir un agent virtuel pour illustrer nos concepts. Il existe des agents extrêmement sophistiqués permettant de reproduire une vaste gamme d'émotions [64]. Dans notre cas et pour nos besoins de démonstration, un agent beaucoup plus simple peut suffire. Nous avons choisi de ressusciter le célèbre Clippy de Microsoft, disponible en plusieurs habillages, dont celui de Merlin.

Clippy est disponible en Javascript. Le système consiste donc en un proxy qui intercepte la page web et insère un code Javascript qui va afficher Merlin sur la page. Le code Javascript en question dialogue par websocket avec Prolog dans lequel l'agent est implémenté.

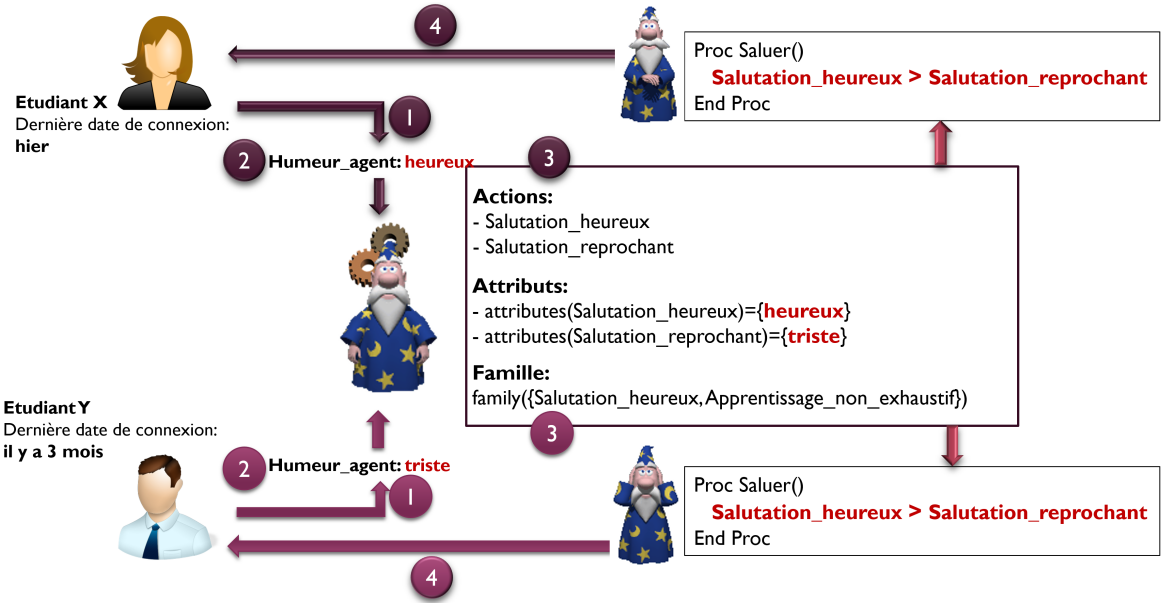


Figure 4.3 – Exemple de modification du comportement de Merlin.

La figure 4.3 illustre une situation que l'on rencontre fréquemment sur les MOOCs : le cas des utilisateurs qui ne sont pas assez assidus. L'objectif est de montrer que Merlin peut agir comme un coach et aider l'apprenant à garder sa motivation en utilisant la technique décrite au chapitre 3. Dans l'exemple illustré, la procédure de salutation est personnalisée en fonction de la date de dernière connexion de l'apprenant : si celle-ci est proche, Merlin affiche un air enjoué alors qu'il affiche un air triste si elle est lointaine.

Selon la structure du MOOC suivi, Merlin peut proposer ou orienter les choix de l'utilisateur vers différentes actions : passer un test, réviser un cours ou apprendre de nouvelles notions. Une dernière catégorie est approchée dans la section 4.3 : la recommandation de ressources pédagogiques ouvertes et liées en fonction du profil de l'utilisateur et de ses besoins identifiés.

Le démonstrateur dispose ici d'une description codée en dur du cours suivi : ensemble des ressources le composant ainsi que leurs métadonnées. Dans une situation réaliste, l'agent devrait être capable de trouver cette description par une API dédiée de la plateforme.

4.3 . Recommandation et ressources ouvertes et liées

L'explosion des offres de MOOCs a également accéléré la publication de ressources pédagogiques libres accessibles en ligne. Nous avons conduit des travaux dans le cadre de la thèse de Hiba Hajri [6] sur la recommandation personnalisée de ressources éducatives libres.

Au-delà des questions scientifiques concernant les éléments qui doivent présider à la recommandation et la meilleure façon de calculer celle-ci, la raison de présenter ces travaux ici réside dans le fait que la tâche de recommandation entre complètement dans la vision de l'EIAH piloté par un IVA comme nous allons le montrer. De par son architecture, l'ensemble du système est réalisable dans le modèle formel du calcul des situations présenté au chapitre 3 et dans le cadre du démonstrateur présenté à la section 4.2.

L'objet de ces travaux a consisté à définir un mécanisme d'identification des ressources disponibles en ligne correspondant au point précis où se trouve un apprenant dans son parcours d'un MOOC. Le point précis fait référence à la situation de l'apprenant : la ressource qu'il consulte, les prérequis de cette ressource, les objectifs d'apprentissage de cette ressource, les ressources auxquelles il a été exposé et enfin les préférences de cet apprenant.

Nous avons baptisé ce système de recommandation du nom de MORS pour *MOOC-based Open educational ressources Recommender System*. Il produit une liste classée de ressources recommandées. Son architecture est décrite 4.4 et repose sur 4 modules que nous décrivons ci-dessous.

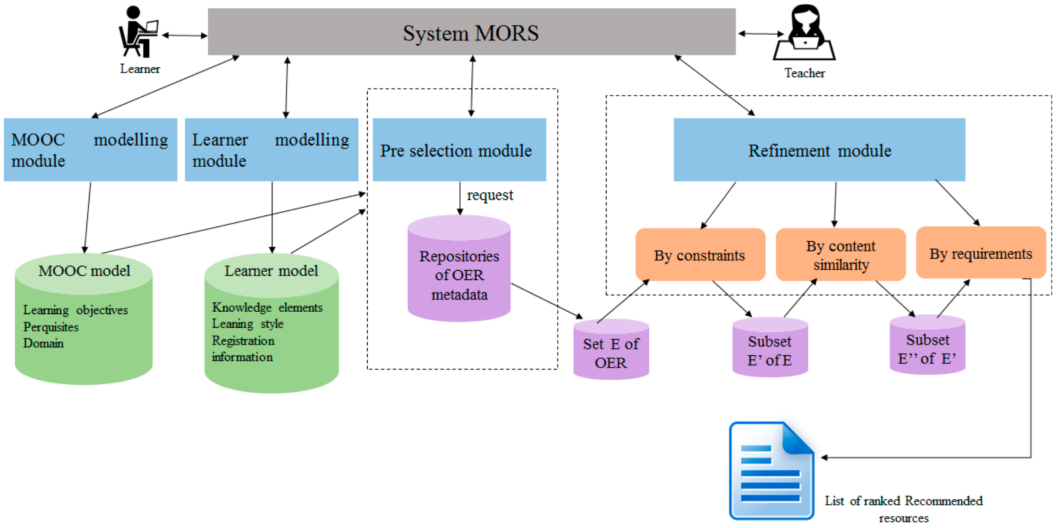


Figure 4.4 – Architecture du système MORS de recommandation de ressources pédagogiques libres en ligne dans un MOOC.

Module de modélisation du MOOC : le MOOC est un cas particulier de notre exemple générique d'EIAH (voir A.1). Le MOOC en lui-même possède des prérequis (décrits en termes de CC) d'entrée. Le MOOC est divisé en sections qui correspondent à des semaines [2] et chaque section possède des objectifs d'apprentissage. Un degré de maitrise à 3 niveaux (*beginner*, *intermediate*, *expert*) est associé aux CC. Ce module de modélisation du MOOC correspond à la modélisation du domaine de l'IVA.

Module de modélisation de l'apprenant : ce module contient différents éléments de profil,

2. Le rythme est un paramètre important dans un MOOC : les étudiants qui ne suivent pas le rythme de progression décrochent.

comme les CC maitrisées et leur degré de maitrise, mais aussi les langues maitrisées par l'apprenant. On retrouve ici le profil de l'utilisateur en interaction avec l'IVA et auquel l'IVA essaie de s'adapter en fournissant la recommandation la plus appropriée.

Module de présélection : en partant des CC en cours d'étude par l'apprenant, des points d'entrée SPARQL vers des descriptions de ressources pédagogiques libres sont interrogés à la recherche de ressources délivrant ces mêmes CC. Les paramètres de durée, de langue, de difficulté de ces ressources entre autres sont récupérés. Ce module de calcul est réalisable en Indigolog.

Module d'affinement : les ressources présélectionnées sont filtrées pour éliminer celles qui seraient trop inadéquates, puis classées par ordre de proximité avec le profil de l'apprenant et la situation courante. Ce module de calcul est également réalisable en Indigolog.

Le domaine du MOOC est représenté par un ensemble de CC composé des prérequis à l'entrée du MOOC, puis des objectifs d'apprentissage fixés par section. Initialement, l'apprenant possède un niveau de maîtrise des CC du MOOC correspondant au niveau des prérequis exigés pour suivre le MOOC. Chaque semaine ou section du MOOC, de nouveaux objectifs d'apprentissage font évoluer le niveau de maîtrise des CC par l'apprenant. Un quiz de fin de section permet de mesurer le décalage entre la maîtrise réelle et attendue des CC du MOOC.

La recommandation est déclenchée lorsqu'un tel décalage est observé : - à l'entrée du MOOC parce que l'apprenant ne présente pas un niveau suffisant par rapport aux prérequis, - à la fin d'une section, parce que l'évaluation par le quiz a identifié une ou des lacunes par rapport aux objectifs d'apprentissage.

La recommandation est fondée sur une requête par mots-clés sur des entrepôts de ressources pédagogiques ouvertes et liées. Ces mots-clés sont calculés sur la base de la description du domaine du MOOC et de la CC spécifique pour laquelle nous cherchons une ressource complémentaire. Les seuls termes fournis par l'enseignant pour la description du MOOC et des objectifs d'apprentissage ne garantiraient pas suffisamment de trouver des ressources correspondantes. La recherche est élargie en utilisant un module de détermination de synonymes à partir des termes fournis par l'enseignant, basé sur DBPedia [3] pour augmenter la probabilité de trouver des ressources pertinentes.

La liste des ressources candidates est ensuite filtrée par le module d'affinement selon plusieurs critères. Certaines contraintes doivent être respectées par une ressource candidate :

— l'apprenant doit connaître la langue dans laquelle la ressource est proposée ;

— la ressource ne doit pas nécessiter de prérequis (CC) non maîtrisés par le candidat ;

— la ressource doit apporter un niveau de maîtrise de la CC visée au moins égal à l'objectif d'apprentissage courant de cette CC.

Une fois les ressources filtrées par ces contraintes, une mesure de similarité sémantique permet de les ordonner par proximité par rapport à la requête. Enfin une dernière étape de calcul permet d'affiner encore ce classement en évaluant la pertinence de la ressource :

— par rapport à un temps d'apprentissage nécessaire : la ressource proposée ne doit pas être beaucoup plus longue que les ressources du MOOC suivi ;

— par rapport à un style d'apprentissage caractéristique de l'apprenant ;

— ou tout autre prédicat caractéristique d'un fluent de la situation courante.

Le processus de recommandation n'a pas été implémenté sur notre architecture d'IVA présentée en section précédente, car elle était en cours de développement au même moment. Ce processus de recommandation a été implémenté au-dessus de la plateforme OpenEdX grâce au mécanisme des XBlocks mentionné 38. Trois XBlocks ont été définis :

1. un XBlock pour calculer le profil du MOOC et celui de l'apprenant ;

3. http://wiki.dbpedia.org/

2. un XBlock pour calculer la recommandation à l'entrée du MOOC ;
3. un XBlock pour calculer la recommandation à la fin de la section courante.

Le calcul de la recommandation à l'entrée du MOOC est en effet différent : il faut trouver dans l'historique d'apprentissage de l'apprenant si les CC prérequises sont maîtrisées, ou bien lui faire passer un test pour le déterminer. Il faut également collecter certaines informations à son sujet comme par exemple son style d'apprentissage, le type de ressources qu'il préfère, ou tout autre paramètre pris en compte dans le calcul ultérieur.

Le mécanisme de recommandation a été évalué sur un MOOC de OpenClassrooms intitulé « Faites une base de données avec UML » selon un protocole rigoureux avec des résultats positifs [6, chap. 12 et 13].

Au-delà de ces résultats positifs, il est intéressant de regarder l'intégration de ce mécanisme dans l'architecture à base d'IVA proposée. La transposition du calcul de recommandation vers Indigolog peut s'effectuer avec différents grains :

— soit de façon monolithique en définissant une procédure externe à Indigolog, qui serait une procédure de *sensing* retournant la liste des ressources recommandées ;

— soit en transposant les étapes de préselection et de filtrage en Indigolog, ce qui permettrait de leur appliquer les mécanismes de transformation en fonction du profil de l'apprenant. Par exemple, la préférence pour un type de ressource conduirait à une procédure de filtrage spécifique pour cet apprenant.

Dans les deux cas, l'agent récupère par *sensing* une liste de ressources, soit filtrée, soit à laquelle il applique lui-même une procédure de filtrage transformée par le profil de l'utilisateur. La proposition de consultation de ressources complémentaires est déclenchée par la détection d'une lacune. L'agent a donc une opportunité pour présenter ces ressources d'une façon adaptée au profil de l'utilisateur, en utilisant différentes modalités, plutôt que de simplement les inclure de façon neutre dans la page du MOOC.

4.4 . Conclusion

Il est satisfaisant d'avoir pu développer une preuve de concept sur une plateforme telle que OpenEdX. L'idée de l'IVA servant de compagnon à l'apprenant est séduisante. La mise en oeuvre d'un tel agent nécessiterait bien sûr d'être évaluée sur un plan psycho-cognitif. D'un point de vue purement informatique, la solution proposée permet d'envisager la programmation de stratégies de personnalisation / personnification de façon simple, ou du moins claire. Une méthodologie pourrait émerger : définir le comportement neutre de l'EIAH, puis selon les caractéristiques de profil à accompagner, définir les actions spécifiques et leurs connexions avec les caractéristiques de profil. Comme nous l'avons vu, l'intérêt premier est celui de la garantie d'atteindre l'objectif par tous les parcours altérés.

Production scientifique et encadrements

Encadrement de stage de Master :

— H. Hawes, « Personnalisation et Personnification des Tuteurs Intelligents », Université de la Manouba, École Nationale des Sciences de l'Informatique, Stage de fin d'études, sept. 2015

Co-encadrement de thèse :

— H. Hajri, « Personnalisation des MOOC par la réutilisation de Ressources Éducatives Libres », Thèse de doctorat, Université Paris-Saclay (ComUE), 8 juin 2018

Logiciels :

— MORS : système pour recommander des ressources éducatives ouvertes et libres dans un MOOC.

Publications :

— H. Hajri, Y. Bourda, and F. Popineau, "Querying Repositories of OER Descriptions: The Challenge of Educational Metadata Schemas Diversity", in *Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015*, series Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015, volume LNCS, Toledo, Spain: Springer, Sep. 2015, pages 582–586. DOI: 10/gmfpf6

— H. Hajri, Y. Bourda, and F. Popineau, "MORS: A System for Recommending OERs in a MOOC", in *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, Jul. 2017, pages 50–52. DOI: 10/ghfbvb

— H. Hajri, Y. Bourda, and F. Popineau, "Personalized Recommendation of Open Educational Resources in MOOCs", in *Computer Supported Education - 10th International Conference, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Revised Selected Papers*, series Communications in Computer and Information Science, B. M. McLaren, R. Reilly, S. Zvacek, and J. Uhomoibhi, Eds., volume 1022, Cham: Springer International Publishing, 2018, pages 166–190. DOI: 10.1007/978-3-030-21151-6_9

— H. Hajri, Y. Bourda, and F. Popineau, "A System to Recommend Open Educational Resources during an Online Course:" in *Proceedings of the 10th International Conference on Computer Supported Education*, Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, Mar. 2018, pages 99–109. DOI: 10/gmfn5w (disponible en annexe)

5 - Conclusion

5.1 . Données et expérimentations

Comment se doter du profil d'un apprenant donné ? C'est une tâche probablement complexe qu'il faut mitiger avec la volonté d'exploiter rapidement les aspects de personnalité de cet apprenant. Un système tel que celui décrit possède l'avantage de reposer sur une cartographie des personnalités. Les stratégies de personnalisation sont bien sûr laissées à la discrétion des psycho-cogniticiens et enseignants concepteurs d'EIAH. On pourrait imaginer que le nouvel apprenant passe un test psychotechnique pour évaluer son profil selon la nomenclature *Five Factors Model / Neo PI-R*, ce qui permettrait d'initialiser son profil d'apprenant.

L'architecture présentée est parfaitement adaptée d'un point de vue informatique à la construction d'une plateforme stable et évolutive pour la programmation d'un agent virtuel interactif. Le *sensing* peut capter des signaux locaux dans le temps : fatigue, nervosité, désintérêt...Ces informations peuvent être prises en compte par un programme agent personnalisé.

Une question intéressante à se poser concerne l'interaction des profils de l'apprenant et de l'agent. D'un point de vue informatique, rien n'empêche d'appliquer les transformations des deux profils simultanément. Il faut porter une attention particulière aux actions et à leurs affinités avec les profils pour que les échanges entre l'agent et l'apprenant restent intéressants et féconds. Formellement, au-delà de la simple exigence de l'atteinte de l'objectif d'apprentissage, il faudrait introduire des notions d'utilité pour prendre en compte les actions modulées des deux profils face-à-face. Ceci pourrait se faire par exemple en adaptant l'algorithme expectiminimax[1]. D'un point de vue psycho-cognitif, il faudrait évaluer et valider quel type de personnalité agent peut ou doit se trouver en face d'un type donné de personnalité apprenant.

Le profil initial de l'apprenant demanderait à être réévalué régulièrement. Les interactions avec l'agent pourraient permettre de détecter des incohérences entre le profil estimé et les observations. Les recherches actuelles sont focalisées sur l'analyse de larges cohortes sur un cours donné, beaucoup moins sur le suivi du comportement des apprenants au long cours.

Dans le cadre des travaux présentés, il serait utile de suivre un apprenant au travers de plusieurs cours, pour pouvoir construire un profil de personnalité réellement utilisable sur un temps long. Une des plus grandes difficultés dans le domaine de l'application de l'IA à l'éducation réside dans l'expérimentation en grandeur nature et dans la collecte de données. La construction de profils, la captation de données en situation réelle, mais surtout leur exploitation posent des problèmes éthiques difficiles. Nous ne pouvons pas soumettre un étudiant à une expérience d'apprentissage qui lui serait défavorable. Un système EIAH basé sur un IVA ne serait probablement acceptable que dans une situation de soutien à l'enseignant et c'est certainement vers une tâche de soutien qu'il faudra orienter des travaux ultérieurs.

5.2 . Questions ouvertes

Est-ce que les IVA sont une option pour accompagner l'enseignement ? Les politiques pédagogiques pour l'enseignement des sciences préconisent largement une démarche d'investigation, centrée sur l'expérience de l'élève qui doit découvrir le domaine et ses lois. Il semble que ce choix repose sur une lecture très partielle de la littérature et que nombre de publications avec essais contrôlés aient été ignorées [73], [43]. Or certaines publications montrent au contraire que les meilleures performances sont obtenues grâce à un enseignement explicite et prescriptif. Les auteurs ajoutent qu'il est bien sûr important de garder une attitude positive envers la science. Toutefois, les activités d'investigation ne doivent pas se

1. Nous ne sommes bien sûr pas dans le cas d'un jeu adversarial, mais dans le cas d'un jeu coopératif, d'où la nécessité d'une adaptation : les deux joueurs cherchent à maximiser l'utilité globale.

dérouler au détriment de l'apprentissage des concepts. Or un enseignement prescriptif, qui nécessite d'assurer que l'apprenant conserve un engagement, semble donner des arguments pour supposer qu'un accompagnement personnalisé par un IVA pourrait être un moyen effectif pour l'apprentissage des sciences. Bien sûr, cela nécessiterait de mobiliser des experts cogniticiens, psychologues pour construire les personnalités des IVA et la manière adéquate de répondre à différentes situations pour différents apprenants. Ceci fait déjà partie de la connaissance métier de l'enseignant.

Dans [7], nous avons présenté une application sous la forme d'un exerciseur réalisable dans le cadre formel de l'IVA pour un EIAH présenté dans ce travail. Cet exerciseur a pour rôle de faire réviser des CC en sélectionnant des exercices à soumettre à l'apprenant. La politique de choix des exercices est personnalisable en fonction de l'apprenant.

Est-il nécessaire de construire une adaptation plus continue ? L'apprentissage automatique a fait rentrer l'IA dans l'ère du continu : là où GOFAI[2] était discrète, parce que symbolique, on peut se poser la question du besoin d'une adaptation plus continue.

Tout d'abord remarquons que l'approche symbolique peut aussi bénéficier d'une plus grande continuité : on peut introduire des coefficients réels pour le niveau de l'apprenant ou pour des priorités de règles et bien d'autres choses. Il reste que le fond du traitement proposé repose sur un nombre fini de choix et une combinatoire de parcours.

Le *knowledge tracing* (KT) supporté par les méthodes modernes de machine learning laisse entrevoir la possibilité d'une personnalisation très fine des activités. On passe de modèles avec très peu de paramètres à des modèles avec transformateurs qui peuvent en compter des millions. Le *knowledge tracing* ne suffit pas a piloter un EIAH, il faut l'accompagner d'une politique de recommandation qui peut elle-même être apprise. Toutefois, la voie que prend l'apprentissage automatique dans le KT n'est peut-être qu'un leurre pour deux raisons :

1. malgré les bons résultats publiés, on s'aperçoit qu'il existe des dérives d'une cohorte à une autre. Les méthodes d'apprentissage automatique sont appliquées dans des conditions telles que la généralisation n'est pas garantie[3] ;
2. les résultats d'un apprenant sont éminemment multi-factoriels et les conditions de sa réussite sont modifiables. Prétendre piloter un EIAH par un apprentissage à partir d'observations massives semble pour l'instant illusoire.

Les IVA offrent la possibilité de créer les conditions de la réussite de l'apprenant. Le *knowledge tracing* est utile pour mesurer l'évolution de l'apprenant, mais un IVA avec un comportement spécifié, reposant sur les meilleures pratiques pédagogiques, permet de construire un système d'interaction dynamique avec les apprenants, dans un cadre contrôlé.

Que ferait-on de différent aujourd'hui ? La représentation des connaissances reposerait complètement sur une approche ontologique à l'aide de OWL et un raisonnement intégré. Le couplage entre la programmation logique et la représentation logique est parfaitement maitrisé [71].

L'approche que nous avons choisie repose sur la création de variantes du programme initial : on peut apposer l'expression « à la compilation » (*compile-time*) à cette approche, par rapport à une approche qui ordonnerait les choix « à l'exécution » (*run-time*).

Ce choix se justifie par des contraintes techniques qui relèvent du génie logiciel et de la conception de l'EIAH. La compilation d'un agent adapté à un profil résulte en un code plus simple et plus facile à comprendre qu'un code qui prendrait en charge simultanément plusieurs profils (voir A.2). Malgré tout le concepteur n'aurait pas forcément à être exposé à la complexité du code sous-jacent. Dans cette veine, une option a été envisagée avec le langage λ-Prolog [61] qui par la possibilité de programmer en logique d'ordre supérieur

2. *Good Old Fashioned Artificial Intelligence*
3. Il y a une grande variabilité dans les résultats publiés dans la littérature avec les mêmes techniques.

et de manipuler des λ-termes, permettrait certainement une implémentation élégante d'un mécanisme intégré de choix à l'exécution. Cette voie n'a pas été poursuivie en raison de la diffusion assez confidentielle de λ-Prolog.

Les méthodes de l'apprentissage automatique permettent de regarder les diverses exécutions d'un programme sous un autre angle. La programmation différentielle [62] permet d'adapter un programme générique en fonction de paramètres appris. L'idée sous-jacente est à rapprocher de notre méthode de transformation bien que l'objectif soit différent. Le point de vue développé par un interpréteur Prolog différentiel serait à étudier pour éventuellement apporter une solution plus élégante à notre problème. On pourrait penser à apprendre les différentes variantes du programme neutre.

Bibliographie

[25] J. André, R. K. Furuta, and V. Quint, Eds., *Structured Documents* (The Cambridge Series on Electronic Publishing). Cambridge ; New York: Cambridge University Press, 1989, 220 pages.

[26] K. Billings and D. Moursund, "Computers in education: An historical perspective", *ACM SIGCUE Outlook*, volume 20, number 1, pages 13–24, Sep. 1988. DOI: 10.1145/382236.382854.

[27] F. Bouchet and J.-P. Sansonnet, "Classification of Wordnet Personality Adjectives in the NEO PI-R Taxonomy", page 8,

[28] F. Bouchet and J.-P. Sansonnet, "Influence of Personality Traits on the Rational Process of Cognitive Agents", presented at the Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International, IEEE Computer Society, Aug. 1, 2011, pages 81–88. DOI: 10.1109/WI-IAT.2011.170.

[29] P. Brusilovsky, "Intelligent Tutor, Environment and Manual for Introductory Programming", *Educational and Training Technology International*, volume 29, number 1, pages 26–34, Feb. 1992. DOI: 10.1080/0954730920290104.

[30] P. Brusilovsky, "Adaptive Hypermedia", *User Modeling and User-Adapted Interaction*, volume 11, number 1, pages 87–110, Mar. 1, 2001. DOI: 10.1023/A:1011143116306.

[31] P. Brusilovsky, E. Schwarz, and G. Weber, "ELM-ART: An intelligent tutoring system on world wide web", in *Intelligent Tutoring Systems*, C. Frasson, G. Gauthier, and A. Lesgold, Eds., series Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1996, pages 261–269. DOI: 10.1007/3-540-61327-7_123.

[32] P. Costa and R. Mccrae, "Neo PI-R professional manual", *Psychological Assessment Resources*, volume 396, Jan. 1, 1992.

[33] P. De Bra, G.-J. Houben, and H. Wu, "AHAM: A Dexter-based reference model for adaptive hypermedia", in *Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia : Returning to Our Diverse Roots Returning to Our Diverse Roots - HYPERTEXT '99*, Darmstadt, Germany: ACM Press, 1999, pages 147–156. DOI: 10.1145/294469.294508.

[34] G. De Giacomo, Y. Lespérance, H. J. Levesque, and S. Sardina, "IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents", in *Multi-Agent Programming*, A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. H. Bordini, Eds., Boston, MA: Springer US, 2009, pages 31–72. DOI: 10.1007/978-0-387-89299-3_2.

[35] G. De Giacomo and H. J. Levesque, "Projection using Regression and Sensors", page 6,

[36] G. De Giacomo, H. J. Levesque, and S. Sardiña, "Incremental execution of guarded theories", *ACM Transactions on Computational Logic*, volume 2, number 4, pages 495–525, Oct. 2001. DOI: 10.1145/383779.383782.

[37] I. E. Dror, "Technology enhanced learning: The good, the bad, and the ugly", *Pragmatics & Cognition*, volume 16, number 2, pages 215–223, Jul. 24, 2008. DOI: 10.1075/pc.16.2.02dro.

[38] "Forum on the Impact of Open Courseware for Higher Education in Developing Countries, UNESCO, Paris, 1-3 July 2002: Final report - UNESCO Bibliothèque Numérique".

[39] C. Fritz and S. A. McIlraith, "Decision-Theoretic Golog with Qualitative Preferences", in *Proceedings of the Tenth International Conference*, P. Doherty, J. Mylopoulos, and C. Welty, Eds., The Lake District, UK: AAAI Press, 2006, pages 153–163.

[40] L. R. Goldberg, "An alternative "description of personality": The big-five factor structure", *Journal of Personality and Social Psychology*, volume 59, number 6, pages 1216–1229, Dec. 1990. DOI: 10/dhjp2v.

[41] M. Grandbastien et J.-M. Labat, *Environnements informatiques pour l'apprentissage humain* (IC2). Paris : Hermès sciences publications Lavoisier, 2006.

[42] P. Hayes, "What the Frame Problem is and Isn't", in *The Robot's Dilemma*, Z. W. Pylyshyn, Ed., Ablex, 1987.

[43] P. Kirschner, J. Sweller, and R. Clark, "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching", *Educational Psychologist*, volume 41, Jun. 1, 2006. DOI: 10.1207/s15326985ep4102_1.

[44] K. R. Koedinger, A. T. Corbett, and C. Perfetti, "The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning", *Cognitive Science*, volume 36, number 5, pages 757–798, Jul. 2012. DOI: 10.1111/j.1551-6709.2012.01245.x.

[45] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR: An architecture for general intelligence", *Artificial Intelligence*, volume 33, number 1, pages 1–64, Sep. 1, 1987. DOI: 10.1016/0004-3702(87)90050-6.

[46] R. Larose, « Enseignement géré par télématique », *Revue des sciences de l'éducation*, tome 15, numéro 3, page 455, 1989. DOI : 10.7202/900644ar.

[47] A. M. Lesgold and P. J. Durlach, *Adaptive Technologies for Training and Education*. Cambridge University Press, 2012, 380 pages.

[48] Y. Lespérance, H. J. Levesque, F. Lin, and R. B. Scherl, "Ability and Knowing How in the Situation Calculus", *Studia Logica*, volume 66, number 1, pages 165–186, Oct. 1, 2000. DOI: 10.1023/A:1026761331498.

[49] H. Levesque, F. Pirri, and R. Reiter, "Foundations for the Situation Calculus", *Electronic Transactions on Artificial Intelligence*, volume 2, number 3-4, pages 159–178, 1998.

[50] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl, "GOLOG: A logic programming language for dynamic domains", *The Journal of Logic Programming*, volume 31, number 1-3, pages 59–83, Apr. 1997. DOI: 10/bdtvts.

[51] J. McCarthy and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in *Readings in Artificial Intelligence*, Elsevier, 1981, pages 431–450. DOI: 10.1016/B978-0-934613-03-3.50033-7.

[52] T. J. Muller, A. Heuvelink, K. van den Bosch, and I. Swartjes, "Glengarry glen ross: Using BDI for sales game dialogues", M. O. Riedl and G. Sukthankar, Eds., 2012.

[53] C. Mulwa, S. Lawless, M. Sharp, I. Arnedillo-Sanchez, and V. Wade, "Adaptive educational hypermedia systems in technology enhanced learning: A literature review", page 12, DOI: 10.1145/1867651.1867672.

[54] S. Murugesan, "Understanding Web 2.0", *IT Professional*, volume 9, number 4, pages 34–41, Jul. 2007. DOI: 10.1109/MITP.2007.78.

[55] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*, 2nd edition. Cambridge University Press, Sep. 25, 2017.

[56] B. Porter, M. Haseltine, and N. Batchelder, "2015 Open edX Presentations - Confluence", in *The Open EdX Conference*, 2015.

[57] Z. W. Pylyshyn, *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Ablex, 1987.

[58] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language", in *In MAAMAW '96: Proceedings of the 7th European*, Springer-Verlag, 1996, pages 42–55.

[59] R. Reiter, "The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression", in *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honour of John McCarthy*, V. Lifshitz, Ed., San Diego, California, USA: Academic Press Professional, Inc., 1991, pages 359–380.

[60] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, Mass: MIT Press, 2001, 424 pages.

[61] O. Ridoux, *Lambda-Prolog de A à Z... ou presque*. Rennes, France, 1998, 163 pages.

[62] T. Rocktäschel and S. Riedel, "End-to-end Differentiable Proving", I. Guyon *et al.*, Eds., pages 3788–3800, 2017.

[63] H. Rollett, M. Lux, M. Strohmaier, G. Dosinger, and K. Tochtermann, "The Web 2.0 way of learning with technologies", *International Journal of Learning Technology*, volume 3, number 1, pages 87–107, Feb. 1, 2007. DOI: 10.1504/IJLT.2007.012368.

[64] F. de Rosis, C. Pelachaud, I. Poggi, V. Carofiglio, and B. D. Carolis, "From Greta's mind to her face: Modelling the dynamics of affective states in a conversational embodied agent", *International Journal of Human-Computer Studies*, Applications of Affective Computing in Human-Computer Interaction, volume 59, number 1, pages 81–118, Jul. 1, 2003. DOI: 10.1016/S1071-5819(03)00020-X.

[65] S. J. Russell, P. Norvig, and E. Davis, *Artificial Intelligence: A Modern Approach* (Prentice Hall Series in Artificial Intelligence), 3rd ed. Upper Saddle River: Prentice Hall, 2010, 1132 pages.

[66] S. Sardina and Y. Lespérance, "Golog Speaks the BDI Language", in *Programming Multi-Agent Systems*, L. Braubach, J.-P. Briot, and J. Thangarajah, Eds., redacted by D. Hutchison *et al.*, volume 5919, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pages 82–99.

[67] S. Sardina and L. Padgham, "Goals in the context of BDI plan failure and planning", in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS '07*, Honolulu, Hawaii: ACM Press, 2007, page 1. DOI: 10.1145/1329125.1329134.

[68] J. F. Sowa, "Conceptual graphs", in *Handbook of Knowledge Representation*, series Foundations of Artificial Intelligence, F. van Harmelen, V. Lifschitz, and B. W. Porter, Eds., volume 3, Elsevier, 2008, pages 213–237.

[69] K. Van den Bosch, A. Brandenburgh, T. J. Muller, and A. Heuvelink, "Characters with Personality!", in *Intelligent Virtual Agents*, Y. Nakano, M. Neff, A. Paiva, and M. Walker, Eds., redacted by D. Hutchison *et al.*, volume 7502, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pages 426–439.

[70] M. Wermelinger, "Conceptual Graphs and First-Order Logic", in *Conceptual Structures: Applications, Implementation and Theory, Third International Conference on Conceptual Structures*, series Lecture Notes in Computer Science, volume 954, Santa Cruz, California, USA: Springer, 1995, pages 323–337. DOI: 10.1007/3-540-60161-9_47.

[71] J. Wielemaker, W. Beek, M. Hildebrand, and J. van Ossenbruggen, "ClioPatria: A SWI-Prolog infrastructure for the Semantic Web", *Semantic Web*, volume 7, number 5, A. Polleres, Ed., pages 529–541, Jun. 23, 2016. DOI: 10.3233/SW-150191.

[72] H. Wu, "A Reference Architecture for Adaptive Hypermedia Systems", Technische Universiteit Eindhoven, 2002, 173 pages.

[73] L. Zhang, P. Kirschner, W. Cobern, and J. Sweller, "There is an Evidence Crisis in Science Educational Policy", *Educational Psychology Review*, Nov. 6, 2021. DOI: 10.1007/s10648-021-09646-1.

A - Exemples de transformation de programmes agents

A.1 . Exemple de personnalisation d'un EIAH

Cet exemple a été développé dans le cadre de la thèse de Georges Dubus [5].

Nous présentons ici un exemple de personnalisation d'un agent destiné à piloter un EIAH en utilisant les concepts précédents. L'agent accompagne un apprenant dans l'apprentissage d'un cours. L'apprenant dispose d'une latitude de choix sur l'ordre dans lequel il va consulter et travailler les différents grains pédagogiques. Ces grains pédagogiques peuvent être des définitions, des explications, des exemples ou encore des exercices.

L'agent permet à l'apprenant de passer à une nouvelle section du cours lorsque suffisamment de grains de la section courante ont été vus, ou lorsque l'apprenant réussit un exercice validant la section courante.

Si l'agent détecte que l'apprenant passe trop vite sur une ressource, il présente un exercice pour évaluer si la compétence liée à la ressource est acquise. Si c'est le cas, l'agent amène l'apprenant à la section suivante du cours, sinon il lui propose de faire une pause.

Les grains pédagogiques sont liés à des concepts ou composantes de connaissance (KC pour *knowledge component*). Ces concepts sont organisés en un graphe grâce à une relation de dépendance.

La modélisation du domaine repose sur les fluents suivants :

— $concept(c)$ indique que c est un concept du cours ;

— $prereq(c, c')$ indique que c est un prérequis de c' et doit donc être délivré avant ce dernier ;

— $definition(r)$ indique que la ressource r est une définition ;

— $explaination(r)$ indique que la ressource r est une explication ;

— $example(r)$ indique que la ressource r est un exemple ;

— $exercice(r)$ indique que la ressource r est un exercice ;

— $about(r, c)$ indique que la ressource r concerne le concept c.

En dehors du modèle logique, chaque concept dispose d'un nom d'affichage et chaque ressource d'un contenu que le proxy peut afficher. La réalisation des exercices est prise en charge par le proxy qui signale simplement à l'agent si l'exercice a été réussi. L'application comporte quatre pages :

— Une page permettant d'afficher une liste de concept. L'agent envoie une liste de concepts avec des priorités, puis le proxy renvoie le choix de l'utilisateur. Les priorités permettent de guider l'utilisateur tout en lui laissant un choix large ; elles amènent des affichages différents (mis en valeur, normal, estompé).

— Une page permet d'afficher une définition, une explication ou un exemple. Le proxy informe l'agent si l'utilisateur a sauté la ressource plutôt que de la lire.

— Une page permet d'afficher un exercice. Le proxy envoie le résultat à l'agent quand l'exercice est terminé.

— Une page suggérant à l'utilisateur de faire une pause. Le proxy rend la main à l'agent une fois que la pause est terminée.

La définition du proxy correspondante est la suivante :

$$\begin{aligned}[\langle &showConcepts(list), conceptsPage, [askConcept]\rangle,\\ &\langle showResource(r), resourcePage, [askSkipped]\rangle,\\ &\langle showExercice(r), exercicePage, [askResult]\rangle,\\ &\langle suggestPause, suggestPausePage, [waitDuringPause]\rangle].\end{aligned} \tag{A.1}$$

L'agent dispose d'un fluent $seen(r)$ permettant de savoir si l'utilisateur a déjà vu une ressource r. L'action $markSeen(r)$ permet de modifier ce fluent. Les définitions sont immédiates.

Le niveau de connaissance d'un concept c est représenté par un nombre entre 0 et 1. Il augmente à chaque ressource lue et à chaque exercice réussi. Le fluent $level(c)$ contient ce niveau, et $incrLevel(c, v)$ l'incrémente à la valeur v. Le concept est considéré comme connu ($learned(c)$) si le niveau de connaissance atteint 1.

$$Poss(incrLevel(c, v)) \equiv concept(c) \wedge number(v) \tag{A.2}$$

$$\begin{aligned} level(c, do(a, s)) = l \equiv \\ a = incrLevel(c, v) \\ \wedge\, l = max(level(c, s) + v, 1) \\ \vee\, a \neq incrLevel(c, \_) \wedge l = level(c, s) \end{aligned} \tag{A.3}$$

$$\forall c\ concept(c) \implies level(c, s_0) = 0 \tag{A.4}$$

$$learned(c, s) \equiv level(c, s) = 1 \tag{A.5}$$

Pour construire la liste des concepts, l'agent dispose d'un fluent fonctionnel *concepts* qui a pour valeur une liste de paires de concepts – priorités. Les priorités possibles sont *high*, *medium* et *low*. L'agent peut manipuler ce fluent avec les actions *clearConcepts*, $addHigh(c)$, $addMedium(c)$ et $addLow(c)$. Leurs définitions sont les suivantes :

$$Poss(clearConcepts, s) \equiv True, \tag{A.6}$$

$$\begin{aligned} Poss(addHigh(c), s) \equiv Poss(addMedium(c), s) \equiv Poss(addLow(c), s) \\ \equiv concept(c) \wedge \neg\exists p\ (c, p) \in concepts(s), \end{aligned} \tag{A.7}$$

$$\begin{aligned} concepts(do(a, s)) = v \equiv \\ v' = concepts(s) \wedge [\\ a = addHigh(c) \wedge v = v' + \langle c, high \rangle \\ \vee\, a = addMedium(c) \wedge v = v' + \langle c, medium \rangle \\ \vee\, a = addLow(c) \wedge v = v' + \langle c, low \rangle \\ \vee\, a \neq addHigh(\_) \wedge a \neq addMedium(\_) \wedge a \neq addLow(\_) \wedge v = v'. \end{aligned} \tag{A.8}$$

Le programme de l'agent est structuré en plusieurs procédures. Les principales procédures sont *makeConceptsList*, qui crée la liste de concepts à envoyer à l'utilisateur, et $handle(c)$ qui fait tout ce qui est nécessaire pour enseigner c à l'utilisateur. Le programme principal est :

$$\begin{aligned} &clearConcepts; makeConceptsList; sendConcepts(concepts); \\ &askConcept; handle(result(askConcept)). \end{aligned} \tag{A.9}$$

La procédure *makeConceptsList* parcourt tous les concepts qui ne sont pas encore connus de l'utilisateur et les ajoute à la liste avec la priorité moyenne si tous les prérequis sont connus et la priorité basse si ce n'est pas le cas. Le fluent $ready(c)$ permet de savoir si tous les prérequis sont connus.

$$ready(c, s) \equiv \forall c'\ prereq(c', c) \implies learned(c', s) \tag{A.10}$$

```
proc makeConceptsList
  while ∃c ¬learned(c) ∧ ¬c ∈ concepts
    πc ¬learned(c)?;
    if ready(c)
      addMedium(c)
    else
      addLow(c)
    endIf
  endWhile
endProc
```
(A.11)

La procédure *handle*(*concept*) choisit des ressources et les affiche jusqu'à ce que le concept soit connu.

```
proc handle(c)
  while ¬known(r)
    do πc about(r, c) ∧ ¬seen(r)?; present(r)
endProc
```
(A.12)

La procédure *present*(*resource*) appelle simplement *presentExercice* ou *presentOther* en fonction du type de la ressource.

```
proc present(r)
  if example(r) then presentExercice(r) else presentOther(r) endIf
endProc
```
(A.13)

La procédure *presentExercice* envoie un exercice à l'utilisateur. Si l'exercice est réussi, il est marqué comme *seen* pour ne plus être affiché, et le niveau d'apprentissage du concept est mis à jour.

```
proc presentExercice(e)
  showExercice(e); askResult;
  if result(askResult) then markSeen(e); updateKnowledge(e) else endIf
endProc
```
(A.14)

La procédure *updateKnowledge*(*resource*) met à jour le niveau de connaissance pour le concept de *resource*.

```
proc updateKnowledge(r)
  πc about(r, c)?; incrLevel(c, 0.25)
endProc
```
(A.15)

La procédure *presentOther*(*resource*) présente une ressource puis met à jour le niveau de connaissance. Si l'utilisateur n'a pas lu la ressource, alors un exercice est proposé pour déterminer si le concept est connu, ou si l'utilisateur est juste lassé.

```
proc presentOther(r)
  showResource(r); askSkipped;
  if result(askSkipped) then
    πc about(r, c)?; boredomExercice(c)
  else
    markSeen(r); updateKnowledge(r)
  endIf
endProc
```
(A.16)

Enfin, la procédure *boredomExercice*(*concept*), déclenchée quand l'utilisateur saute une ressource sans la lire, propose un exercice : si l'exercice est réussi, le concept est considéré connu, sinon, on propose à l'utilisateur de faire une pause.

```
proc boredomExercice(c)
  πr about(r, c);
  showExercice(e); askResult;
  if result(askResult) then
    markSeen(e); incrLevel(c, 1)
  else
    suggestPause; waitDuringPause
  endIf
endProc
```
(A.17)

Pour personnaliser cette application, nous prenons en compte les facteurs suivants.

1. Le but : l'utilisateur souhaite apprendre un concept et seuls les concepts nécessaires à ce but seront vus.
2. La vitesse d'apprentissage : certains utilisateurs ont besoin de passer beaucoup de temps sur chaque concept, d'autres moins.
3. La capacité d'abstraction : selon sa manière d'apprendre, un utilisateur peut préférer des définitions, des explications, ou des exemples.

Tout d'abord, nous souhaitons permettre à l'utilisateur de choisir un concept à apprendre : l'application mettra en avant les ressources permettant d'atteindre ce concept.

Nous définissons un prédicat $prereq^*$ permettant de savoir si un concept mène à un autre : il s'agit de la clôture transitive de *prereq*.

$$prereq^*(c, c') \equiv c = c' \vee \exists c''\ prereq(c, c'') \wedge prereq^*(c'', c') \tag{A.18}$$

Enfin, nous définissons les attributs de l'action *addMedium* de manière à ce qu'elle soit favorisée pour les concepts voulus. L'attribut du profil correspondant à un but est $goal(c)$.

$$variableAttributes(addMedium(c)) = \{\langle prereq^*(c, g), goal(g)\rangle \mid \forall g\ concept(g)\} \tag{A.19}$$

Un fois les attributs définis, nous pouvons appliquer le processus de transformations sur la procédure *makeConceptsList*. Par exemple, un utilisateur souhaitant apprendre l'algorithme de Dijkstra aura pour profil $\{goal(dijkstra) : +2\}$. La procédure transformée sera :

```
proc makeConceptsList
  while ∃c ¬learned(c) ∧ ¬c ∈ concepts
    πc ¬learned(c)∧prereq*(c, dijkstra)?;
    if ready(c)
      addMedium(c)
    else
      addLow(c)
    endIf
  endWhile
endProc.
```
(A.20)

Certains utilisateurs ont besoin de passer plus de temps sur des concepts, d'autres moins. Dans le profil, la capacité d'apprendre rapidement est représentée par l'attribut *fastLearner* (et un besoin de plus de temps est représenté par son opposé, *slowLearner*).

Nous définissons des variantes de la procédure *updateKnowledge*(r) pour les différentes vitesses d'apprentissage : une personne apprenant rapidement ne verra que deux ressources par concept, une personne apprenant lentement en verra 6. Ces variantes sont réunies dans une famille, et chacune est associée à une valeur de *fastLearner*.

$$\begin{array}{l}\textbf{proc}\ \mathit{fastUpdateKnowledge}(r)\\ \quad \pi c\ \mathit{about}(r,c)?; \mathit{incrLevel}(c, 0.5)\\ \textbf{endProc}\end{array} \tag{A.21}$$

$$\begin{array}{l}\textbf{proc}\ \mathit{slowUpdateKnowledge}(r)\\ \quad \pi c\ \mathit{about}(r,c)?; \mathit{incrLevel}(c, 0.17)\\ \textbf{endProc}\end{array} \tag{A.22}$$

$$\mathit{family}(\{\mathit{updateKnowledge}, \mathit{fastUpdateKnowledge}, \mathit{slowUpdateKnowledge}\}) \tag{A.23}$$

$$\mathit{attributes}(\mathit{slowUpdateKnowledge}(r)) = \{\mathit{slowLearner}\} \tag{A.24}$$

$$\mathit{attributes}(\mathit{fastUpdateKnowledge}(r)) = \{\mathit{fastLearner}\} \tag{A.25}$$

Le processus de transformation se fait au sein des procédures *presentExercice* et *presentOther*. C'est l'étape de transformation des familles d'actions (section 3.8) qui opère ici. Les stéréotypes d'apprenants rapides, lents et normaux sont traduits respectivement par les valeurs de profil $+1$, -1 et 0.

Pour le profil neutre $\{\mathit{fastLearner} : 0\}$, aucune action de la famille d'action n'a d'affinité positive : aucune transformation n'est opérée.

Pour l'apprenant rapide, *fastUpdateKnowledge* est préféré à *updateKnowledge* et la procédure *presentExercice* est transformée ainsi (il en va de même pour *presentOther*) :

$$\begin{array}{l}\textbf{proc}\ \mathit{presentExercice}(e)\\ \quad \mathit{showExercice}(e); \mathit{askResult};\\ \quad \textbf{if}\ \mathit{result}(\mathit{askResult})\ \textbf{then}\\ \qquad \mathit{markSeen}(e); \underline{\mathit{fastUpdateKnowledge}(e)\rangle\mathit{updateKnowledge}}(e)\ \textbf{else}\ \textbf{endIf}\\ \textbf{endProc}.\end{array} \tag{A.26}$$

De la même manière, pour un apprenant lent, la procédure *slowUpdateKnowledge* est préférée à la procédure *updateKnowledge*, et la procédure *presentExercice* est transformée ainsi :

$$\begin{array}{l}\textbf{proc}\ \mathit{presentExercice}(e)\\ \quad \mathit{showExercice}(e); \mathit{askResult};\\ \quad \textbf{if}\ \mathit{result}(\mathit{askResult})\ \textbf{then}\\ \qquad \mathit{markSeen}(e); \underline{\mathit{slowUpdateKnowledge}(e)\rangle\mathit{updateKnowledge}}(e)\ \textbf{else}\ \textbf{endIf}\\ \textbf{endProc}.\end{array} \tag{A.27}$$

Selon sa capacité d'abstraction, un utilisateur préférera des définitions, des explications ou des exemples. Cela est traduit dans le profil par des valeurs $+1$ et -1 pour les attributs *wantsDef*, *wantsExpl* et *wantsExam*. Ces attributs sont liés à la procédure *present*(r) en fonction de la valeur de r :

$$\begin{array}{l}\mathit{variableAttributes}(\mathit{present}(r)) = \{\\ \quad \langle \mathit{definition}(r), \mathit{wantsDef}\rangle,\\ \quad \langle \mathit{explaination}(r), \mathit{wantsExpl}\rangle,\\ \quad \langle \mathit{example}(r), \mathit{wantsExam}\rangle\}.\end{array} \tag{A.28}$$

La procédure *handle* sera modifiée en fonction du profil. Par exemple, pour un utilisateur préférant les explications et les exemples (profil $\{wantsExpl : +1, wantsExam : +1\}$), le processus de transformation générera la procédure suivante :

proc *handle*(*concept*)
  while $\neg$*known*(*resource*)
    do π*concept about*(*resource*, *concept*) $\wedge$ $\neg$*seen*(*resource*)?;
      explaination(*resource*) $\vee$ *definition*(*resource*)?; *present*(*resource*) (A.29)
  $\rangle$
    do π*concept about*(*resource*, *concept*) $\wedge$ $\neg$*seen*(*resource*)?; *present*(*resource*)
endProc.

A.2 . Exemple de personnification d'un agent

Cet exemple a été développé dans le cadre de la thèse de Georges Dubus [5] et publié dans [12]. Pour cet exemple, nous avons utilisé un modèle plus élaboré pour les valeurs de profil. Au lieu de se cantonner à des valeurs dans l'intervalle -2 à $+2$ comme dans la section 3.7, nous utilisons aussi un modèle à 5 valeurs, mais qui permet de lier un attribut à son opposé. Nous utiliserons un opérateur de négation et nous désignerons par *neg*(*extravert*) l'attribut opposé à l'attribut *extravert*. Les valeurs d'attribut seront *req*$-$ et *req*$+$ pour indiquer un rejet ou un besoin du trait, *pref*$-$ et *pref*$+$ pour indiquer la répulsion ou l'inclination pour le trait et *neutral* pour indiquer une indifférence pour le trait. Ce modèle est illustré ci-dessous à la figure A.1.

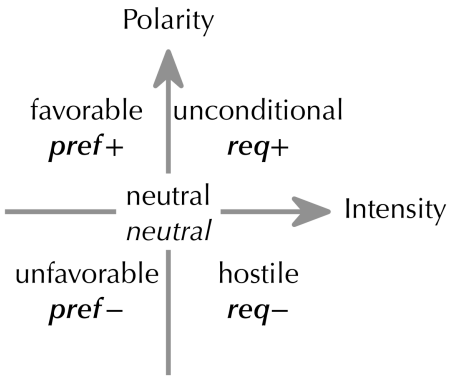


Figure A.1 – Système de valeurs de profil à 2 dimensions.

Dans cette section, nous présentons un scénario d'agent virtuel intelligent inspiré d'un cas d'utilisation réel[69]. Ce scénario ne se déroule pas dans le cadres des EIAH. Malgré tout, ce scénario est intéressant parce qu'il a été publié et qu'il nous permet d'illustrer notre approche en montrant ses avantages en terme d'expressivité.

Le scénario en question a trait à la personnification d'un agent dans une conversation. Dans un jeu dédié à l'entraînement des compétences de communication, un joueur joue le rôle d'un agent immobilier qui doit convaincre un personnage non-joueur (PNJ), joué par un agent virtuel, de visiter une maison. Lors d'une conversation, le joueur peut poser des questions pour découvrir les souhaits du PNJ et peut communiquer des informations, des interprétations et des opinions dans le but le convaincre que la maison répond à ses attentes, en mettant en valeur les qualités de la maison qui font écho aux souhaits de l'acheteur. Le PNJ peut également prendre l'initiative dans le dialogue et poser des questions pour obtenir les informations qu'il souhaite, donner des opinions, et prendre la décision de visiter la maison ou mettre fin à la conversation.

Le PNJ dispose également d'une personnalité. Les deux traits modélisés sont l'*extraversion* et l'*agréabilité* du modèle *Five Factors Model* [32]. Ces traits sont modélisés à un niveau grossier, sans considérer les sous-traits qui composent ces traits. La personnalité a un grand nombre de conséquences. Par exemple, un personnage extraverti a tendance à donner des informations tandis qu'un personnage introverti a tendance à poser des questions. Un personnage extraverti exprime des souhaits et des opinions tandis qu'un personnage introverti préfère énoncer des faits. Un personnage agréable parlera de la maison ou de l'environnement plutôt de que de lui-même. Il cherchera à continuer les conversations alors qu'un personnage désagréable aura tendance à changer de sujet de conversation sans prévenir. Un personnage désagréable aura également tendance à rejeter les interprétations de ses interlocuteurs. [69] présente un grand nombre de conséquences de la personnalité dans ce scénario ; nous en avons sélectionné deux pour illustrer notre formalisme.

Dans [52], le PNJ est piloté par un agent BDI dont les croyances sont les connaissances que le PNJ a sur la maison en question, dont les buts sont l'obtention d'informations et la prise de décision, et dont les plans sont des stratégies de conversation permettant d'atteindre les buts. La personnalité du PNJ est implémentée de manière simple : les comportements à adopter sont donnés sous la forme d'expressions conditionnelles dépendant du profil psychologique de l'agent (la table A.1 illustre la manière dont cela est formalisé).

```
if (agreeable) then              /* Boolean
  if (extravert) then            /* Boolean
    if (probability .5 ) then    /* With equal chance:
      return: tell(wish)         /* Tell a wish
    else
      return: tell(opinion)      /* Tell an opinion
    end if
  else                           /* Act introvert:
    return: tell(fact)           /* Tell fact about the house
  end if
else                             /* Act non-agreeable:
  return: tell(fact)             /* Tell fact about itself
end if
```

Table A.1 – Pseudocode exprimant la sélection du type d'information

Nous avons choisi d'illustrer notre formalisme avec cet exemple pour plusieurs raisons. D'une part, les publications présentant ce scénario contiennent des évaluations montrant l'utilité de la personnification de l'agent dans ce cas. D'autre part, un agent exprimé avec BDI peut être facilement traduit en un agent exprimé en Golog en gardant la même architecture[66].

Enfin, ce travail est un bon exemple de la complexité que notre approche cherche à résoudre : l'expression nécessaire pour prendre en compte deux traits sur une seule action (table A.1) est complexe, et croît exponentiellement avec le nombre de traits à gérer. Notre approche permet de définir les interactions entre les traits [1] et les actions et fournit, pour un profil donné, une transformation du programme appropriée pour le profil, comme illustré par la figure A.2.

Une ontologie de prédicats regroupe les différents sujets pertinents pour cette conversation. Par exemple, un prédicat *KitchenSurface* représente la connaissance de la superficie de la cuisine. Ces prédicats sont organisés de façon hiérarchique, ce qui permet à l'agent de connaître les rapports entre les sujets de conversation. Par exemple, le prédicat *House* a pour sous-prédicat *Kitchen*, qui a lui-même pour sous-prédicat *KitchenSurface*, ce qui signifie que parler de la surface de la cuisine, c'est parler de la cuisine et de la maison, et que l'opinion de l'agent de la

1. Dans notre approche, nous les appelons attributs

surface de la cuisine influe sur son opinion de la maison de manière générale.

Une ontologie d'éléments de conversation contient les différents types d'éléments qui peuvent être crus, transmis ou reçus par l'agent. Ces types d'éléments sont expliqués ci-dessous.

Fait Un fait indique une connaissance absolue de la vérité d'un prédicat. Par exemple, le fait que la superficie de la cuisine est de 12 mètres carrés s'exprime ainsi :

$$Fact(KitchenSurface, 12). \tag{A.30}$$

Interprétation Une interprétation indique une valeur subjective pour un prédicat. Par exemple, l'interprétation selon laquelle la superficie de la cuisine est grande s'exprime ainsi :

$$Interpretation(KitchenSurface, large). \tag{A.31}$$

Lorsque le vendeur transmet une interprétation à l'acheteur, celui-ci peut l'accepter ou la refuser.

Opinion Une opinion représente la disposition vis-à-vis d'un certain prédicat. Elle est exprimée par un nombre entre 0 et 1, 0 signifiant « très négatif » et 1 « très positif ». Par exemple, l'opinion que la cuisine satisfait à 85 pour cent les besoins de l'agent s'exprime ainsi :

$$Opinion(Kitchen, 0.85). \tag{A.32}$$

Souhait Un souhait indique la ou les valeurs que l'acheteur considère comme idéales pour une maison. Si un agent désire une cuisine ayant une superficie entre 11 et 20 mètres carrés, on écrira :

$$Wish(KitchenSurface, [11, 20]). \tag{A.33}$$

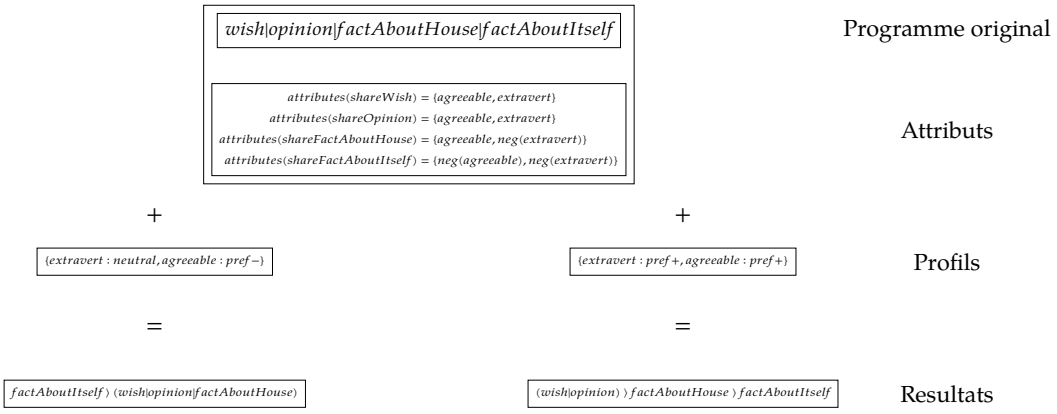


Figure A.2 – Notre approche

Enfin, une ontologie des messages formalise la communication. Les types de messages sont *Tell*, *Ask* et *Acknowledge*.

Un message *Tell* est utilisé pour transmettre une information (que ce soit un fait, une interprétation, une opinion ou un souhait). Par exemple, le message informant que la cuisine fait 12 mètres carrés s'exprime ainsi :

$$Tell(Fact(KitchenSurface, 12)). \tag{A.34}$$

Un message *Ask* est utilisé pour demander une information. Par exemple, le message demandant le souhait de l'acheteur concernant la superficie de cuisine s'exprime :

$$Ask(Wish(KitchenSurface)). \tag{A.35}$$

Un message *Acknowledge* est utilisé pour confirmer qu'une information a été reçue, en exprimant éventuellement une opinion. Par exemple, le message indiquant que l'acheteur a reçu l'interprétation du joueur, mais la rejette, s'exprime :

$$Acknowledge(interpretationReceivedNeg). \tag{A.36}$$

L'agent dispose d'une base de connaissances constituée de faits, d'interprétations, d'opinions et de souhaits. Cette base de connaissances est représentée en calcul des situations par un fluent *KB*. Le fluent $KB(f, s)$ est vrai si f est considéré par l'agent comme vrai dans la situation s. Par exemple, le fait que l'agent considère dans une situation s que la cuisine est grande est exprimé ainsi :

$$KB(Intepretation(KitchenSurface, large), s). \tag{A.37}$$

Une action *assert* permet d'ajouter un élément à la base de connaissances. Par exemple, l'action $assert(Intepretation(KitchenSurface, large))$ ajoute $Intepretation(KitchenSurface, large)$.

La base de connaissances est mise à jour continuellement au cours de la conversation. Un ensemble de règles permettent de dériver des interprétations à partir des faits et de déterminer l'opinion de l'acheteur vis-à-vis d'un prédicat donné. Ainsi, quand l'agent reçoit un nouveau fait, les interprétations et opinions correspondantes sont également mises à jour. Ces règles sont de la forme $\phi \models fact \rightarrow element$. Par exemple, le lien entre la superficie de la cuisine et son interprétation comme étant grande s'exprime ainsi :

$$\forall x\ 12 \leq x \leq 20 \models Fact(KitchenSurface, x) \rightarrow Intepretation(KitchenSurface, large). \tag{A.38}$$

Ces règles sont utilisées lors de la mise à jour de la base de connaissances. L'axiome de l'état successeur pour *KB* est le suivant :

$$\begin{aligned} KB(f, do(a, s)) \equiv\ & a = assert(f) \\ & \vee\ a = assert(f') \wedge f' \rightarrow f \\ & \vee\ KB(f, s) \wedge \neg invalidates(a, f) \end{aligned} \tag{A.39}$$

$$\begin{aligned} invalidates(a, f) \equiv\ & a = assert(f') \\ & \wedge f = Opinion(p, v) \\ & \wedge \exists v'\ v \neq v' \\ & \wedge (f' = Opinion(p, v') \\ & \quad \vee f' \rightarrow Opinion(p, v')). \end{aligned} \tag{A.40}$$

À l'action *add* s'ajoute trois autres actions. *sensePlayerMessage* récupère un élément de conversation envoyé par le joueur et le place dans le fluent *playerMessage*. Si le joueur a dit plusieurs choses depuis la dernière exécution de *sensePlayerMessage*, c'est le message le plus ancien qui est utilisé (les messages sont placés dans une file). Si le joueur n'a rien dit, le fluent *playerMessage* vaudra la valeur spéciale *nothing*. L'agent peut également envoyer un message au joueur avec l'action *tell* et accuser réception avec l'action *acknowledge*.

L'agent peut se comporter de deux manières différentes : passivement ou activement. Quand il se comporte passivement, il répond aux questions du joueur et prend en compte les informations qui lui sont transmises. L'agent peut également prendre la parole spontanément pour poser des questions spécifiques sur la maison, exprimer son opinion ou donner des informations sur ses souhaits.

```
proc buyer
  (senseUserMessage; if userMessage ≠ nothing
     then reactiveBehavior
     else proactiveBehavior endIf )*
endProc
```
(A.41)

```
proc reactiveBehavior
  πf
  if message = tell(f) then
  evaluate(message)
  else
    message = ask(f)?;answer(f)
  endIf
endProc
```
(A.42)

evaluate est la procédure permettant à l'agent de décider s'il accepte ou non l'information transmise par l'utilisateur. L'agent accepte toujours un fait, mais peut rejeter une interprétation, par exemple si elle n'est pas compatible avec sa propre interprétation des faits. Il en va de même pour une opinion.

```
proc evaluate(message)
  πf;message = tell(f)?;
  if f = Fact(_) then
    assert(f);ackApprove(f)
  else
      ¬incompatible(f)?;assert(f);ackApprove(f)
    |
      ackReject(f)
  endIf
endProc
```
(A.43)

Les procédures *ackApprove*(f) et *ackReject*(f) sont deux procédures simples dont les définitions sont *acknowledge*$(accept)$ et *acknowledge*$(reject)$.

Le fluent *incompatible* est vrai si f est incompatible, c'est à dire que c'est l'agent tire une interprétation ou une opinion différente de celles que le joueur donne à partir des faits qu'il connaît.

$$incompatible(f,s) \equiv f = e(p,v) \wedge \exists f', v'.v' \neq v \wedge KB(f',s) \wedge f' \rightarrow e(p,v') \tag{A.44}$$

La procédure *answer* permet à l'agent de répondre à une demande de souhait ou d'opinion du joueur. Elle est relativement simple : l'agent répond toujours aux questions.

```
proc answer(f)
  πelement, predicate, value
  f = element(predicate) ∧ KB(element(predicate, value)?;
  tell(element(predicate, value))
endProc
```
(A.45)

Quand il se comporte activement, l'agent peut choisir de poser des questions spécifiques sur la maison, ou partager une information.

```
proc proactiveBehavior
  askQuestion | shareInformation
endProc
```
(A.46)

Quand il partage de l'information, il choisit un type d'information, puis une information de ce type qu'il n'a pas encore dévoilé. *shareWish*, *shareOpinion*, *shareFactAboutHouse* et *shareFactAboutItself* sont des procédures cherchant un fait du type en question qui n'a pas encore

été énoncé et le partageant avec le joueur. Leurs implémentations sont laissées de côté, car elles sont longues et n'apportent rien à la présente démonstration.

$$\begin{aligned}&\textbf{proc } \mathit{shareInformation}\\ &\quad \mathit{shareWish} \mid \mathit{shareOpinion} \mid \mathit{shareFactAboutHouse} \mid \mathit{shareFactAboutItself}\\ &\textbf{endProc}\end{aligned} \tag{A.47}$$

Quand l'agent pose une question, il choisit un prédicat pour lequel il n'a pas encore d'information et pose la question au joueur. Le joueur peut répondre par un fait, une interprétation ou une opinion. Dans le cas d'une interprétation ou d'une opinion, l'agent peut la rejeter, comme quand le joueur partage une information directement.

$$\begin{aligned}&\textbf{proc } \mathit{askQuestion}\\ &\quad \pi p.\neg\exists e, v.KB(e(p,v))?;\\ &\quad \mathit{ask}(p); \mathit{senseUserMessage}; \mathit{evaluate}(\mathit{message})\\ &\textbf{endProc}\end{aligned} \tag{A.48}$$

L'agent acheteur peut être personnifié de multiples manières. Nous allons en illustrer deux, et montrer que les deux altérations peuvent être utilisées ensemble sans problèmes.

L'agent peut partager spontanément des informations avec le joueur quand il prend la parole. Le type d'information partagée dans ce cas peut dépendre de la personnalité de l'acheteur. Un personnage extraverti (*extravert*) aura tendance à exprimer des souhaits et des opinions, tandis qu'un personnage introverti aura tendance à exprimer des faits ou poser des questions à propos des faits. Un personnage agréable (*agreeable*) aura tendance à parler de la maison ou de l'environnement, tandis qu'un personnage désagréable aura tendance à parler de lui-même. Dans l'implémentation de [69], cette personnalisation est faite par une énumération des cas possibles (table A.1).

Notre agent choisit quelle information partager dans la procédure *shareInformation* (equation (A.47)). L'agent non personnifié choisit arbitrairement entre *shareWish*, *shareOpinion*, *shareFactAboutHouse* et *shareFactAboutItself*. En exprimant les attributs auxquels sont associées ces actions, il est possible de transformer automatiquement la procédure en fonction d'un profil.

Nous exprimons le trait « introverti » comme la négation de « extraverti » ($neg(\mathit{extravert})$). Ainsi, un agent introverti évitera les actions associées à l'extraversion, et vice versa. Nous faisons de même pour les traits « agréable » et « désagréable ». Compte tenu de la description des traits *agreeable* et *extravert*, la définition de la fonction *attributes* est la suivante :

$$\begin{aligned}\mathit{attributes}(\mathit{shareWish}) &= \{\mathit{agreeable}, \mathit{extravert}\}\\ \mathit{attributes}(\mathit{shareOpinion}) &= \{\mathit{agreeable}, \mathit{extravert}\}\\ \mathit{attributes}(\mathit{shareFactAboutHouse}) &= \{\mathit{agreeable}, neg(\mathit{extravert})\}\\ \mathit{attributes}(\mathit{shareFactAboutItself}) &= \{neg(\mathit{agreeable}), neg(\mathit{extravert})\}.\end{aligned} \tag{A.49}$$

La définition de la fonction *attributes* permet l'utilisation du processus automatique de transformation défini à la section 3.8 pour transformer la procédure *shareInformation* en fonction d'un profil. Les résultats de la transformation [2] pour toutes les valeurs de *agreeable* et *extravert* parmi *pref* −, *neutral* et *pref* + sont donnés en table (A.2).

Face à un choix entre plusieurs actions ou procédures, le processus cherche à les classer de manière à ce que les actions ou procédures favorisées par le profil soient préférées à l'exécution. Dans un premier temps, le processus détermine pour chaque procédure si elle

2. Les noms des actions sont abrégés : le préfixe « share » a été retiré

agreeable	*extravert*	Procédure transformée
pref −	*pref* −	*factAboutItSelf* ⟩ *factAboutHouse* ⟩ (*wish* | *opinion*)
pref −	*neutral*	*factAboutItSelf* ⟩ (*wish* | *opinion* | *factAboutHouse*)
pref −	*pref* +	(*wish* | *opinion* | *factAboutItSelf*) ⟩ *factAboutHouse*
neutral	*pref* −	(*factAboutHouse* | *factAboutItSelf*) ⟩ (*wish* | *opinion*)
neutral	*neutral*	*wish* | *opinion* | *factAboutHouse* | *factAboutItSelf*
neutral	*pref* +	(*wish* | *opinion*) ⟩ (*factAboutHouse* | *factAboutItSelf*)
pref +	*pref* −	*factAboutHouse* ⟩ (*wish* | *opinion* | *factAboutItSelf*)
pref +	*neutral*	(*wish* | *opinion* | *factAboutHouse*) ⟩ *factAboutItSelf*
pref +	*pref* +	(*wish* | *opinion*) ⟩ *factAboutHouse* ⟩ *factAboutItSelf*

Table A.2 – Résultat de la transformation de la procédure *shareInformation* en fonction du profil.

appartient à une classe, en évaluant ses attributs par rapport au profil. Par exemple, pour le profil {*agreeable* : *pref* +, *extravert* : *pref* +}, les procédures *shareWish* et *shareOpinion* ont pour attributs *agreeable* et *extravert* ; elles ont donc pour valeur {*pref* +} et sont de classe C_{pref+}. De la même manière, *shareFactAboutHouse* a pour valeur {*pref* +, *pref* −} et n'a pas de classe, et *shareFactAboutItself* a pour valeur {*pref* −} et est de classe C_{pref-}. Le processus construit alors le programme résultant avec des choix préférentiels entre les différentes classes, des plus favorables aux moins favorables ($C_{req+} > C_{pref+} >$ pas de classe $> C_{pref-} > C_{req-}$). Le programme résultant pour ce profil est donc

(*shareWish* | *shareOpinion*) ⟩ *shareFactAboutHouse* ⟩ *shareFactAboutItself* .

Notre approche permet d'obtenir les variations de programmes, et donc de comportement, pour différents profils en exprimant simplement les attributs des différentes actions du programme. Cela permet de définir des comportements complexes sans avoir à spécifier exhaustivement le comportement pour chaque combinaison de trait.

Quand le joueur donne des informations à l'agent, l'agent peut choisir de les accepter ou de les refuser. Les faits sont toujours acceptés, car l'agent considère que le joueur ne ment jamais. Les opinions et interprétations peuvent par contre être refusées si elles sont en contradiction avec ce que pense l'agent (si l'agent a une interprétation différente des mêmes faits). Ce comportement est régi par la procédure *evaluate* (A.43).

Pour personnifier l'agent, on peut modifier ce comportement. Ainsi, un agent extraverti aura tendance à présenter implicitement son opinion en réponse à une information (« Super ! »), tandis qu'un agent introverti se contentera de confirmer qu'il a compris (« D'accord. »). De plus, un agent désagréable aura tendance à ne même pas fournir de réponse. Enfin, un agent agréable aura plutôt tendance à accepter les opinions des autres qu'un agent désagréable.

Dans la procédure existante, les réponses possibles pour l'agent sont *ackApprove* et *ackReject*, qui communiquent l'acceptation ou le rejet de l'information de manière neutre. Pour permettre à l'agent d'exprimer autre chose, nous introduisons de nouvelles procédures et définissons deux familles d'actions, correspondant aux différentes manières de communiquer l'approbation et la désapprobation.

Tout d'abord, une procédure *ackApproveWithOpinion* qui permet à l'agent de confirmer qu'il accepte ce qui lui a été dit, tout en donnant son opinion sur cette information. Cette procédure n'est disponible que si l'agent a une opinion sur le sujet. Cette procédure est

associée à un comportement extraverti.

$$
\begin{array}{l}
\textbf{proc}\ \mathit{ackApproveWithOpinion}(f) \\
\quad \pi \mathit{predicate}, \mathit{opinion} \\
\quad f = \_(\mathit{predicate}, \_) \wedge \mathit{KB}(\mathit{Opinion}(\mathit{predicate}, \mathit{opinion}))? \\
\quad \textbf{if}\ \mathit{opinion} > 0.5\ \textbf{then} \\
\qquad \mathit{acknowledge}(\mathit{accept\_happy}) \\
\quad \textbf{else} \\
\qquad \mathit{acknowledge}(\mathit{accept\_unhappy}) \\
\quad \textbf{endIf} \\
\textbf{endProc}
\end{array}
\tag{A.50}
$$

$$\mathit{attributes}(\mathit{ackApproveWithOpinion}) = \{\mathit{extravert}\} \tag{A.51}$$

Ensuite, une procédure *ackApproveSilently* qui permet à l'agent de ne pas communiquer son approbation. Concrètement, la procédure ne fait rien. Une autre procédure *ackRejectSilently* permet à l'agent de refuser silencieusement. Ces deux procédures sont distinctes, car elles font partie de familles différentes : l'une peut être utilisée pour l'approbation, l'autre pour la désapprobation. Toutes deux sont associées à un comportement désagréable.

$$\textbf{proc}\ \mathit{ackApproveSilently}(f) \quad \mathit{nil}\ \textbf{endProc} \tag{A.52}$$

$$\textbf{proc}\ \mathit{ackRejectSilently}(f) \quad \mathit{nil}\ \textbf{endProc} \tag{A.53}$$

$$\mathit{attributes}(\mathit{ackApproveSilently}) = \{\mathit{neg}(\mathit{agreeable})\} \tag{A.54}$$

$$\mathit{attributes}(\mathit{ackRejectSilently}) = \{\mathit{neg}(\mathit{agreeable})\} \tag{A.55}$$

Deux familles signalent quelles procédures sont interchangeables avec quelles autres :

$$\mathit{family}(\{\mathit{ackApprove}, \mathit{ackApproveWithOpinion}, \mathit{ackApproveSilently}\}), \tag{A.56}$$

$$\mathit{family}(\{\mathit{ackReject}, \mathit{ackRejectSilently}\}). \tag{A.57}$$

Enfin, l'action d'ajouter un message à la base est liée au trait « agréable ».

$$\mathit{attributs}(\mathit{assert}(f)) = \{\mathit{agreeable}\} \tag{A.58}$$

Le processus de transformation utilise ces familles pour remplacer *ackApprov* et *ackReject* par des alternatives plus appropriées au profil psychologique de l'agent. En outre, le choix non-déterministe entre l'approbation et la désapprobation pourra être remplacé par un choix préférentiel en fonction du profil.

Ainsi, pour un agent extraverti ($\{\mathit{extravert} : \mathit{pref}+, \mathit{agreeable} : \mathit{neutral}\}$), le résultat de la transformation est le suivant. Les différences sont soulignées. Pour chaque procédure faisant partie d'une famille de procédures (en l'occurrence, *ackApprove* et *ackReject*), chaque membre de la famille est évalué et classé à l'aide des règles définies à la section 3.7. Ainsi, *ackApproveWithOpinion* est de classe $\mathit{pref}+$, puisque son unique attribut est *extravert*, donc la valeur dans le profil est $\mathit{pref}+$. De même, *ackRejectSilently* et *ackApproveSilently* sont de classe $\mathit{pref}-$ et *ackApprove* et *ackReject* n'ont pas de classe. Les procédures de classes négatives [3] sont écartées, et les procédures du programme sont remplacées par un choix entre les procédures restantes dans une famille, avec une préférence pour les procédures de plus grande

classe. Ainsi, la procédure $(ackApproveWithOpinion(f) \rangle ackApprove(f))$ remplace la procédure $ackApprove(f)$ et c'est la seule transformation qui a un effet dans cet exemple.

```
proc evaluate(message)
  πf; message = tell(f)?;
  if f = Fact(_) then
    assert(f); (ackApproveWithOpinion(f) ⟩ ackApprove(f))
  else
      ¬incompatible(f)?; assert(f); (ackApproveWithOpinion(f) ⟩ ackApprove(f))
    |
      ackReject(f)
  endIf
endProc
```
(A.59)

Pour un agent désagréable $(\{extravert : neutral, agreeable : pref-\})$, le résultat est le suivant. En plus du remplacement des procédures par un choix entre les procédures de la famille, la transformation des choix non-déterministes a également un effet. Pour chaque choix non-déterministe « | », on compare les parties gauches et droites : si on peut prouver que l'une est plus grande que l'autre par les règles définies à la section 3.8, le choix est remplacé par un choix préférentiel « ⟩ » avec une préférence pour le plus grand des deux. Dans ce cas, on prouve que

$$ackRejectSilently(f) \rangle ackReject(f)$$

est préférable à

$$\neg incompatible(f)?; assert(f); (ackApproveSilently(f) \rangle ackApprove(f)).$$

```
proc evaluate(message)
  πf; message = tell(f)?;
  if f = Fact(_) then
    assert(f); (ackApproveSilently(f) ⟩ ackApprove(f))
  else
      ackRejectSilently(f) ⟩ ackReject(f)
    ⟩
      ¬incompatible(f)?; assert(f); (ackApproveSilently(f) ⟩ ackApprove(f))
  endIf
endProc
```
(A.60)

Enfin, pour un agent agréable et introverti $(\{extravert : pref-, agreeable : pref+\})$, aucune action des familles d'actions n'a de classe positive, donc seule la transformation du choix à un effet. Ici, on peut prouver que $\neg incompatible(f)?; assert(f); ackApprove(f)$ est supérieur à

3. Une procédure de classe négative $(C\_)$ est une procédure dont l'affinité est inférieure à $\{pref-\}$.

ackReject(*f*).

```
proc evaluate(message)
  πf; message = tell(f)?;
  if f = Fact(_) then
    assert(f); ackApprove(f)
  else
      ¬incompatible(f)?; assert(f); ackApprove(f)
    ⟩
      ackReject(f)
  endIf
endProc
```
(A.61)

Ainsi, notre approche permet de définir des altérations complexes du comportement en définissant simplement les nouveaux comportements à incorporer et les attributs liés à ces nouveaux comportements et aux anciens. La procédure de transformation automatique utilise les attributs pour déterminer quand incorporer ces nouveaux comportements dans le programme, ce qui permet une altération complexe sans nécessiter une expression complexe.

Ces deux transformations touchant des points différents du programme, il est tout à fait possible de les appliquer simultanément pour obtenir un agent présentant les deux altérations de comportement. Avec le programme agent résultant, un agent extraverti va à la fois partager ses souhaits et opinions spontanément, et répondre aux informations qui lui sont données en donnant son avis dessus.

Des problèmes ne se posent que dans les cas où les altérations touchent des parties similaires du programme, auquel cas le résultat dépend du processus de transformation lui-même. Pour cette raison, nous considérons que le processus est semi-automatique : le concepteur de l'application peut vouloir valider le résultat de la transformation.

B - Articles choisis

Une sélection d'articles publiés suite aux travaux présentés est incluse ci-dessous. Ces articles sont référencés en fin des chapitres afférents.

GEAHS: A Generic Educational Adaptive Hypermedia System Based on Situation Calculus

Cédric Jacquiot[1], Yolaine Bourda[1], and Fabrice Popineau[2]

[1] Supélec, Plateau de Moulon, 3 rue Joliot-Curie, 91192 Gif/Yvette CEDEX, France
{cedric.jacquiot,yolaine.bourda}@supelec.fr
[2] Supélec, 2 r Edouard Belin, 57070 Metz, France
fabrice.popineau@supelec.fr
http://www.supelec.fr

Abstract. GEAHS is a platform designed to ease the development of Adaptive Educational Hypermedia, using standard formalisms. In this document, we explain the underlying principles of this platform. Genericity is achieved thanks to an adaptation engine based on situation calculus and RDF. This paper describes the main aspects of our system, as well as the use we make of situation calculus to create a simpler, more reusable adaptive hypermedia system.

1 Introduction

Many Adaptive Hypermedia Systems (AHSs) have been developed for educational purpose in the past few years. Most of them have been created for a precise purpose, and are not reusable. In other words, these systems have to be build from scratch any time one wants to develop a new teaching platform. P. Brusilowsky [1] described a methodology for creating an adaptive hypermedia. P. de Bra [2] studied the theoretic and generic ground that should be used for the creation of Adaptive Hypermedia Systems. Our goal has been threefold. First, we wanted to create an engine, based on principles close to those of AHAM [2], using standard or recommended formalisms, in order to make it as reusable as possible. Second, we wanted to provide a model both simple and powerful. We wish that as many people as possible are able to reuse our system. In order to achieve this, we also provide an extensible set of reusable adaptation rules. This way, it is possible to avoid the creation from scratch of new adaptation rules. The adaptation is provided with generic built-in rules and metadata that can be reused (and modified if necessary) by the AHS creators. Third, we wished to prove that situation calculus, introduced in [3] for adapting the semantic web, can be applied to the problem of generic adaptive hypermedia. We also wish to show that it is a satisfactory solution for the second goal we want to achieve.

Our architecture is composed of three interacting components : the learner model, designed to represent metadata about the learner (in RDF); the domain model, designed to represent metadata about the learning resources (in RDF); and the adaptation engine, composed of a set of rules as well as a situation calculus-based infering engine, which provides the adaptation (currently, we only provide link adaptation).

In Sect. 2, we will show how we use situation calculus for creating an adaptive hypermedia. In Sect. 3, we will see how we use logic to create our architecture. In Sect. 4, we will compare our situation calculus approach to the condition/action approach developed in [4].

W. Nejdl and P. De Bra (Eds.): AH 2004, LNCS 3137, pp. 413–416, 2004.

2 Using Situation Calculus

Situation Calculus was first introduced in order to manage robotics problems. Nevertheless, the idea has emerged in [3] that Situation Calculus could be used for driving web applications. This way, situation calculus could help program agents that are often very complex.

Situation calculus [5] is based exclusively on FOL, i. e. deductions can be done using nothing but FOL. It is a subset of FOL to which has been added the notions of situations and actions. A situation is a group of static facts that can be evaluated at a given instant. Actions are more or less complex procedures that apply to a situation. An action can be possible or not. A set of primitive action can be given, and complex actions are built from the primitive actions (sequence, test, nondeterministic choice ...).

An action a is made on a situation s leading to a situation s' if a is possible in situation s and if s' is the consequence of action a in situation s. Accomplishing a complex action results in accomplishing choices and sequences of primitive actions.

The predicates $poss$, do and $primitive\_action$ are predefined in situation calculus. $poss(action, situation)$ is true iff $action$ is possible in $situation$. $Primitive\_action(action)$ is true iff $action$ is a primitive action. $Do(action, situation1, situation2)$ is true iff $poss(action, situation1)$ and $action$ applied to $situation1$ leads to $situation2$.

In order to calculate the situation changes, the primitive action must be calculable, i. e. primitive actions must be defined. The fluents - the data describing a situation - must also be described. It is even the most important part, since calculating the next situation is achieved by calculating the next value of the fluents. The fluents' description gives us their new value from the action and previous situation. The way to calculate possibilities has to be given, since an action cannot be done if it is not possible.

On top of the basic situation calculus, we have added the notion of desirability. This notion is related to what a learner really needs to know to reach his objective. For example, if several documents have the current document as a common pre-requisite, they can all be possible documents to read next. But only those who are pre-requisites of the objective are really desirable.

We have worked on adapting situation calculus to our specific problem, and we found the following solution. First, we have split the possible actions into two kinds. The first kind is dedicated to the user's actions. The second kind is dedicated to the engine's actions. In a given situation, the system knows the up-to-date learner profile, the domain, and the position of the learner in the document space. This is considered to be the current situation. The system uses one type of action: displaying the links. This action does not trigger any rule. Once the possible links are displayed, the user can do actions. He/she can read the document, take a test, click on a link and so on.

3 Logic in GEAHS

Rules in GEAHS are described in FOL. First, FOL has a defined semantics. Then, in some cases, translating rules from natural language to FOL is not a difficult task. Our purpose has been to allow as many people as possible to create their own rules if they are not satisfied with the set of pre-defined rules we provide.

Rules can be written with classic FOL operators, even though it is not naturally provided in Prolog, the language we used to implement our system. In a future developpement, we intend to provide a simple interface to enter and modify the rules.

For an AHS creator, using such FOL rules can be simpler than using proprietary rules. First, if she happens to know FOL, her learning of our AH creation system will be much shorter. Then, if she already has rules written in FOL (which seems to be a natural way to express "logic" rules), she will be able to reuse them easily. Comparing these rules to other kinds of rules found in many adaptive systems, it does not seem more difficult to write FOL rules than to write rules in other formalisms. And at least these rules have a commonly-understood meaning.

We shall soon work on the problems of consistency and completion (including their tractability) of the rule sets.

As we implement the whole system in Prolog, we need all forms to be boiled down to Horn clauses in the end. Situation calculus was already implemented. We implemented the notion of desirability in the Golog program. We also created a module to allow people to write logic in its standard form. This module makes standard logic understandable by Prolog. The clauses are not transcribed into Horn clauses, but dynamically analysed. The analyse is mostly instantaneous, as the main logic operators are already parts of Prolog possibilities.

The main program is in charge of reading, writing and displaying data. It launches the Golog mechanism. This mechanism uses rules written in standard logic, which are analysed through our logic module.

In the end, all forms of logic are easily interfaced, working together in a simple way. We do have a homogeneous system, and this system's semantics are clearly defined.

4 Situation Calculus and Condition/Action

In Wu's condition/action system, the AHS creator defines a set of pairs of conditions and actions. Each time the user makes a physical action, or each time the user representation evolves, the rules' conditions are checked. If the condition is true, the corresponding action is triggered. As long as rules are triggered (possibly by other rules' consequences), their actions are applied. Rule triggering can loop. In order to be sure to achieve termination in a condition/action model, Wu and De Bra introduced restrictions to the rules that can be described using this formalism. In the end, the model they use is a transformed version of condition/action, and thus, non-standard.

In GEAHS, each time the user makes an action, the finite set of (ordered) rules is triggered. When a rule is applied, it can have consequences on other rules that come after it in the defined order, but it cannot trigger other rules. Thus, the risk for an AH creator to make up rules with no termination is reduced by the intrinsic functioning of our system. On the other hand, we did not prove yet that situation calculus is as expressive or efficient as condition/action.

5 Conclusion

GEAHS is a powerful system which allows all kinds of people - especially those who are not expert in computer science or software engineering - to create an AHS for ed-

ucation. All formalisms and techniques used for GEAHS are either standards/recommendations, or well-defined calculus based on FOL. This allows our system to import, export and reuse data from or to other systems and formalisms. This is a fundamental aspect in today's semantic web, if we want to be able to have fully distributed documents (coming from different sources).

Our future work will consist in several points. We wish to provide simple techniques for content and style adaptation. We also want to study the use of OWL, the Web Ontology Language for representing our data and metadata, since it provides pre-defined and useful relations. We also wish GEAHS to be able to determinate the consistency and completion of the set of rules on may create.

References

1. Brusilowsky, P.: Methods and Techniques of Adaptive Hypermedia. Adaptive Hypertext and Hypermedia. ED. Kluwer Publishing (1995) 1–43
2. De Bra, P., Houben, G-J, Wu, H.: AHAM: A Dexter-Based Reference Model for Adaptive Hypermedia. Conference on Hypertext (1999) 147–156
3. McIlraith, S.: Adapting Golog for Programming the Semantic Web. Conference on Knowledge Representation and Reasoning (2002)
4. Wu, H.: A Reference Architecture for Adaptive Hypermedia Applications. PhD Thesis, Eindhoven University of Technology, The Netherlands
5. Levesque, H., Reiter, R., Lesperance, Y., Lin F., Scherl, R.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31** (1997) 59–84

GLAM: A Generic Layered Adaptation Model for Adaptive Hypermedia Systems

Cédric Jacquiot[1], Yolaine Bourda[1], Fabrice Popineau[1], Alexandre Delteil[2], and Chantal Reynaud[3]

[1] Department of Computer Science, Supélec, Plateau de Moulon, 3 rue Joliot-Curie 91192 Gif-sur-Yvette CEDEX, France
{Cedric.Jacquiot, Yolaine.Bourda, Fabrice.Popineau}@supelec.fr
http://www.supelec.fr
[2] France Telecom R&D, 38 rue du général Leclerc, 92130 Issy-les-Moulineaux, France
Alexandre.Delteil@francetelecom.com
[3] Université Paris-Sud XI, CNRS (LRI) & INRIA(Futurs), 91405 Orsay, France
Chantal.Reynaud@lri.fr

Abstract. This paper introduces GLAM, a system based on situation calculus and meta-rules, which is able to provide adaptation by means of selection of actions. It is primarily designed to provide adaptive navigation. The different levels of conception, related to different aspects of the available metadata, are split in different layers in GLAM, in order to ease the conception of the adaptation system as well as to increase the potential use of complex adaptation mechanisms. GLAM uses meta-rules to handle these layers.

1 Introduction

With the development of networking technologies, and more specifically of the internet, the number of documents available both inside and outside organizations, such as companies or universities for example, has been increasing dramatically. It is commonly agreed upon that most companies' intranets are not very efficient at providing pertinent documents to the employees, let alone to order the documents or adapt them to the user's status in the company. Many Adaptive Hypermedia Systems (AHSs) have been developed in the past few years, and can provide adaptation within an annotated corpus of documents.

The problem of adaptation in AHSs has been addressed in several ways. Generally, an adaptation engine is made of adaptation data - most often a set of adaptation rules -, which can potentially be redefined by the AHS creator in order to provide different forms of adaptation, and of an inference engine, which applies the rules in order to select documents' contents and links, and in order to update the user model, i.e. mostly his knowledge, as in [1]. Some systems like [2] provide mechanisms to check if the adaptation rules are coherent and if the use of the set of rules won't result in a bad situation - like no adaptation at all for some users, or infinite loops of rules triggering each other.

V. Wade, H. Ashman, and B. Smyth (Eds.): AH 2006, LNCS 4018, pp. 131–140, 2006.

Most of these systems (cf. [2], [3]) provide adaptive navigation support and adaptive presentation, as defined in [4]. They consider fragments, atomic resources, to build up pages to provide to the user. They also modify links' visibility according to the user's model and goal.

In this paper, we focus our attention on adaptive navigation. We wish to be able to provide direct guidance, adaptive sorting and adaptive hiding. Our purpose was to create a purely declarative model, and to see how far situation calculus [5], which offers such a declarative approach, can be used as a model for adaptation in a closed-domain context [6]. Situation calculus offers a way to describe user/system interactions in an AHS very simply. Situation calculus allows us to define a model taking actions into account. An action can be "reading a document", "taking a test", "making exercises" etc. Rules classify actions according to the user's knowledge, preferences and position in the domain.

GLAM is a model for adaptation. Thus, it can be used to create different applications, in different domains. It defines a natural way to describe adaptation, thanks to the declarative aspect of situation calculus. This benefits to an AHS creator, since he has less efforts to produce in order to translate his needs into our formalism. We also provide a generic inference engine, which can take any GLAM based adaptation data into account to provide adaptation.

We also introduce a layered rule model. Usually, rules for adaptation can take into account any data relative to the user's preferences, on the one hand, and to his knowledge of the domain, on the other hand. Writing such rules can be quite complex: their premises can be numerous and of different natures. Such complex rules are hard to debug as well as to maintain, and simpler rules only offer limited adaptation. Moreover, user's preferences and knowledge are very different kinds of data about the user, thus they influence adaptation in different ways. Preferences are domain-independant. They help to choose between different kinds of documents to offer: examples, illustrations, summaries, or how fast the user can learn. On the other hand knowledge, which is domain-dependant, tells what document can or cannot be read by the user, what concept can or cannot be treated. Thus, GLAM includes a rule system layered according to the nature of the user metadata being used. The layering is achieved using meta-rules.

In section 2, we present the global architecture of GLAM In section 3, we present situation calculus and its role in our context. In section 4 we describe our adaptation model, and we detail its layered rule system. Finally we compare our approach to other well-known approaches and conclude with some perspectives about making a full generic AHS using GLAM for adaptation.

2 Global Architecture

As in most AHSs, we consider a user model, a domain model and an adaptation model. In this paper, we won't discuss the details of the user or domain models, but we focus on the adaptation model. However, we have designed GLAM in such a way that it can handle various kinds of metadata about documents and users - including relations.

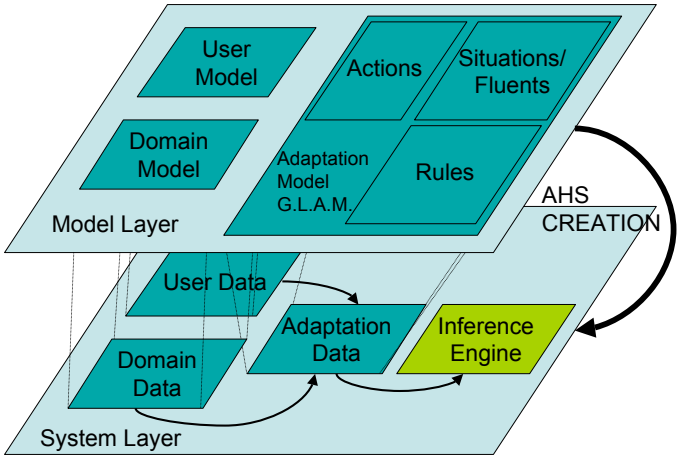


Fig. 1. GLAM's architecture

In this section, we introduce the global architecture of GLAM The way our system works is presented in figure 1. We present our architecture according to two abstraction layers. The model layer, which provides models for implementing the data into the AHS, and the system layer, which provides the data itself and the inference engine. The data is passed to the inference engine, which is able to compute the next possible step(s). As in many systems, the inference engine is reusable, since it can work with different adaptation data.

Our adaptation model is based on Sheila McIlraith's [7] adaptation of situation calculus presented in [7], and on results specific to planning techniques [8, 9]. Situation calculus provides a simple-to-understand, well-founded and expressive-enough basis for an adaptation model. Situation calculus is made of actions, situations and precondition rules (cf. section 3).

As detailed in section 4, we have enriched situation calculus in two ways. First, we have added notions related to preconditions in situation calculus, and redefined notions found in [7]. Secondly, and most importantly, we have defined a layered rule model, which replaces and enriches situation calculus's simple rule system. Criterions relative to the user are taken into account at different levels of the rule model, in order to ease the development of a system implementing potentially complex adaptation strategies.

Our inference engine uses data, described using our own version of situation calculus, to provide adaptation : it is able to tell what actions are best suited for the user's next step, according to his profile and position in the domain.

3 Situation Calculus

3.1 Why Using Situation Calculus?

Situation calculus [5] is a logic model, based on first order logic, that allows to describe observable situations modified by actions and to reason about these

different items. Thus, our reasons for using situation calculus to provide adaptation are twofold. First, situation calculus allows to closely model reality : at any given time, situation calculus represents a situation as the result of a sequence of actions applied to an initial situation. Then, in any given situation, it is possible to do some actions among all existing actions. The fact that an action can or cannot be accomplished in a given situation is provided by rules, taking into account the current situation and the potential actions. Once an action is accomplished, the situation is modified according to this action.

In an AHS, a situation is given by the position of the user in the domain - which is the result of the various actions the user has achieved and of his initial knowledge and position - and by his preferences. An action in a given situation is to access a document that the user's current knowledge allows him to read. Once the user accomplishes an action, the situation is modified, i.e. his profile is updated. Thus, situation calculus is natively close from the reality of a user using an AHS and can easily be used to provide an adaptation model and the corresponding inference engine in an AHS.

3.2 Notions of Situation Calculus

Situation calculus is a formalism made of three distincts elements :

- Actions: they are the basis of situation calculus. They modify the situation when they are achieved. There are two kinds of actions : primitive actions and composite actions a.k.a procedures. In GLAM, we only focused on primitive actions, since the user takes them one at a time. One needs to declare every primitive action he intends to use using the *primitive_action* predicate. For example:
  $primitive\_action(read(Document))$.
  $primitive\_action(read\_about\_concept(Concept, Select\_Example)$.
  This last example shows that, if we add a document composition engine to our system layer, it is possible to achieve content adaptation on top of adaptive navigation. However, we did not address this possibility (yet).
- Fluents: they allow to observe the current situation. They are predicates defined for the initial situation, as well as for the situation resulting of an action "done" in the previous situation. The predicate *holds(Observation, Situation)* allows to describe if an *Observation* is true in a given *Situation*. The operator *do* transforms .For example:
  $holds(read(Document), initial\_situation) \leftarrow false$
  $holds(read(Document), do(read(Document), Previous\_Situation))$.
  $holds(read(Document), do(read(Other\_Doc), Previous\_Situation)) \leftarrow$
  $\quad holds(read(Document), Previous\_Situation)$.
  means that no document has been read in the initial situation. A document is read if the last action that was "done" consisted in reading it or if it was already read in the previous situation.
- Preconditions: the *poss* operator is used to define rules that state if an action is possible or not in a given situation.

More details about situation calculus can be found in [5] and [10]. Sheila McIlraith [7] added a finer level of precondition by defining the notion of desirability. It is then possible to accomplish an action if the resources are available, and desirable to do it if the user's knowledge is sufficient. Josefina Sierra-Santibañez [9], who studied situation calculus as a way to achieve heuristic planning, refines this model by introducing a notion of order among the actions.

4 Modifying the Situation Calculus Preconditions to Match AHSs Needs

In GLAM, we decided to reuse the notions of primitive actions and fluents without redefining them. As one can define the actions and fluents he wants in a declarative manner, there appeared to be no need to redefine those notions in our context. However, preconditions are treated in a very simple manner: if a precondition rule is true, the corresponding action is possible, otherwise it is impossible. Moreover, they are described by a rule system using horn clauses, which is not especially made for AHSs (cf. 4.2). Thus we decided to redefine the notion of precondition to make it fit better in AHSs.

4.1 Redefining the Notion of Precondition

In order to redefine preconditions in situation calculus, we took ideas from [7] and [9]. We decided to use the notions of desirability and preference. These notions allow to create a double classification among actions. First, some actions are desirable, some are possible and some are bad. Secondly, within any of the previous categories, we can establish an order among actions, potentially allowing us to guide the user in a step-by-step manner. However, since we work in a closed-domain context, we did not use the same definitions as those proposed in [7]: the notion of document availability is not very useful in this context. Thus, we decided to redefine these notions so that they provide indications about the action's level of interest within the adaptation model.The notions of possibility, desirability and order are redefined as follows:

- It is possible (predicate `poss`) to accomplish an action if the user is able to understand the document that will be presented to him as a result of this action.
- It is desirable (predicate `good`) to accomplish an action if the user is able to understand the document that will be presented to him as a result of this action, and if this document leads to his goal(s).
- It is better (predicate `better`) to accomplish an action than another if it is more adapted to the learner's preferences than the other.
- It is bad to accomplish an action if it is neither desirable, nor possible. Thus, there is no predicate associated to bad actions, avoiding potential conflicts among rules.

4.2 The Layered Rule System

Layering the Adaptation Rules. AHSs can usually adapt their behavior to different aspects of the user : preferences, knowledge, goal [3, 11]. Rules in AHSs are often a mix of several different aspects [3]. In our version of situation calculus, one can describe rules for selecting good actions, possible actions, and actions better than other actions. These rules are triggered after each action is achieved, in order to compute the next possible one(s). They are independent. They cannot trigger each other. Even though they are triggered in a given order, it has no influence on the result, which is a partially ordered set of actions. Preconditions only help computing the category - bad, possible, desirable, better than - to which an action belongs. However, as in many systems, if one wishes to take several aspects of the user into account in a given rule, he has to add premises to the rule. This can lead to quite complex rules, where it becomes difficult to distinguish the different aspects of the user taken into account for adaptation. Maintaining such a rule system can be tedious.

In order to address this problem, our idea was to "adapt the adaptation" according to the user's characteristics. To do so, we define two levels for rules. First, our base rules only use data about the domain and the position of the user in this domain, i.e. domain-related knowledge. They describe different adaptation strategies, potentially incompatibles. Then, the inner characteristics of the user, like learning preferences, learning speed for example, are taken into account at the meta-level. These characteristics are domain independant, i.e. they can be reused in several domains. A set of meta-rules uses the user's characteristics to select the adaptation base rules which will be used to provide adaptation for this user. For example, some base rules can describe a linear strategy, some can add the necessity for the user to practice exercises, some rules can also indicate that the fastest way to the goal is the best way. Rules recommending exercises and rules that help to achieve the goal as fast as possible will be mutually exclusive. Thus, the meta-rules will help select the correct rules for the current user according to his profile.

This approach offers several advantages. First, the relations between the user and the domain on one side, and the inner characteristics of the user on the other side, which are often separated in the user models, are no longer mixed in the rules. They are separated at different levels of the adaptation model. The base rules provide a mean for action selection in a given position in the domain, whereas the meta-rules select the best set of adaptation rules for the current user. Maintaining the system becomes simpler since base rules and meta-rules are smaller, and thus easier to understand for the AHS creator, than classic base rules. Finally, this way of selecting rules is much finer than a classic stereotyped approach [3], since it is possible to mix up numerous characteristics about the user, and every different set of characteristics can provide a potentially different set of adaptation rules to use in the inference engine.

A Meta-rule Model. Our meta-rule model is an adaptation of a model created by Jagadish [12]. This model was originally designed for automatic database

updates. It offers possibilities to help a system creator: it uses efficient algorithms to control the properties of the meta-rules.

It offers a formal approach for selecting rules according to their premises. In [12], a rule is said to be fireable if, and only if, all its premises are true. Meta-rules allow to select a subset of the set of all fireable rules, called the execution set. This set is generated using four kinds of meta-rules. Each kind of meta-rule is a binary relation between rules. The first kind of meta-rule is called requirement meta-rule, and allows to express that a rule requires another rule to be selected for execution, in order to be in the execution set. The second kind is called exclusion, and allows to describe what pairs of rules cannot be in the execution set together. The third one is called preference, and allows to describe which rule to prefer over which other rule when the two rules are exclusive from one another. Finally, the order meta-rules provide a total or partial order among the rules in the execution set.

The system also provides axioms that explain how to deduce new meta-rules from a set of given meta-rules, to detect determinism and order issues, and to select the execution rules. Thus, they can help the system creator by detecting flaws in his set of rules. For example, if a rule is exclusive from itself, the system can inform the creator that it is useless. The system can also detect cases where two rules are exclusive from one another, and yet there is no way to prefer one rule. This is called the determinism problem, and can be checked in polynomial time. If a total order among selected rules is required, the fact that the order meta-rules provide a total order for all possible fireable set can also be computed in polynomial time. All those verifications only depend of the meta-rules. They do not depend on the base rules.

Our AHS-oriented meta-rule System. We modified this system by changing the notion of fireability, which badly fit with situation calculus, and by incorporating account AHS specific notions, related to the user's capacity.

For AHSs, we define our rule system as a tuple $< V, F, M, \delta >$ where :

- V is a set of base adaptation rules, which take the form of horn clauses whose results are degrees of desirability for actions. These rules take into account the current situation and a potential action, and determine if the action is desirable (or possible, or better than another one). For example:
  $good(read(Document), Situation, User) \leftarrow$
  $not(read(User, Document)), good\_for\_partial\_goal(Document),$
  $current\_document(CDocument),$
  $prerequisite(CDocument, Document).$
  means that it is desirable to read a document if it has not already been read, if it leads to the goal and is one of the documents directly linked to the current document by a prerequisite relation. The only premises allowed for base rules are domain-related relations.
- F is a set of firing criterions. These criterions are related to the user's inner characteristics. For example : $fast\_learner = true$.
- M is a set of meta-rules. We reused the four kinds of meta-rules introduced in [12], but our meta-rules are between sets of rules instead of single rules,

allowing us to give information about all rules related to a specific user's characteristic at the same time. For example:
$\{rule1, rule2\} \supset \{rule4, rule5\}$
means that rules 1 and 2 require rules 4 and 5 to be selected for execution in order to be in this execution set.

- δ is a function that associates base rules with firing criterions. Unlike in [12], we cannot use the premises of a rule in a given situation as a firing criterion, since a rule is used several times in a give situation, according to the action that is being tested. δ allows to compute the γ function, which associates every criterion with the set of rules fireable according to this criterion. Thus, it is possible to write meta-rules about sets of rules related to a specific criterion, i.e. to write meta-rules about requirement, exclusion, preference and order between user's criterions. For example:
$\gamma(fast\_learner = false) \supset \gamma(need\_exercise = true)$
means that rules for slow users will include rules providing exercises.

Once those four elements are described, the part of the inference engine dedicated to the rule processing uses the axioms in [12] and formulas derived from these axioms, which make deductions about our meta-rules, to generate all deductible meta-rules. It checks if some rules will never be selected, if the system will always select the same set of base rules for a given set of criterions, if all rules that can be selected simultaneously are totally ordered, and if some set of criterions can lead to an empty set of adaptation rules. These verifications are intended to help the AHS creator who creates a GLAM-based AHS. They all are in polynomial time depending on the number of meta-rules. Providing meta-rules for sets of rules instead of single rules eases the creation of the system as well as it helps debugging it, since the meta-rules for set of rules are less numerous than meta-rules for single rules.

After checking all those properties, the system is ready to use. For each user, it evaluates the different firing criterions. It pre-selects the "fireable" rules related to this criterions. Then, it looks at the meta-rules to see if all rules can be put in the execution set or if some must be withdrawn, e.g. if all their prerequisites are not in the fireable set, or if they are exclusive of other preferred rules. Once the execution set is computed, it is ordered.

The decomposition of our rule model in four different parts makes it clearer to distinguish the different categories of elements to take into account. It limits the potential mix up of premises in the base rules as well as their number. Finally, it allows to add new base rules without modifying the meta-rules, thanks to the associations between rules and criterions.

5 Related Work

In this section, we discuss different approaches and how our system differs.

In AHAM [11], like in many other systems, the adaptation model relies on condition-action rules. Theses rules provide adaptation as well as user model update. Condition-action rules can trigger each other, and thus often need to

be restricted to ensure termination and confluence. On the contrary, in our rule system, the nature of rules prevents us from termination problems, and we implemented a verification algorithm which can check if the system is deterministic in all cases, i.e. confluent.

The adaptation system in [13] uses TRIPLE rules for directly interrogating the user and domain data. These rules can recommend documents, which is close from action selection. Moreover, the recommendations can be more or less strong, which is close to our notion of possibility/desirability/preference. However, the layering in our system allows to separate different semantic levels in our data: preferences and knowledge are dealt with separately in GLAM, in order to simplify the conception of the system. By using distinct rule levels, we prevent an AHS creator from writing very long rules, with many premises. This layering is not arbitrary: it lies upon an intrinsic semantic separation of the nature of data manipulated to provide adaptation. Moreover, TRIPLE rules can only manipulate "subject+predicate+object" statements, whereas G.LA.M. rules can manipulate other data representation formalisms.

SCARCE [3] uses stereotypes to select the kind of adaptation to provide. In LAG and LAG-XLS [14], user-related preferences allow to select an adaptation strategy, for example, to adjust presentation of selected concept to present to the user. Our rule system goes beyond stereotypes or selection of strategy. In GLAM, one describes relations between groups of rules associated to one or more user criterion(s). These relations allow to describe the necessary rules for a strategy without having to select them manually. Moreover, a strategy can be based on several user criterions. If one takes 5 binary criterions into account, the number of potential strategies is $2^5 = 32$, which means that without meta-rules, one would have to describe 32 coherent groups of rules manually!

6 Conclusions and Future Work

In this paper, we introduced GLAM, an adaptation model for closed-content domain AHSs. We have already implemented a prototype system using this model, which is able to provide adapted navigation. We implemented the verifications for the rule system. They allowed us to check if our examples were correct and to ease the debugging process.

Using situation calculus as the core for adaptation allowed us to have a declarative representation of the system as close as possible to reality, hopefully easing the reusability of GLAM in many potential contexts. Our layered rule system introduces a way to describe adaptation by selecting adaptation strategies.

Thanks to the form of its rules, GLAM can take into account any kind of relations in the user and domain models. These models can be very different, since the different properties and relations can be different according to the domain. We now intend to provide meta-models for the user model as well as for the domain model. These meta-models are meant to help generating models fully compliant with GLAM They will also offer reusable relations and metadata, in order to ease the creation of AHSs.

References

1. DeBra, P., Houben, G.J., Wu, H.: AHAM: A dexter-based reference model for adaptive hypermedia. In: UK Conference on Hypertext. (1999) 147–156
2. DeBra, P., Aerts, A., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: Aha! the adaptive hypermedia architecture. In: UK Conference on Hypertext. (2003) 81–84
3. Garlatti, S., Iksal, S., Tanguy, P.: Scarce: An adaptive hypermedia environment based on virtual documents and semantic web. In: Adaptable and Adaptive Hypermedia System. (2004) 206–224
4. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction **6**(2-3) (1996) 87–129
5. Reiter, R.: Proving properties of state in the situation calculus. Artificial Intelligence, 64(2):337-351 (1993)
6. Jacquiot, C., Bourda, Y., Popineau, F.: Reusability in geahs. In: Proceedings of Workshops in Connection with the 4th International Conference on Web Engineering, Rinton Press (2004) 199–209
7. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: roceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02). (2002)
8. Fikes, R., Nilsson, N.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **1** (1971) 27–120
9. Sierra-Santibañez, J.: Heuristic planning: A declarative approach based on strategies for action selection. Artificial Intelligence **153** (2004) 307–337
10. Levesque, H.J., et al.: Golog: A logic programming language for dynamic domains. Logic-based artificial intelligence pp 257 - 279 (2000)
11. Wu, H., de Kort, E., DeBra, P.: Design issues for general-purpose adaptive hypermedia systems. In: HYPERTEXT '01: Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, New York, NY, USA, ACM Press (2001) 141–150
12. Jagadish, H.V., Mendelzon, A.O., Mumick, I.S.: Managing conflicts between rules. J. Comput. Syst. Sci. **58**(1) (1999) 13–28
13. Dolog, P., Henze, N., Nejdl, W., Sintek, M.: The personal reader: Personalizing and enriching learning resources using semantic web technologies. In: Proceedings of the third International Conference on Adaptive Hypermedia and Adaptive Web-Based Sytems, Springer (2004) 85–94
14. Stash, N., Cristea, A., DeBra, P.: Explicit intelligence in adaptive hypermedia: Generic adaptation languages for learning preferences and styles. In: Proceedings of the HT 2005 CIAH Workshop. (2005)

2011 23rd IEEE International Conference on Tools with Artificial Intelligence

A formal approach to personalization

Georges Dubus, Fabrice Popineau, Yolaine Bourda
SUPELEC Systems Sciences (E3S) - Computer Science Department
Gif sur Yvette, France
Email: georges.dubus@supelec.fr, fabrice.popineau@supelec.fr, yolaine.bourda@supelec.fr

***Abstract*—Personalized systems are a response to the increasing number of resources on the Internet. In order to facilitate their design and creation, we aim at formalizing them. In this paper, we consider the relationship between a personalized application and its non-personalized counterpart. We argue that a personalized application is a formal extension of a non-personalized one. We aim at characterizing the syntactic differences between the expression of the personalized and non-personalized versions of the application. Situation calculus is our framework to formalize applications. We introduce two scenarios of non-personalized application that we personalize to illustrate our approach.**

***Keywords*-situation calculus; personalization; web application; adaptive systems; weaving;**

I. Introduction

The Internet is quickly becoming the main way of communicating and sharing information. The amount of content available online has increased exponentially. A single website often contains so much information that a user can't grasp it all, and may even be unable to navigate to find the information he wants. Personalized systems – systems that behave differently depending on the user – emerged to provide guidance to users in such cases.

Different types of personalized systems exist to answer different problems. Adaptive hypermedia guide the navigation in a set of resources, recommender systems offer items to the user that are likely to interest him, and personalized information retrieval systems provide the user with answers to his queries that are tailored to his needs. These different systems have distinct inner workings and distinct logical models, if any. Some systems cross the borders between those types. For example, an e-learning web application may provide an adaptive hypermedia with learning resources, and also recommendations of external resources, such as a book to read to go further in a subject.

Personalized systems are not as widely used as they should be because their creation is a difficult process [1]. There are tools to help with the creation of some of the systems (adaptive hypermedia, recommender systems), but they are dedicated to one single system, and there are no general tools covering all the personalized systems. We aim at defining a general formalization of personalized systems. This will enable us to create tools to help with the design of personalized systems and to prove that such systems behave correctly.

To define that formalization, we consider the idea that a personalized application can be viewed as an extension of a non-personalized application. We consider applications whose behaviour can be expressed with the situation calculus, and show that the personalization can be expressed by a syntactical extension of the equations describing the application. We illustrate this idea using two scenarios of applications to which we add personalization.

Section 2 presents the idea that the personalization can be expressed as an extension of an application. Section 3 explains the formalism we use. Section 4 illustrates our proposal using the formalism with two examples. Section 5 concludes.

II. Personalization

A. Personalization

A personalized application is an application that reflects some features of the user in a user model and apply this model to adapt various aspects of the application to the user.

When no profile is available for a user (for example an unauthenticated user), the application behaves in the same way as with any other unknown user: it behaves like a non-personalized application. The dual of that remark is that the personalized application is the result of adding personalization to a non personalized behaviour.

This approach of splitting the personalized application in various parts may ease the design of such personalized applications. Our goal is to show those personalized applications can be built as extensions of non-personalized applications on a formal level.

B. Approach

To deliver personalized information to a user is a complex task that may require a good amount of intelligence. We build our view of the problem in the context of artificial intelligence rational agents as presented by [2]. For that reason, we base our approach on a logical formalism. Expressing the applications in a logical formalism will

1082-3409/11 $26.00 © 2011 IEEE
DOI 10.1109/ICTAI.2011.43

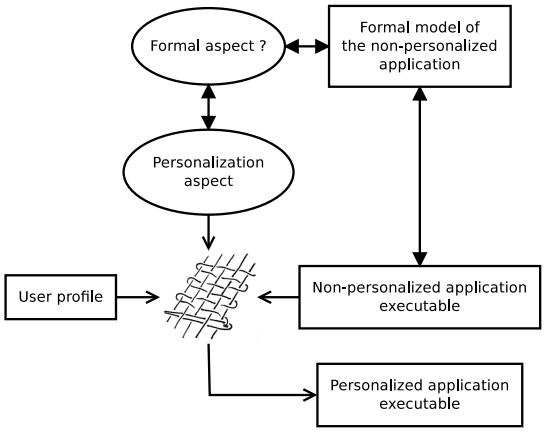


Figure 1. Weaving the personalization

also allow us to highlight syntactically the component that is the personalization.

It's not obvious that any personalized application can be expressed in a logical formalism. At first, we will only consider applications that can be expressed logically. Past works have shown that logical formalisms are relevant for the study of personalized applications [3], [4].

The idea of combining a behaviour with a component is present in the aspect-oriented programming paradigm [5]. In our approach, the behaviour of the application is defined in a formalism such as Golog, and compiled into an executable. The operation of adding personalization can be seen as weaving[1] the non-personalized application with the user profile using a personalization aspect to get the personalized application, as shown in figure 1. This means changing the execution of the non-personalized executable by adding or changing some behaviours based on the user profile.

Our ultimate goal is to find an equivalent of the weaving at the formal level. As in the aspect-oriented paradigm, we would like to weave the formal model of the non-personalized application to generate a formal model of the personalized application.

C. Advantages

This approach opens the way to the creation of automatic or semi-automatic tools for the creation of personalized applications. If we extract the syntactic difference between a non-personalized application and its personalized counterpart, then we'll be able to create a tool transforming a non-personalized application into a personalized one, or at least easing the transformation.

Furthermore, the use of a logical formalism for the characterization of the behaviour of the application enables the proof of various properties of the application.

[1]Weaving is the term used by the aspect-oriented community to designate the action of modifying the behaviour of a program using aspects (isolated representations of significant concepts in a program).

III. Explanation of the formalism

A. The choice of the situation calculus

We want to model the behaviour of a web application interacting with one or many users. For that, we need a formalism able to express the behaviour of the application from its point of view. This would enable us to implement the application using its model. We would also like a formalism that allows to prove the validity of the model.

We choose the situation calculus. This formalism models agents interacting with their environment and acting upon it. It has been used to express agents interacting on the web [6], [7] or to express a hypertext system [8]. It has also been used previously to express the personalization of adaptive hypermedia [3]. We want to explore its use to express the personalization in other types of personalized applications. Because the model is defined logically, it allows to prove all kind of properties on the modelled applications. For example, it enables the proof that there will always be a response from the application, whatever the user does.

B. Situation calculus

The situation calculus [9] is a first order language designed to represent changing worlds. All changes are the result of primitive *actions*. Actions may take parameters. For example, $Drop(x)$ is the action of dropping an object x. We denote the *initial situation*, in which no action has happened yet, by s_0. All changes are the results of applying a sequence of primitive actions to the initial situation. The result of the application of an action α to a *situation* s is the situation $do(\alpha, s)$. Both actions and situations are first order terms.

An introduction to the situation calculus and the extensions we use here can be found in [10].

To fulfill the need to take user response into account, we use guarded action theories [11]. In a guarded action theory, there is a finite number of *sensing functions* (or *sensor functions*), which are unary functions whose argument is a situation that model a sensor available to the agent. These functions are used to define two sets of axioms which specify an action theory:

- Guarded successor state axioms, of the form

$$\alpha(\vec{x}, a, s) \implies [F(\vec{x}, do(a, s)) \equiv \gamma(\vec{x}, a, s)] \quad (1)$$

where α is a fluent formula, F is a relational fluent and γ is a fluent formula. These axioms enable

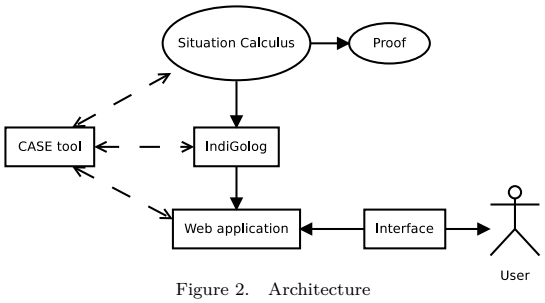


Figure 2. Architecture

regression, i.e. deriving the value of a fluent in a situation from the value in previous situations.

- Guarded sensed fluent axioms, of the form

$$\beta(\vec{x}, s) \implies [F(\vec{x}, s) \equiv \rho(\vec{x}, s)] \quad (2)$$

where β is a sensor-fluent formula, F is a relational fluent and ρ is a sensor formula. These axioms enable sensing the value of an axiom in a certain situation, by using *sensing functions*.

The guards α and β are used to specify when the regression and sensing formulas are usable. This enables us to have fluents that can be sensed in certain situation, and can be modified by actions in other situations.

C. IndiGolog

IndiGolog [12], [13] (extension of Golog [14] and ConGolog [15]) is a high level logic programming language in which actions of a guarded action theory of the situation calculus are instructions. It enables the construction of complex programs by assembling the primitive actions of the situation calculus with various constructs. This language has been used to program robots [15] or agents of the semantic web [6].

D. Architecture

In our framework, the application is composed of an agent in IndiGolog that interacts with users through an interface that abstracts the interaction by providing sensor functions to the agent.

Figure 2 depicts our architecture: on the theoretical level, the action theory defined using the situation calculus enables us to prove some properties of the systems. This action theory is implemented using IndiGolog on the second level. Our web application is expressed using this implementation and interacts with the user. A CASE tool helps with the design of the personalized system.

The aim of the CASE tool is to help the creation of the action theory and its IndiGolog implementation, and to help the addition of personalization to the non-personalized application.

IV. Illustration

We illustrate our proposal by modeling two scenarios using situation calculus, and adding personalization to the modeled applications.

We choose different kinds of personalized systems to build our scenarios: a recommender system and an adaptive educational game.

A. News aggregator

1) Presentation: We consider a simplified scenario of a news aggregator. The system proposes news articles to the user. For each article, the user states whether he read it and liked it, or ignored it. The system then uses this information to recommend the user articles he may like. This recommendation can be based on various factors. For example, the system might recommend articles that are related to articles the user liked.

This scenario can be extended to a full fledged application where news articles sources are handled, recommended, banned and so on.

2) Model: This application handles news articles. The relation $article(x)$ denote that x is a news article. The relation $related(x_1, x_2)$ denotes that two articles are related to each other.

There are 3 fluents: $presented(x, s)$, $liked(x, s)$ and $ignored(x, s)$ which denotes that an article has been presented to, liked by or ignored by the user.

There is one primitive action: $present(x)$ (the agent presents an article to the user).

In the initial situation, nothing has been presented to the user:

$$\forall x.article(x) \implies \neg presented(x, s_0) \wedge \neg liked(x, s_0) \wedge \neg ignored(x, s_0) \quad (3)$$

We only present articles which have never been presented. The fluent *presented* is set to true by the action of presenting.

$$Poss(present(x), s) \equiv \neg presented(x, s) \quad (4)$$

$$True \implies [presented(x, do(a, s)) \equiv a = present(x) \vee presented(x, s)] \quad (5)$$

There are three sensor functions:

- **likedLastArticle**(s) tells if the user has liked an article by calling an external function.
- **passedLastArticle**(s) tells if the user has passed an article by calling an external function.
- **lastArticle**(s) returns the last presented article.

$$\textbf{lastArticle}(s) = x \implies liked(x, s) \equiv \textbf{likedLastArticle}(s) \quad (6)$$

$$\textbf{lastArticle}(s) = x \implies passed(x, s) \equiv \textbf{passedLastArticle}(s) \quad (7)$$

Also, we know that these fluents can change only when the article is proposed to the user. There is a similar axiom for the fluent $passed(x, s)$.

$$a \neq present(x) \implies liked(x, do(a, s)) \equiv liked(x, s) \tag{8}$$

We also introduce the fluent $recommended(x, s)$ that denotes an article is recommended for the user. A possible recommendation strategy could be to recommend articles that are related to articles that the user liked.

$$recommended(x, s) \equiv \exists x'.related(x, x') \wedge liked(x', s) \tag{9}$$

The behaviour of the application expressed in Golog is:

$$(\Sigma(\pi(x).[recommended(x)?;\ present(x)]))^* \\ \rangle\rangle \quad (\pi(x).present(x))^* \tag{10}$$

The first process, which presents recommended articles to the user, has the highest priority. This ensures that only recommended items are presented, as long as there are some. When there are no more recommended articles to present, the other process sends random articles to the user until a new item is recommended.

The first process uses the search operator Σ to make sure only recommended articles are chosen in the nondeterministic choice. No search operator is used in the second process, so the article to present is chosen arbitrarily (possibly randomly).

It is possible to prove that this application will not stop presenting articles before all the articles are read.

3) Personalization: We personalize this application by taking into account the subjects that interest the user. For example, the user has previously been asked what topics he's interested in, and the topics are stored in his profile. This profile is accessible to the agent through the fluent $interestedIn(y, s)$ which denotes that the user is interested in the subject y. We also introduce a fluent $dealsWith(x, y, s)$ which denotes an article x deals with the subject y.

The user interacting with the personalized application will get the same recommendations as with the non-personalized application. In addition to these recommendations, he will also be recommended articles that deal with subject he's interested in. For that, the definition of $recommended(x, s)$ is changed to:

$$recommended(x, s) \equiv \exists x' \ (related(x, x') \wedge liked(x', s)) \\ \underline{\vee \exists y \ interestedIn(y, s) \wedge dealsWith(x, y, s)} \tag{11}$$

The changes introduced by the personalization are not necessarily monotonic. For example, a user could ask not to see articles from a specific author. The personalization would then remove articles from the recommendation pool instead of adding them. The addition to the equation (9) would then be:

$$\wedge \exists y \ author(y, x) \wedge \neg dislikedAuthor(y, s) \tag{12}$$

Further personalization could include using the articles the user like to update his profile.

B. Educative game for children

1) Presentation: We consider a simplified version of a game to teach vocabulary to children. A picture and four words are presented to a child. He as to chose the word corresponding to the picture. Pictures of the same concept are presented again until the concept is known by the child. The concepts presented to the child are selected randomly among those unknown to him. The difficulty is adapted depending on the child's results. The use case is as follows:

- A picture and four words are presented to the child.
  - If the child selects the wrong word: the error is shown with an image of the selected word.
  - If the child selects the right word: a congratulation message is displayed, and the success is remembered in order to know that the child knows the word.
- The system asks another question (a picture and four words).

2) Model: This application handles concepts that we want a user to learn. They are denoted by the relation $concept(x)$.

The fluents are:

- $proposed(x, s)$ — a concept is proposed to the user as an alternative in the current question.
- $rightanswer(x, s)$ — the concept is the right answer to the current question.
- $known(x, s)$ — the concept is known to the user.

The primitive actions are:

- $addWrongAnswer(x)$ — the application adds a wrong answer to the current question.
- $addRightAnswer(x)$ — the application adds a right answer to the current question.
- ask — the prepared question is asked to the user. This is only possible if a right answer has been added to the question.

There are also two sensor functions:

- **lastRightAnswer** returns the right answer to the last asked question.
- **correct** states whether the user answer correctly to the last question.

The preconditions axioms are:

$$Poss(ask, s) \equiv \exists x.rightanswer(x, s) \tag{13}$$

$$Poss(addWrongAnswer(x), s) \equiv \\ \neg proposed(x, s) \wedge \neg known(x, s) \tag{14}$$

$$Poss(addRightAnswer(x), s) \equiv \\ \neg proposed(x, s) \wedge \neg known(x, s) \\ \wedge \neg(\exists x')rightanswer(x', s) \tag{15}$$

The guarded successor state axioms are:

$$True \implies [proposed(x, do(a, s)) \equiv a = addWrongAnswer(x) \vee a = addRightAnswer(x) \vee proposed(x, s) \wedge \neg a = ask] \quad (16)$$

$$True \implies [rightanswer(x, do(a, s)) \equiv \vee a = addRightAnswer(x) \vee rightanswer(x, s) \wedge \neg a = ask] \quad (17)$$

$$a \neq ask \vee \neg rightanswer(x) \implies [known(x, do(a, s)) \equiv known(x, s)] \quad (18)$$

The guarded sensed fluents axioms:

$$\mathbf{lastRightAnswer}(s) = x \implies [known(x, s) \equiv \mathbf{correct}(s)] \quad (19)$$

The Golog program expressing the behaviour of the application is:

proc *addOneWrongAnswer*()
$\Sigma(\pi(x, addWrongAnswer(x)))$
endProc (20)

proc *addOneRightAnswer*()
$\Sigma(\pi(x, addRightAnswer(x)))$
endProc (21)

proc *prepareQuestion*()
addOneWrongAnswer; *addOneWrongAnswer*;
addOneWrongAnswer; *addOneRightAnswer*
endProc (22)

$$(prepareQuestion; ask)* \quad (23)$$

The program defines three procedures:

- *addOneWrongAnswer* adds a wrong answer by selecting one possible wrong answer.
- *addOneRightAnswer* adds a right answer by selecting one possible right answer.
- *prepareQuestion* adds three wrong answer and one right answer in order to prepare a question.

The main program consists only of: prepare a question, ask it and repeat.

3) Personalization: We add personalization two this scenario in two different ways:

- We change the number of possible answers depending on how quickly the user answers.
- We define a notion of difficulty for the questions, and change the difficulty depending on how quickly the user answers.

Firstly, we need to define what a quick answer is. For that, we introduce the sensor function **lastAnswerTime**(s) which returns the time the user took to answer the last question. We decide a user answered quickly to a question if he answered correctly in less than 10 seconds. The fluent $quickAnswer(x, s)$ denotes that the user answered quickly to the question whose right answer is x.

$$\mathbf{lastRightAnswer}(s) = x \wedge correct(s) \implies quickAnswer(x, s) \equiv \mathbf{lastAnswerTime}(s) < 10 \quad (24)$$

$$a \neq ask \vee \neg rightanswer(x) \implies quickAnswer(x, do(a, s)) \equiv quickAnswer(x, s) \quad (25)$$

We also introduce a functional fluent $totalQuickAnswer(s)$. $totalQuickAnswer(s) = n$ that denotes that n different questions have been answered quickly (there are n different x such that $quickAnswer(x, s)$)[2].

Using that fluent, we can define a first approach of personalization for this application: changing the number of possible response depending on how quickly the user answered so far. Here, we will add one possible answer each 10 fast answered questions.

We first need to re-define the *prepareQuestion* procedure to create a question with a variable amount of answers, then modify the main program to use it. $prepareQuestion(n)$ prepares a question with n wrong answers.

proc *prepareQuestion*(n)
if $n = 0$ **then** *addOneRightAnswer*
else *addOneWrongAnswer*;
$prepareQuestion(n - 1)$
endProc (26)

$$(prepareQuestion(3); ask)* \quad (27)$$

The personalized application's main program is:

$$(prepareQuestion(3 + totalQuickAnswer()/10); ask)* \quad (28)$$

Our second approach of personalization requires the question to have a difficulty. Each concept must have a difficulty. The fluent $difficulty(x, n)$ denotes that the concept x have the difficulty y, where n is a number. Any concept chosen for a question must be easier than the current difficulty. The current difficulty is defined from the number of quick answers so far.

$$currentDifficulty(s) = totalQuickAnswer(s)/10 \quad (29)$$

The precondition axioms for *addWrongAnswer* (14) and *addRightAnswer* (15) change to become:

$$Poss(addWrongAnswer(x), s) \equiv \neg proposed(x, s) \wedge \neg known(x, s) \wedge \exists n[difficulty(x, n) \wedge n < currentDifficulty(s)] \quad (30)$$

[2]The actual definition of $totalQuickAnswer(s)$ is left for now because defining "There exists n different terms" with a variable n can be tricky using only first-order logic, and is much easier to implement in a language such as prolog than to define.

$$Poss(addRightAnswer(x), s) \equiv \neg proposed(x, s) \wedge \neg known(x, s) \wedge \neg(\exists x')rightanswer(x', s) \wedge \underline{\exists n[difficulty(x, n) \wedge n < currentDifficulty(s)]} \quad (31)$$

C. Analysis

We have modeled two scenarios of applications and added personalization to them. The personalization was easily added by adding the relevant fluents (mostly representing the user profile) and modifying the equations to use those fluents.

Furthermore, we can see that the modifications of the application belong in one of two categories:

- Modifications in order to add flexibility while keeping the same behaviour: the golog program (26) is a rewrite of the program (22) in which the number of wrong answer can vary. *prepareQuestion*(3) as define in (26) has exactly the same effect as *prepareQuestion* as defined by (22).
- Modifications of the behaviour such as:
  - Modifications of parameters: the program (28) is a modification of the program (27) in which the argument to *prepareQuestion* can vary depending on the user profile;
  - Addition or modification of conditions taking into account the user profile: the equation (11) is a modification of the equation (9) by adding a condition to take into account the user profile.

V. Conclusion

We developed the idea that a personalized application can be seen as a formal extension of a non-personalized application. We illustrated this idea by modeling two scenarios of personalized applications. These examples showed that once the application is defined in the logical formalism of situation calculus, personalization can easily be added by minor modifications of the formalization of the non-personalized application.

The next step in our work is to extract and characterize the transformation between non-personalized and personalized application in order to apply it automatically, or at least provide tools able to help in the transformation.

References

[1] N. Stash, A. Cristea, and P. D. Bra, "Adaptation languages as vehicles of explicit intelligence in Adaptive Hypermedia," *International Journal of Continuing Engineering Education and Life Long Learning*, vol. 17, no. 4, p. 319–336, 2007.

[2] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice hall, 2010.

[3] C. Jacquiot, Y. Bourda, F. Popineau, A. Delteil, and C. Reynaud, "GLAM: A generic layered adaptation model for adaptive hypermedia systems," in *Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2006, p. 131–140.

[4] M. Baldoni, C. Baroglio, and V. Patti, "Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions," *Artificial Intelligence Review*, vol. 22, no. 1, p. 3–39, 2004.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP*. SpringerVerlag, 1997.

[6] S. McIlraith and T. Son, "Adapting Golog for programming the semantic web," in *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, 2001, p. 195–202.

[7] ——, "Adapting golog for composition of semantic web services," in *Principles of knowledge representation and reasoning-international conference*, 2002, p. 482–496.

[8] R. Scherl, "A GOLOG Specification of a HyperText System," in *Logical Foundations for Cognitive Agents*. Springer-Verlag, 1999, p. 309–324.

[9] J. Carthy and P. Hayes, "Some philosophical problems from the standpoint or artificial intelligence," *Machine Intelligence*, 1969.

[10] G. Dubus, F. Popineau, and Y. Bourda, "Situation calculus and personalized web systems," in *International Conference on Intelligent Systems Design and Applications*, 2011.

[11] G. D. Giacomo and H. Levesque, "Projection using regression and sensors," in *International joint conference on artificial intelligence*, vol. 16, 1999, p. 160–165.

[12] G. D. Giacomo, H. Levesque, and S. Sardina, "Incremental execution of guarded theories," *ACM Transactions on Computational Logic (TOCL)*, vol. 2, no. 4, p. 495–525, 2001.

[13] G. Giacomo, Y. Lespérance, H. Levesque, and S. Sardina, "IndiGolog: A high-level programming language for embedded reasoning agents," *Multi-Agent Programming:*, p. 31–72, 2009.

[14] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, "GOLOG: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 31, no. 1-3, p. 59–83, 1997.

[15] G. D. Giacomo, Y. Lespérance, and H. Levesque, "ConGolog, a concurrent programming language based on the situation calculus," *Artificial Intelligence*, vol. 121, no. 1-2, p. 109–169, 2000.

2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT)

Parametric reasoning agents extend the control over standard behaviors

Georges Dubus, Fabrice Popineau, Yolaine Bourda
SUPELEC Systems Sciences (E3S)
Computer Science Department
Gif-sur-Yvette, France
Email: {georges.dubus,fabrice.popineau,yolaine.bourda}@supelec.fr

Jean-Paul Sansonnet
LIMSI-CNRS
BP 133
F-91403 Orsay cedex France
Email: jps@limsi.fr

***Abstract*—In this paper, we study how to extend the control over the behavioral space of logically specified agents so as they can take into account parameters issuing from constraints and/or preferences that are external to their core reasoning process. Our approach is supported by the proposition of a set of generic transformations to apply to the original agent program that is expressed in IndiGolog, a variant of the Golog language. Several of these transformations exploit the non-determinism present in IndiGolog programs. We also propose an automatic process to apply all those transformations on the basis of a set of parameters. Three case-studies show the significance of the approach in situations where agents are in interaction with human users.**

I. Introduction

An agent-based model [1], [2] is a generic way of considering the interaction between a user and a computer system. Logically specified agents provide the extra advantage of logical reasoning, and the possibility to prove the soundness of their expression and behavior [3], chapters 2 and 7 of [4]. Such agents run an agent program specified in a dedicated language like Golog [5], DyLog [6], Placa [7], *etc.*

Since the beginning, research on logically specified agents has focused on the rationality and the genericity of the proposed behaviors. However, for some years now, there has been a fast development of applications and services where artificial agents are in interaction with the general public. Indeed, studies about agent/human interaction have shown that variability in the agent's behavior are required by the users [8]: subjects often criticize the mechanical behavior of traditional agents, especially during interactive sessions. Moreover the expression by an agent of human-like behaviors has a strong impact upon its believability [9] and its acceptability [10]. Writing a new agent for each variation of the behavior is obviously not an option. So there is a need for agents that adapt their behavior according to parameters issuing from the situation of interaction.

For long, designers have put aside those parameters: the human user with whom the agent is in interaction is a generic user without any specific preferences, and the agent program is focused on rationality, hence it does not have a personality. There are two challenges in handling those parameters. The first challenge is the resulting combinatorial explosion, and logic programs are very sensitive to this dimension. The second and possibly deeper challenge is to guarantee that all the various programs obtained by assigning different values to those parameters share a common behavior. This is the main goal of our proposal.

These considerations lead us to distinguish two main classes of agent behaviors:

- *Standard behaviors* are executed by traditional rational agents;
- *Parametric behaviors* are executed by agents adding to their rational process the handling of parameters associated with the situation of interaction.

We propose a constructive approach to the notion of a parametric reasoning agent[1], by extending a standard agent. The parametric behavior is derived from the standard behavior by applying transformations, hence extending the control over the standard behavior. We define a set of transformations that produce those variations of a given agent program while ensuring its overall functionality. We also define an automatic process that takes a standard agent program and an extension of its action theory provided by the application designer and results in a parametric agent program. We focus on two significant classes of situations where agents are in interaction with human users: *personalized* systems [13] and *personified* systems [14], [15]. We are interested in the action theory in which the agent program is executed. Hence, we will consider only features or personality traits that are relevant to change the way an action is undertaken. Both cases share the same principle that a profile is attached to the agent. In one case, it is the profile of the user and the agent uses this profile to satisfy the user; in the other case it is the profile the agent is expected to run with. We consider agents whose program is expressed with IndiGolog, a programming language for agents described in section II. The process itself and the definitions related to the profile are described in section III, and their use is demonstrated using three case-studies in section IV.

II. Agents

We first introduce the Golog languages family and we explain why we need one of the most advanced form of Golog, namely IndiGolog. Next, we detail two directions in which we expect to apply our program transformation technique :

[1] In the literature, the term 'parametric agent' refers also to several research topics, where it applies to the design process phase rather than the standard behavior of the agent. For example, in Parametric Design Agents (PDA), it is associated with reusable components for knowledge models [11] or intelligent parametric design tools supporting collaborative engineering [12].

978-1-4799-2902-3/13 $31.00 © 2013 IEEE
DOI 10.1109/WI-IAT.2013.105

personalization of an application interacting with a user and personification of a program agent.

A. The situation Calculus and Golog

The situation calculus [16] was designed to represent changing worlds as a set of formulas in first order logic. All changes are the result of primitive actions. There is a special constant s_0 that denotes the initial situation, and a function symbol do where $do(a, s)$ denotes the successor situation to s resulting from performing a. The state of the world in a situation is described by a set of fluents – predicates that depends on the situation. To define the world of an agent, the programmer creates an action theory representing the agent's belief about the state of the environment and the preconditions and effects of the actions.

Golog [5] is a procedural programming language defined on top of the situation calculus. It provides a set of constructs to express complex procedural programs. The interpreter for the language uses a situation calculus action theory to reason and find a provably correct execution of the program.

A number of constructs are available to define complex programs for the agent.

- a — primitive actions: a situation calculus action term.
- $\phi?$ — tests: a test on ϕ in the middle of a sequence blocks if ϕ is false. This is used to block branches in case of a non-deterministic choice: only a branch where ϕ is true can proceed.
- $\delta_1; \delta_2$ — sequences: δ_1 is executed, then δ_2 is executed.
- $\delta_1 | \delta_2$ — non-deterministic choice of actions: successfully executed if either δ_1 or δ_2 is successfully executed.
- $\pi(x)\delta(x)$ — non-deterministic choice of parameters: successfully executed if $\delta(x)$ is successfully executed for some value of x, where x is a free variable in $\delta(x)$.
- δ^* — non-deterministic iteration: δ is executed zero or more times.
- **if** ϕ **then** δ_1 **else** δ_2 — conditionals: if ϕ is true, δ_1 is executed, else δ_2 is.
- **while** ϕ **do** δ — while loops: δ is executed while ϕ is true.
- **proc** $P(\vec{v})\delta$ **endProc** — procedure: defines a procedure P that may be called later in the program.

Plain Golog does not provide a way for the agent to take into account changes in the environment that do not come from his own actions. This is a problem for the expression of embedded agents, which need to update their internal representations according to changes in the environment. IndiGolog [17] is an extension to Golog that provides a solution to that problem by using sensing. It enables the user to define sensing actions which update the value of sensing fluents from an external interface during online execution. The way IndiGolog handles sensing creates a clear cut between the agent and the environment which is needed for embedded agents. Indigolog is a rich language that has been used for example to express the personalization of adaptive hypermedia [18] or other kinds of personalized systems [19].

B. Application domains

In this study, we consider two main domains involving interactions between agents and human users. Each domain defines a class of applications for parametric agents with a specific orientation: user model and agent model. Each of these models is described by a set of parameters which are human personality traits in our application domains.

We propose to start with a standard behavior: one which is known to work in the absence of any parameter. We then transform this standard program to obtain an altered program. The standard program contains non deterministic alternatives which, in the absence of any specific parameter, are resolved by random choice. The altered program makes use of parameters to resolve those alternatives by explicit choice. New choices may also be introduced with conditions. We guarantee that when no parameter is given, the altered program falls back on the standard program. However, agents with distinct profiles all behave according to their specific profile.

1) Personalization: A personalized system is a system that reflects some features of the user in a user model and applies this model to adapt various aspects of the system to the user. Adaptive Hypermedia [20] are an example of such system.

Most of the time, personalization is achieved by plugging (or in the worst case hardwiring) ad-hoc rules for handling predetermined situations. This is not bullet proof and in many situations it may lead to an incomplete agent and also to difficulties in the debugging process of the agents.

Moreover [21] showed that there is no formal method to build such an agent. The main reason is that there is no formal definition of personalization. Intuitively, the behavior of a personalized version of an agent program should follow more or less closely its non-personalized version. The problem is with the definition of "more or less closely". Actually, for δ a (non-personalized) program and ψ a profile related formula, every **if** $(\psi)\delta'$ **then** δ could be a personalized version of δ, without any further condition on δ'. And we certainly do not want that because the global behavior would be out of control.

We make use of parametric agents to implement a restricted and well defined class of personalized agent programs. We impose that the personalized behavior differ only on the result of non-deterministic choices, or by modifications of actions. We also make sure that an empty profile does not change the program.

2) Personification: According to [22] and more recently to [23], the decision making process of rational agent can be influenced by psychological features such as personality traits. These authors claim that it makes their behavior look more 'human-like' (notion of 'believable agents' as defined by Bates [9]), thus increasing their acceptability factor [10], which is considered useful in new applications where agents are more and more in interaction with humans.

Research on computational implementation of psychological profiles [24], has focused recently on psychological theories involving *trait* taxonomies, such as the Five Factor Model (FFM) [25] and its subsequent versions, enriched with *facets*, such as the NEO PI-R taxonomy [26]. Using the psychological profile of the agent as parameters would be a new way to implement these psychological agents, or even to turn a strictly rational agent into a rational-and-psychological agent.

Assume an *affective* robot [27], [28], provided with psychological features – say personality traits [25], such as lazy,

brutal, cautious *etc.* Assume the robot is given a goal to visit-a-house, composed of n rooms r_i. This goal exhibits possible flexibility in its achievement because the robot is not strictly compelled to visit all rooms.

Now suppose that a given agent is provided with the psychological trait cautious; then it may refuse to enter locked rooms r such that $locked(r)$, while another agent, provided with trait brutal, may break the door of a locked room. Both agents reach the goal: they both have visited the house their own way.

III. Transformations over IndiGolog programs

In this section, we describe an automatic alteration process for IndiGolog programs to transform a standard agent into a parametric agent. First, at III-A, we introduce a new IndiGolog operator that we need in the transformed programs. Then, at III-B and III-C, we introduce the profile and explain how to link profiles to the actions of the program through affinity. Next, we introduce basic transformations that slightly change the behavior of an IndiGolog program at sections III-D and III-E. Finally, we define the alteration process that uses affinity to apply the basic transformations at section III-F.

A. Changes Needed in IndiGolog

IndiGolog has a non-deterministic choice operator |, but it does not allow to specify *preferences* upon the execution of one branch or the other, which we need to express the altered agent. We introduce the nondeterministic choice with preference operator denoted by $\rangle$. $\delta_1 \rangle \delta_2$ means "try δ_1, and in case of failure try δ_2". Note that as stated in [17], the IndiGolog implementation tries nondeterministic branches left-to-right, so the implemented | has the same behavior as our new $\rangle$ operator. However, it is of theoretical importance to clarify when the choice between alternatives is random, and when there is a preference between those alternatives. The formal semantics for the operator are given by formulas 1 and 2 in terms of IndiGolog transition semantics : $Trans(\delta, s, \delta', s')$ means that by executing one elementary step of the program δ in the situation s leads to the situation s' with the program δ' remaining to be executed ; $Final(\delta, s)$ means that the execution of δ is finished in s.

$Do(\delta, s, s')$ [17] means that there is an execution of program δ that starts in s and terminates in s'.

$$Final(\delta_1 \rangle \delta_2, s) \equiv Final(\delta_1, s) \vee \neg\exists s".Do(\delta_1, s, s") \wedge Final(\delta_2, s) \quad (1)$$

$$Trans(\delta_1 \rangle \delta_2, s, \delta', s') \equiv \exists s".Do(\delta_1, s, s") \wedge Trans(\delta_1, s, \delta', s') \vee \neg\exists s".Do(\delta_1, s, s") \wedge Trans(\delta_2, s, \delta', s') \quad (2)$$

B. Attributes and Profiles

First of all, the application designer must define what can alter the behavior, namely a set of attributes which are linked to actions, and are used to decide if an action must be favored or avoided. The nature of the attributes depends on the type of alteration that is done to the agent. The relation between attributes and actions is detailed in section III-C.

Definition 1. An attribute is an atom. The set of all attributes A is defined by the application designer.

The behavior of an agent is altered by providing the agent with a positive or negative view toward attributes. A profile encompasses all views of a given agent. For each attribute, the agent can be absolutely opposed, slightly opposed, neutral, slightly favorable or absolutely favorable. These are respectively denoted by the integer values -2, -1, 0, 1 and 2 [23]. We consider that values of 1 and -1 indicate preference and should not alter much the program, but values of 2 and -2 indicate requirement, and may lead to invalid programs if they cause the agent to veto a critical part of the program. The 0 value denotes ambivalence or neutrality. The application designer should keep this in mind when defining a profile.

Definition 2. A profile is a function from the set of all attributes A to $\{-2, -1, 0, 1, 2\}$.

In the case of a personalized agent, the attributes refer to things that may or may not please the user, and the profile tells what pleases and displeases a specific user. In the case of a personified agent, the attributes refer to psychological connotations of actions, that the personified agent wants to favor or avoid. Such profile can be viewed as a psychological profile of the agent since it tells what kind of behavior it likes or dislikes.

C. Attributes and affinity of an action

To automatically alter a program, we need a way to tell if an action (*resp* a subprogram) is preferred to another action (*resp* another subprogram) with regard to a given profile. This requires that actions carry extra information, outside of the situation calculus, that can be used to know whether they are desirable, neutral or disliked, with regard to a given profile. This is implemented by attaching to actions attributes, similar to those in the profile.

For example, in the case of an affective robot with only few possible traits, we may have $A = \{lazy, brutal, cautious\}$. A possible profile p is defined by $p(lazy) = 2$, $p(brutal) = -1$ and $p(cautious) = 0$

Definition 3. $attributes$ is a function from the set of actions to the set of sets of attributes. For a given action a, $attributes(a)$ is the set of attributes attached to a and it is provided by the application designer.

We join the actions and the profile to get the affinity of an action to a profile: a numeric value telling whether an action is suitable for a profile.

Definition 4. The affinity function aff_p maps actions to numbers and is defined by

$$aff_p(a) = \sum_{at \in attributes(a)} p(at)$$

Thus, an action a_1 is preferable to another action a_2 for a given profile p if $aff_p(a_1) > aff_p(a_2)$. To transform entire programs, we need to know the affinity of a program. Thus, we extend the domain of aff_p to all programs with an inductive definition. The execution of the program not being known at compile-time, we must do approximations, like assuming that

a non-deterministic choice is random and that the affinity is the mean of its components.

Definition 5. We extend the affinity function aff_p to programs by inductive construction on all the operators of IndiGolog. For parsimony, only some cases are shown.

$$\begin{aligned} aff_p(\phi?) &= 0 \\ aff_p(\delta_1;\delta_2) &= aff_p(\delta_1) + aff_p(\delta_2) \\ aff_p(\delta_1|\delta_2) &= \frac{1}{2}(aff_p(\delta_1) + aff_p(\delta_2)) \\ aff_p(\delta_1\rangle\delta_2) &= aff_p(\delta_1) \\ aff_p(\pi x.\delta) &= aff_p(\delta) \\ aff_p(\delta^*) &= aff_p(\delta) \end{aligned}$$

Actions with arguments (which model verbs with objects) must be defined differently, because the attributes can depend on the argument (the meaning of an action depends on the objects on which it is applied). For example, the same action read can be hard when reading a scholar book but can be easy when reading a newspaper.

Definition 6. The function $attributes_a$ takes an action with an argument $a(x)$ and results in a list of pairs $< predicate, attribute >$. For each of those pairs, $a(x)$ has the attribute when x satisfies the corresponding predicate. This function is provided by the application designer.

$$attributes_a(a(x)) = \{< \phi_1, att_1 >, \ldots, < \phi_n, att_n >\}$$

See section IV-A for an example.

D. Program transformation

Our program transformation process manipulates a set of basic transformations which are defined in this subsection and the next one. The formal way to apply them is studied in subsection III-F.

As said above, our goal is to alter the behavior of agents by changing how they make choices. This requires the program to be under-specified: the execution must not be fully constrained by the program. Instead, the program must present choice points so to have some leeway for the execution and let the agent make choices at runtime.

Choices in IndiGolog programs appear in the non-deterministic choices constructs | and π. We introduce transformations altering the choice in both constructs, that can be applied in any program where those constructs appear.

First of all, it is possible to replace any non-deterministic choice of a program by a preferential choice. The first transformation T1 is just a mean to force an order of evaluation for the two programs.

Transformation 1 (T1). $\delta_1|\delta_2 \rightarrow \delta_i\rangle\delta_j \qquad i,j \in \{1,2\}, i \neq j$

The next transformation T2 changes the order of evaluation of the argument in the non-deterministic choice of argument so that arguments satisfying a predicate ϕ are evaluated first. This is done by duplicating the construct and altering the first one so that the only valid executions are those where $\phi(x)$ is true, falling back to the original program if no such x exists. The fallback part guarantees that if the original program can be executed, then so can the transformed program.

Transformation 2 (T2). $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?;\delta(x)\rangle\pi x.\delta(x)$ Where ϕ a formula containing only fluents.

Transformations T1 and T2 guarantee that the transformed program is as successfully executable as the original program. However, it is desirable to restrict the execution, for example to prevent some behavior that is not suitable for some user. We introduce two new transformations T3 and T4, which are similar to the first two, but with the fallback part removed. Transformation T3 entirely removes the non-determinism. These transformations are to be applied with care, because they can lead to programs that cannot be executed.

Transformation 3 (T3). $\delta_1|\delta_2 \rightarrow \delta_i$ with $i \in \{1,2\}$

Transformation 4 (T4). $\pi x.\delta(x) \rightarrow \pi x.\phi(x)?;\delta(x)$

Theorem 1. Let δ' be the result of the tranformation T1 or T2 applied to a program δ. If δ has a possible execution, then δ' has one too.

A program tranformed with T1 and T2 has the exact same possibilities as the original program, only evaluated in a different order. Thus, if the original program has a possible execution (ie a solution to its problem), the transformed program will at least one possible execution and will solve the problem too. Transformations T3 and T4, on the other hand, remove possibilities, and may break a program (by removing crucial parts). Therefore, they must not be applied lightly. For that, the alteration process ensures to only use those two when the profile present extreme values (2 or -2), thus giving the responsibility to the application designer.

E. Actions transformation

Until now, we have considered that the program to be transformed is under-specified and has non-deterministic choices and that transformations can be used to direct choices. This approach helps ensuring that the transformed behavior does not diverge much from the standard behavior. However, it is possible to introduce new choices in a program, as long as the new options are only variations of what is already there.

Thus, we want to modify programs by replacing actions by other actions that have slightly different pre and post-conditions, such as opening a door versus breaking a door open, or the same definition but different effects outside of the situation calculus theory, such as displaying a sentence in a text in bold font or in italic font.

This requires the application designer to group similar actions in families. A family is a set of actions that express the same operation but executed in a different way. The exact meaning of "same operation" depends on the domain and on what the actions are. Like for the actions, it is the application designer's responsibility to define a family of actions, and to decide which actions should be considered as doing the same operation.

Definition 7. The predicate $family(F)$ is true if F is a family of actions. $family$ is defined by the application designer.

An example of family is described in section IV-C

Transformation 5 (T5). If $A = \{a_1, \ldots, a_n\}$ is a family of actions

$$\forall a_i \in A \quad a_i \rightarrow a_1|\ldots|a_n$$

Transformation T5 replaces a given action by a choice among all actions in its family. It can be used in conjunction with transformation T1 to reorder the actions and replace $|$ by $\rangle$, in order to favor some actions upon others, and with transformation T3 to remove some actions from this choice.

Since the actions of a family achieve the same operation, replacing one by another does not interfere with the overall functionality of the agent.

F. Alteration process

The alteration process of a program δ is made of three steps that are executed in order: transformation of actions which are part of a family, transformation of non-deterministic choices of programs, and transformation of non-deterministic choices of arguments. It uses the affinity of actions and programs to determine how to apply the transformations. Therefore, on top of the usual IndiGolog application definition, the application designer is required to give additional definitions that are used by the process, summarized in the following table.

Name	Description	Section
$attributes$	list of attributes for each action	III-C
$attributes_a$	for actions with arguments	III-C
$family$	family of actions	III-E
p	profile	III-B

1) Actions: The program is recursively scanned to find actions a_0 such that there exists a family F and $a_0 \in F$.

For each such action a_0, a_0 is replaced by a choice between the actions of the family with T5. Then actions with non-positive affinity are dropped with T3 (we only introduce actions with positive affinity), a_0 is dropped only if its affinity is negative (we keep the original as fallback, except if it non desirable), actions with equal affinity are grouped in a non-deterministic choice and the preferential choice of T1 is used to link the groups (ensuring that actions with the most affinity are considered first during the execution, and actions with the same affinity are equally considered), resulting in the pseudo-program (3).

$$[\,\rangle_{k=2..1}(|_{a_i \in F | aff(a_i)=k} a_i)\,]\,(\rangle\ a_0 \text{ if } aff(a_0) = 0) \quad (3)$$

With a standard profile ($\forall a\ p(a) = 0$), there are no actions with positive affinity, and the result is a_0, the same as in the original program.

2) Choices of programs: The program is recursively scanned to find non-deterministic choices of programs. For each program of the form $\delta_1|\delta_2$, starting by the innermost, affinity of δ_1 and δ_2 are compared, and the one with the most affinity is preferred to the other using transformation T1.

If the difference of affinity exceeds 2 then the option with less affinity is removed using transformation T3.

With a standard profile, $aff(\delta_1) = aff(\delta_2) = 0$, and the program remains unchanged.

3) Choices of arguments: The program is recursively scanned to find non-deterministic choices of arguments. For each program $\pi x.\delta(x)$, the following steps are taken:

1) $\delta(x)$ is scanned to find actions a_i that have x as an argument.
2) For each action $a_i(x)$, $attributes_a(a_i)$ is a list of predicates associated to attributes. We gather all predicates in P. For each predicate, the associated attribute might be regarded by the profile as positive, negative, or standard. Thus, we might want to introduce the predicate, its negation, or not introduce it. If there are n predicates, this means that there are 3^n possibilities to study.
3) For each candidate of the form $c_k = \{\phi_1, \phi_3, \neg\phi_4, \ldots\}$, we compute the affinity of δ by giving each action an affinity equal to:
$$aff_p(a(x)) = \sum_{<att,\phi>} \begin{cases} p(att) & \text{if } \phi \in c_k. \\ -p(att) & \text{if } \neg\phi \in c_k. \\ 0 & \text{otherwise.} \end{cases}$$
4) We take the candidate c_k that maximizes the affinity of δ, and apply transformation T2 to replace $\pi x.\delta(x)$ with $\pi x.\psi(x)?;\delta(x)$ where $\psi(x)$ is the conjunction of the predicates of c_k. This ensures that arguments that satisfy $\psi(x)$ are considered before the others in the choice. If the affinity for c_k is superior to 2, T4 is used instead to make the condition mandatory. In case of equality of affinity, the candidate with the least predicates is taken.

With a standard profile, all affinities are null and the empty candidate is favored, thus leaving the program unchanged.

These three steps are executed in this order for a given profile p, and provide an automatic alteration process for IndiGolog programs.

IV. Case-studies

In this section, we discuss three cases-studies, involving distinct scenarios of alteration:

1) Personalization of recommender systems: recommender systems interact heavily with users. A lot has been done in the area of finding a recommendation that best suits each user. Most systems rely on statistics and/or machine learning: these systems cannot provide an explanation why something is recommended. We are interested in a logical approach to this problem and an IndiGolog agent in this area would be able to provide such an explanation.
2) Personalization of an Educational Hypermedia System (EHS): in these systems, agents have to deal with very complex environments. Agents are supposed to teach to users who expect a human level interaction. This is a good example for writing complex strategies that may need to be altered to take into account different user profiles.
3) Personification of a robot agent: in this situation, the agent is not directly in interaction with a user, at least during the achievement of the goal. However, a user can observe robot's actions and interpret its behavior in psychological terms.

A. Personalization of Recommender Systems

This first case-study deals with a Web-based application that recommends news articles. News articles are attached to topics. The standard agent chooses random articles to recommend, independently of the topic.

Depending on the user profile, we want the transformed program to avoid (or to prefer) articles that deal with certain

topics like sports or economy. We hide the details of implementation and focus on the IndiGolog program that is altered. We assume the existence of domain predicates to mark articles and their topics, as well as an action to push the recommendation to the user. The standard program is:

while $continue$
  do $\pi a.(article(a)?; \text{recommend}(a))$

For the transformation to work, we must define the attributes of the action. In this case, this is simple: the action has attributes *sport* if the article deals with sports, and *economy* if the articles deals with economy. There are some repetition in this definition, but it could easily be generated from the list of topics instead of being entered manually.

$$attributes_a(recommend(a)) = [< dealsWith(a, sport), sport >, < dealsWith(a, economy), economy >]$$

We execute the transformation processes on this program using two different profiles p_1 and p_2. $p_1(sport) = -2$, $p_1(economy) = 0$, $p_2(sport) = 1$, $p_2(economy) = -1$. The resulting program for p_1 only recommends articles that do not deal with sports:

while $continue$
  do $\pi a.(\neg dealsWith(sport, a)?; article(a)?; \text{recommend}(a))$

The program for p_2 prefers articles that deals with sport and tries to avoid articles dealing with economy, but still selects them if nothing else is available:

while $continue$
  do $\pi a.(dealsWith(sport, a) \wedge \neg dealsWith(economy, a)?; article(a)?; \text{recommend}(a)) \rangle \pi a.(article(a)?; \text{recommend}(a))$

B. Personalization of an EH System

The second case-study deals with an EHS which contains pedagogical resources provided with a prerequisite relation. The user can access all the resources. In order to help the user find relevant resources, the altered version personalize the presentation by only providing relevant resources, *i.e.* those who are needed for the user to reach a given pedagogical goal.

There are two domain predicates: $resource(r)$ which means r is a resource, and $prereq(r, r')$ which means that r is a prerequisite of r'. We also use $prereq^*(r, r')$ which is the transitive closure of $prereq$. We assume a subprogram for displaying text on the student's screen. The standard program displays every available resource. The * operator executes its body as many times as possible.

$$(\pi r.resource(r)?; \text{display}(r))^*$$

To execute the transformation, we need to provide information about the attributes. We state that $display(r)$ has for attributes $goal(r')$, if it advances toward the goal r_i for each resource r_i, *i.e.* if r_i is a prerequisite of r_i.

$$attributes_a(display(r)) = [< prereq^*(r, r_1), goal(r_1) >, \ldots, < prereq^*(r, r_n), goal(r_n) >]$$

We can then transform the program using any profile stating a goal. For example, a profile where the goal is to be able to read and understand an article named "Mastering databases": $p(goal(masteringDatabases)) = 2$. It is set to 2, because we do not want to see anything that does not help for our goal. The resulting program with p only show articles useful for our goal:

$$(\pi r.prereq^*(r, masteringDatabases)?; resource(r)?; \text{display}(r))^*$$

C. Personification of Robot Agent

The third case-study deals with a robot agent that circulates inside a building. There are several rooms, all closed and locked, containing keys to other rooms, or gold. The robot has an inventory of infinite size, containing things it caries. It can perform only two actions:

1) open a door d by using a key in its inventory: openWithKey(d);
2) once a door is open, the agent can take any item i in the room: take(i) .

The goal of the agent is to find and pick up the gold.

The program of the agent makes use of a procedure loot(r), to take all the items in a room. The main program loops over the closed doors and opens them until the gold is collected.

proc $\text{loot}(r)$
  while $\exists i.position(i) = r;$
    do $\pi i.(position(i) = r)?; \text{take}(i)$
endProc
proc main
  while $\neg position(gold) = inventory;$
    do $\Sigma(\pi r.closed(r)?; \text{openWithKey}(r); \text{loot}(r))$
endProc

In the initial situation, all doors are closed, room 1 contains the key to room 3, room 2 contains the gold, room 3 contains the key to room 2, and the agent has the key to room 1 in his inventory. A possible execution of this program is: openWithKey(1), take(key(3)), openWithKey(3), take(key(2)), openWithKey(2), take(gold).

We want to alter this agent by attaching to it personality features, as defined in section II-B. For that, we must add some flexibility to the agent to let it make meaningful choices. We introduce two new actions, that are different ways to open door: by breaking them (*break(d)*), or by picking the lock using a pick (*lockpick(d)*) that must be found first.

We attach to the agent a profile containing three traits: outlaw, brutal and practical. Technically, in FFM/NEO PI-R taxonomy [26], the trait outlaw is located in facet Dutifulness of Trait Conscientiousness, which covers the notions of: *loyal*-ness, *virtuous*-ness, *lawabiding*-ness and *truthful*-ness. brutal is associated with three facets: the negative pole of facet Tender-mindedness of Trait Agreeableness; and the facets Angry-Hostility, Impulsiveness of trait Neuroticism. The same method can be applied to a large subset of FFM/NEO PI-R.

To use the transformation process, we need to define that openWithKey, break and lockpick form a family (they all are ways to open doors). We also need to give attributes to

actions, which are in fact the psychological traits that favor these actions. Break gets the attribute brutal, lockpick gets the attribute outlaw, and openWithKey gets the attribute practical.

The attributes and families being defined, we can see how the alteration process affects the program, and the execution. Firstly, let's try a profile p_1 of an agent that is outlaw, dislikes brutality, and is not very practical: $p_1(brutal) = -1$, $p_1(outlaw) = 1$, $p_1(practical) = 0$. The program is modified to favor picking lock over using keys, but never breaks doors:

proc $\text{main} - p_1$
  while $\neg position(gold) = inventory$;
    do $\Sigma(\pi r.closed(r)?;$
      $\text{lockpick}(r)\rangle\text{openWithKey}(r);$
      $\text{loot}(r))$
endProc

Here is an example of execution in the same settings as before (with a lockpick added in room 3): openWithKey(1), take(key(3)), openWithKey(3), take(key(2)), take(pick), lockpick(2), take(gold).

Then, let's try a profile p_2 of a violent agent that is normally outlaw, but is very practical: $p_2(brutal) = 1$, $p_2(outlaw) = 0$, $p_2(practical) = 2$. In the resulting program, there is the ability to break doors, but a preference for opening doors with keys if possible.

proc $\text{main} - p_2$
  while $\neg position(gold) = inventory$;
    do $\Sigma(\pi r.closed(r)?;$
      $\text{openWithKey}(r)\rangle\text{break}(r);$
      $\text{loot}(r))$
endProc

An execution is: openWithKey(1), take(key(3)), break(2), take(gold).

In all three versions, the agent reaches its goal, through different means. We transformed the program according to a profile to give the agent a preference on the means to employ. Each altered agent solves the problem using its preferred actions if possible, falling back to others when needed, thus ensuring that any altered agent is able to solve the problem in some way.

V. Related Work

A. Different approaches to personalization

Other works have already used logically defined rational agents to perform personalization and adaptation tasks. [6] uses DyLog, a modal logic programming language, to create a personalized agent for adaptive tutoring. Their user profile (in this case, the learning goal) is a fluent like any other. Our approach is different because we set the user profile as parameters of execution apart from the domain fluents. Furthermore, their approach is an ad-hoc implementation of the personalized agent: there is no notion of standard agent from which the agent is evolved, so there is no way to guarantee the personalization does not interfere with the standard behavior of the agent.

In [29], the authors present a way to practically implement BDI-style agents on top of IndiGolog. The IndiGolog BDI agent program δ_P is obtained by inductively transforming the original BDI plan-body P program. This transformation makes use of two procedures named $achieve_e(\mathbf{x})$ and $handle(a)$. Those procedures rely heavily on non-determinism to choose an option in the plan library for a given event, and to call the appropriate procedure to resolve an event. Our approach is fully compatible with this mechanism. Surely, the original BDI agent could be parameterized by introducing new plans and rules to handle preferences in an ad-hoc way. But what we offer here is far cleaner and more powerful than that. We can apply our transformations to the overall obtained δ_P program, and especially to the $achieve_e(\mathbf{x})$ and $handle(a)$ procedures. This way, we can stick to the agent standard behavior while allowing variations around this behavior.

In [30], the authors extend IndiGolog to handle prioritized goals. They emphasize the need for the agent language to handle both declarative and procedural goals, because some goals should be declaratively stated as preferred to some others. This has been investigated further in [31] and [32]. This approach has some common points with ours: it introduces a way to specify that some goal should be preferred to another that is close to our "$\rangle$" operator. However, it focuses on explicitly declared goals while we focus on behaviors and tendencies based on arbitrary profiles. Furthermore, its goals influence the global execution of a program while our transformations change local choices.

B. Adding psychological features to BDI agents

Since works of Rousseau and Hayes-Roth [33], extensive research has been undertaken, especially recently, for implementing cognitive and/or psychological features in artificial agents. While [34] have implemented a psychological model, dedicated to emotions, based on traditional SOAR architecture, most authors have proposed improvements of BDI architectures. For example, using the BDI platform JACK, CoJACK [35] provides additional layers which intend to simulate *physiological* human capacities like the duration taken for cognition, working memory limitations (*e.g.* "losing a belief" if the activation is low or "forgetting the next step" of a procedure), fuzzy retrieval of beliefs, limited attention or the use of moderators to alter cognition.

However our approach is distinct from most studies using BDI because we focus on psychological traits rather than cognitive capacities. Moreover psychological parameters influence actions and plans rather than goals. These two remarks also apply to works close to our study [36] or [37], based on the architecture of conversational agent GRETA [38].

VI. Conclusion

We have introduced a first definition of a parametric agent and of an associated process to transform a non-parametric agent into a parametric agent. To do so, we exploit the non-determinism of IndiGolog programs. We also ensure that the parametric behavior is based on the original behavior. We have shown that this transformation is sound.

To the best of our knowledge, our approach is the first to propose an alteration process to transform a plain rational

agent into a parametric agent. This transformation has immediate applications in the domains of personalized agents and personified agents.

Our approach differs from other approaches of personalization where focus is always put on goals. Here we propose to parameterize behavior, which is different: our parametric agent is not goal driven, it is behavior driven.

Still, future work is needed in several directions. First, we want to investigate how to best characterize the different paths an agent can follow. Secondly, we want to study if it is sound to open up for more paths and under which conditions. Lastly, we want to reduce the amount of information the application designer has to provide, in particular by studying if a formal definition can be given for action families.

REFERENCES

[1] M. Wooldridge, "Agent-based computing," *Interoperable Communication Networks*, vol. 1, pp. 71–97, 1997.

[2] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277 – 296, 2000.

[3] M. Wooldridge, *Reasoning about Rational Agents*. MIT Press, 2000.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[5] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl, "Golog: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 31, no. 1, pp. 59–83, 1997.

[6] M. Baldoni, C. Baroglio, and V. Patti, "Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions," *Artificial Intelligence Review*, vol. 22, no. 1, pp. 3–39, 2004.

[7] S. Thomas, "The PLACA agent programming language," in *Intelligent Agents*, ser. Lecture Notes in Computer Science, M. Wooldridge and N. Jennings, Eds. Springer Berlin Heidelberg, 1995, vol. 890, pp. 355–370.

[8] M. Xuetao, F. Bouchet, and J. P. Sansonnet, "Impact of agent's answers variability on its believability and human-likeness and consequent chatbot improvements," in *Proc. of AISB 2009*, Edinburgh, 2009, pp. 31–36.

[9] J. Bates, "The role of emotion in believable agents," *Commun. ACM*, vol. 37, no. 7, pp. 122–125, 1994.

[10] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[11] E. Motta, *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1999.

[12] D. Kuokka and B. Livezey, "A collaborative parametric design agent," in *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, ser. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 387–393.

[13] G. Adomavicius and A. Tuzhilin, "Personalization technologies: a process-oriented perspective," *Commun. ACM*, vol. 48, no. 10, pp. 83–90, oct 2005.

[14] D. Rousseau, "Personality in computer characters," in *AAAI Technical Report WS-96-03*. AAAI Press, 1996, pp. 38–43.

[15] C. Castelfranchi, F. D. Rosis, R. Falcone, and S. Pizzutilo, "Personality traits and social attitudes in multiagent cooperation," *Applied Artificial Intelligence*, vol. 12, no. 7-8, pp. 649–675, 1998.

[16] J. McCarthy and P. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University, 1968.

[17] G. Giacomo, Y. Lespérance, H. Levesque, and S. Sardina, "IndiGolog: A high-level programming language for embedded reasoning agents," *Multi-Agent Programming*, pp. 31–72, 2009.

[18] C. Jacquiot, Y. Bourda, F. Popineau, A. Delteil, and C. Reynaud, "GLAM: A generic layered adaptation model for adaptive hypermedia systems," in *Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2006, pp. 131–140.

[19] G. Dubus, F. Popineau, and Y. Bourda, "Situation calculus and personalized web systems," in *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*. IEEE, 2011, pp. 569–574.

[20] P. Brusilovsky, "Methods and techniques of adaptive hypermedia," *User modeling and user-adapted interaction*, vol. 6, no. 2, pp. 87–129, 1996.

[21] G. Dubus, F. Popineau, and Y. Bourda, "A formal approach to personalization," in *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*. IEEE, 2011, pp. 233–238.

[22] B. Hayes-Roth, "What makes characters seem life-like?" in *Life-like Characters. Tools, Affective Functions and Applications*. Heidelberg: Springer, 2004.

[23] F. Bouchet and J. Sansonnet, "Influence of personality traits on the rational process of cognitive agents," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, vol. 2. IEEE, 2011, pp. 81–88.

[24] O. P. John, R. W. Robins, and L. A. Pervin, Eds., *Handbook of Personality: Theory and Research*, 3rd ed. The Guilford Press, 2008.

[25] L. R. Goldberg, "An alternative description of personality: The Big-Five factor structure," *Journal of Personality and Social Psychology*, vol. 59, pp. 1216–1229, 1990.

[26] P. T. Costa and R. R. McCrae, *The NEO PI-R professional manual*. Odessa, FL: Psychological Assessment Resources, 1992.

[27] E. Sisbot, L. Marin, and R. Alami, "Spatial reasoning for human robot interaction," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2281–2287.

[28] R. Gockley, R. Simmons, and J. Forlizzi, "Modeling affect in socially interactive robots," in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006, pp. 558–563.

[29] S. Sardina and Y. Lespérance, "Golog speaks the BDI language," in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, L. Braubach, J.-P. Briot, and J. Thangarajah, Eds. Springer Berlin / Heidelberg, 2010, vol. 5919, pp. 82–99, 10.1007/978-3-642-14843-9_6.

[30] S. Sardiña and S. Shapiro, "Rational action in agent programs with prioritized goals," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, ser. AAMAS '03. New York, NY, USA: ACM, 2003, pp. 417–424.

[31] C. Fritz and S. McIlraith, "Decision-theoretic Golog with qualitative preferences," in *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, Lake District, UK, June 2006, pp. 153–163.

[32] S. M. Khan and Y. Lespérance, "A logical framework for prioritized goal change," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, ser. AAMAS '10. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 283–290.

[33] D. Rousseau and B. Hayes-Roth, "Personality in synthetic characters," in *Technical report, KSL 96-21*. Knowledge Systems Laboratory, Stanford University, 1996.

[34] J. Gratch and S. Marsella, "A domain-independent framework for modeling emotion," *Journal of Cognitive Systems Research*, vol. 5, no. 4, pp. 269–306, 2004.

[35] R. Evertsz, F. E. Ritter, P. Busetta, and M. Pedrotti, "Realistic behaviour variation in a BDI-based cognitive architecture," in *Proc. of SimTecT'08*, Melbourne, Australia, 2008.

[36] L. Malatesta, G. Caridakis, A. Raouzaiou, and K. Karpouzis, "Agent personality traits in virtual environments based on appraisal theory predictions," in *Artificial and Ambient Intelligence, Language, Speech and Gesture for Expressive Characters,achie AISB'07*, Newcastle, UK, 2007.

[37] M. McRorie, I. Sneddon, E. de Sevin, E. Bevacqua, and C. Pelachaud, "A model of personality and emotional traits," in *Intelligent Virtual Agents (IVA 2009)*, ser. LNAI, vol. 5773. Amsterdam, NL: Springer-Verlag, 2009, pp. 27–33.

[38] C. Pelachaud, "Some considerations about embodied agents," in *Int. Conf. on Autonomous Agents*, Barcelona, 2000.

Personalized Recommendation of Open Educational Resources in MOOCs

Hiba Hajri(✉), Yolaine Bourda, and Fabrice Popineau

LRI, CentraleSupelec, Bat 650 (PCRI), Rue Noetzlin, Gif-sur-Yvette 91405, France
{hiba.hajri,yolaine.bourda,fabrice.popineau}@centralesupelec.fr

Abstract. Today Online Learning Environments (OLE) like MOOCs and LMS are very commonly used and a huge number of students with very different profiles and backgrounds follow the same online courses. Still, personalized experience for attendees is not widely spread on the platforms hosting these courses. At the same time, there is a growing number of open educational resources (OER) that can helpfully enrich the content of online courses and even be chosen to match one-by-one the student tastes. Recommender systems may support personalization in OLE by providing each learner with learning objects carefully selected to help reaching their learning objectives. This kind of recommendation is more specific to compute than usual recommendations like consumer products: the recommendation depends not only on the learner profile, but also on the content of the course, because the recommendation needs to fit precisely with the course format at any point. In this article, we introduce MOORS, a MOOC-based OER recommender system that can be plugged in an OLE to dynamically provide recommendations of OER to learners on the basis of their profiles and the profile of the MOOC. We also describe the process for calculating recommendations from OER metadata, assuming these metadata follow the Linked Opend Data (LOD) principles. Our implementation has been done in Open edX, an open source MOOC platform widely used, however the same approach could be implemented in any OLE as long as the learners profiles and the course profile can be extracted. Finally, we discuss a life-size evaluation of our recommender system.

1 Introduction

For many years now, personalization in technology enhanced learning (TEL) is a major subject of intensive research. But, with the spreading of Massive Open Online Courses (MOOCs) and their open and massive nature, the personalization issue is becoming ever more important. Indeed, a learner who follows a MOOC, freely shared on the web and without any commitment related to a registration fees he paid or a diploma he must obtain, may decide not to finish it at any time, when the MOOC no longer meets his needs. To support personalization in MOOCs, recommender systems are increasingly introduced to enable learners to finish MOOCs and take the better advantage from its content. Recommendations

B. M. McLaren et al. (Eds.): CSEDU 2018, CCIS 1022, pp. 166–190, 2019.
https://doi.org/10.1007/978-3-030-21151-6_9

offered to each learner are adapted to his needs and his learning objectives. They can take the form of pedagogical resources, other learners, discussions, etc.

At the same time, the amount of Open Educational Resources (OER) available on the web is permanently growing. These OERs have to be reused in contexts different from the initial ones for which they were created. Indeed, producing quality OER is costly and requires a lot of time. Reusing OERs can provide an efficient way to enrich online courses content and particularly MOOCs content. But the reuse of OERs depends on the ease with which we can identify them on the web and on the quality and the availability of their descriptions (metadata). In this context, linked Open Data principles are sometimes used for describing OERs in order to facilitate their discovery automatically by machines and then their reuse.

In this paper, we introduce a MOOC-based OER recommender system (MORS). MORS assists learners, who are attending a MOOC, by offering to them complementary OERs, in addition to the internal resources of the MOOC, when these resources are not sufficient for them. MORS recommendations aim to remedy some of learners deficiencies during the MOOC, in order to help them to assimilate the MOOC's knowledge. Recommendations provided by MORS are computed dynamically based on learner profile and on the MOOC profile. MORS is implemented in MOOCs but it can be integrated in other OLE as the course and the learner profiles can be extracted. We choose MOOCs because (1) their large number of learners with varied profiles make them good candidates for personalization, and (2) because targetting an open platform like Open edX will allow us to widely disseminate our system and to make it reusable.

This paper is an extended version of our previous work [8]. We extend our previous work by presenting the experiments conducted to evaluate our solution and discussing the results. This paper begins with related work. In Sect. 3 we introduce the recommendation scenarios proposed by our solution. Section 4 draws the architecture of the proposed system. Section 5 presents how we implemented our solution. The evaluation protocol and results are proposed in Sect. 6. Section 7 concludes the paper and presents future directions.

2 Related Works

Even though Personalization in TEL is a research topic with a long history, studies on MOOCs personalization have started since 2012 [17]. In order to address issues related to the open and the massive nature of MOOCs, different personalization approaches have been introduced. One of the most popular techniques relies on recommender systems. We have studied some of the proposed approaches to personalize MOOCs and adapt their content to the needs and objectives of each learner, based on different criteria.

The studied approaches offer different types of recommendations as pedagogical resources and other learners who can help the learner during the MOOC. Regarding the recommendation of pedagogical resources, we noticed that most approaches compute their recommendations based on internal resources of the

MOOC. For example, [1] targets learners who post a question in a MOOC discussion which reflects a confusion and recommends educational videos related to the confusion subject. [3] recommends additional learning activities to learners who show a lack of knowledge in a particular subject. However, it is interesting to offer to the learner external resources from the web, when the resources of the MOOC fail to meet his needs or do not allow him to assimilate the lacking knowledge.

For approaches that recommend external resources to the learner, we found that they rely on a static set of resources selected from the web. For example, [2] recommends to the learner a set of MOOCs which mostly match his learning objectives. Another approach [14] offers a scenario of activities to each group of learners according to the gap between their actual competencies and the target ones. This scenario can perform a number of recommendations of either internal or external educational resources, captured from the web. It is therefore interesting to be able to select external resources dynamically to take into account the changes on the web.

On the other hand, the approaches offering recommendations of external pedagogical resources to a learner who is following a MOOC are based on some of his characteristics: his knowledge, his preferences, his learning objectives, etc. They are therefore centered on the learner. But, these new resources will complement a MOOC, which represents an initial and main learning path and will enrich its content, for example to remedy the learner gaps or to present a set of alternative resources to those proposed in the MOOC. It is therefore important to propose resources that are both adapted to the characteristics of the learner and to the specificities of the MOOC that they will complete. It is also necessary to take into account the stage of the MOOC in which the recommendations are proposed. The interest of the dynamic calculation of the recommendations compared to the static calculation is that it allows taking into account, on the one hand the evolution of the profile of the learner throughout the MOOC and, on the other hand, the updates and the changes made on external pedagogical resources and repositories storing these resources descriptions.

We have also noticed that most of the approaches have been implemented in specific platforms, such as a specific university or laboratory, and are dedicated to a specific area. For example, the approach proposed by [13] is specific to health MOOCs in the area of Motivational Interviewing. It recommends to the learner the MOOC resources related to concepts they only need to know, by analyzing learners contexts. That is why it is important to introduce a generic solution, independent of the MOOC domain and implemented in one of the MOOC platforms which are accessible to everyone, such as Open edX.

In our work, we propose a generic solution providing recommendations of OERs in a MOOC platform when a lack of knowledge is detected for a learner. These recommendations are computed dynamically based on different learner characteristics and also on MOOC specificities.

3 Recommendation Scenarios

In this section, we describe how our system personalizes a MOOC for a learner. More precisely, we introduce some realistic scenarios of recommendation offered by MORS: where and when exactly the recommendation process is triggered for a learner during the MOOC.

Let's consider the MOOC as a set of sections. Each section offers pedagogical resources as video, text, quiz, etc. We consider also that studying the MOOC requires some prerequisites with a certain performance level defined by the MOOC creator who is the teacher. Each MOOC section provides some learning objectives with a certain performance level defined by the teacher. In our solution, we decide to recommend OERs to the learner at two different kind of stages of the MOOC: before starting the MOOC and after each MOOC's section.

Before Starting the MOOC. Once a learner is enrolled in a MOOC and decides to start its first section, MORS verifies if the learner has the prerequisites of the MOOC with the appropriate performance degrees. If a lack of knowledge is detected in at least one of the MOOC prerequisites, the recommendation process is triggered and a set of OERs dealing with the appropriate prerequisites are recommended to the learner.

At the End of Each MOOC Section. As stated earlier, each MOOC section has at least one learning objective. This learning objective can also be a prerequisite for the following section. So it is important to ensure that the learner has assimilated the section content. That is why at the end of each section, a quiz is presented to the learner where each question aims to assess his assimilation level of at least one of the section learning objective. Now if the learner gets bad results in the quiz, MORS triggers the recommendation process in order to recommend to him a set of OERs dealing with the learning objectives where he failed.

4 System Architecture

In this section we describe the architecture of our system for recommending OERs in a MOOC (MORS). MORS is composed of four major process (Fig. 1): process of generating the MOOC profile, process of generating the learner profile, PreSelection process and refinement process.

Once MORS is integrated in the MOOC to be personalized, the process of generating the MOOC profile generates the profile of the MOOC based on the MOOC model defined in our solution. Then the process of generating the learner profile generates the learner profile based on the learner model defined in our solution. When the system identifies gaps in student mastery of a certain topic, before starting the MOOC or at the end of one of its sections, the PreSelection process requests the external OERs descriptions repositories to collect an initial set of OERs dealing with this topic. This initial set is transmitted to the refinement process that performs selection and ranking operations based on the

learner and the MOOC profiles content at the time of the calculation of the recommendations.

4.1 Process of Generating the MOOC Profile

The role of this process is to generate the profile of a specific MOOC from the MOOC model defined in our approach.

MOOC Model. In the MOOC Model, we represent some of its characteristics: the knowledge elements of the MOOC and the corresponding performance degrees, the domain of the MOOC and the effort needed in terms of time. We consider two types of knowledge elements: learning objectives of each MOOC section and prerequisites of the MOOC.

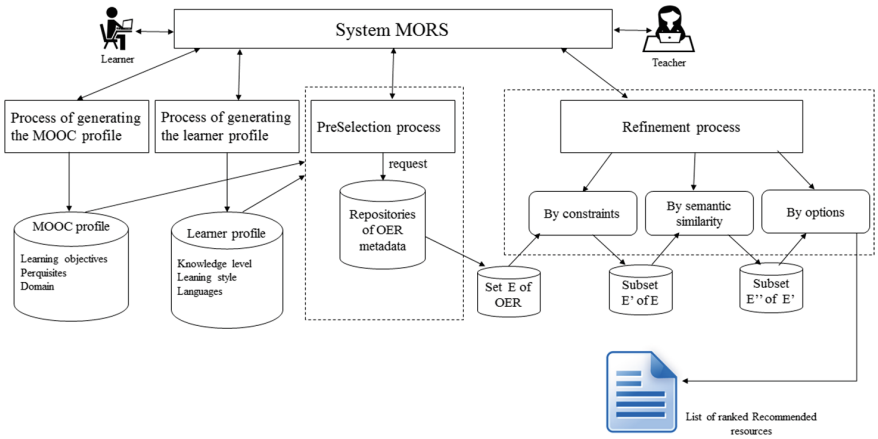


Fig. 1. The architecture of MORS.

Notations. In this paper, we denote the number of MOOC sections by $n_{section}$, the set of MOOC knowledge elements as KE, the set of MOOC prerequisites as P, the set of the learning objectives provided by the kth section as LO_k, where $1 \leq k \leq n_{section}$ and the set of the learning objectives provided by the entire MOOC as LO. Then $LO = \cup_{k=1}^{n_{section}} LO_k$ and $KE = LO \cup P$.

The teacher defines the MOOC knowledge elements together with their performance degrees. In this work, we use the performance degrees as introduced in [9] (1: beginner, 2: intermediate and 3: expert) to which we add (0: no performance).

Definition 1 (Performance Degree by Teacher): *Given a knowledge element ke from KE, the performance degree of ke, set by the teacher, is defined by the function LP_T.*

$$LP_T : \begin{cases} KE \longrightarrow \{0,1,2,3\} \\ ke \longmapsto lp \in \{0,1,2,3\} \end{cases}$$

The prerequisites and the learning objectives of the MOOC associated to their performance degrees are modeled respectively by the vectors $V_{P,MOOC}$ and $V_{LO,MOOC}$. We represent in (Fig. 2) the evolution of knowledge elements during the MOOC. As input, we represent in $V_{P,MOOC}$ the required performance degrees for each prerequisite. Then we represent the evolution of $V_{LO,MOOC}$ during the MOOC. At first, $V_{LO,MOOC}$ is initialized to zero. Thus, at the end of each section, $V_{LO,MOOC}$ is updated with new values. These values correspond to performance degrees expected to be acquired in learning objectives provided by the section.

The MOOC domain and the MOOC effort are represented respectively by the variables D_{MOOC} and Eff_{MOOC}.

Generation of the MOOC Profile. The data required to generate the profile of a specific MOOC based on our MOOC model are collected from the information entered by the teacher when creating this MOOC. The teacher defines the prerequisites of the MOOC, the learning objectives of each of its sections and the corresponding performance degrees. He defines also the domain and the effort of the MOOC.

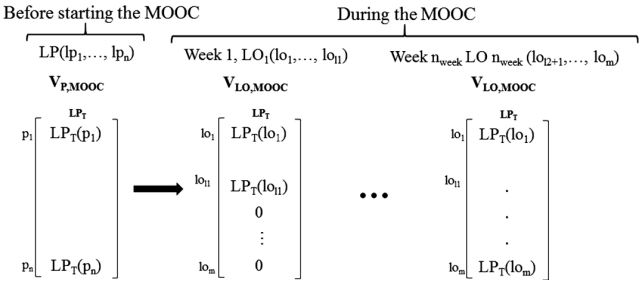


Fig. 2. The evolution of the knowledge elements during the course (from [8]).

4.2 Process of Generating the Learner Profile

The role of this process is to generate and update the profile of a specific learner from the learner model defined in our approach.

Learner Model. In the learner Model, we represent some of his characteristics: his knowledge level, his learning style and the languages he knows. Concerning the learner knowledge level, we consider only his level in the knowledge elements of the MOOC. More precisely, the learner model contains the performances degrees of the learner in the prerequisites and the learning objectives of the MOOC.

Definition 2 (Performance Degree of Learner): *Given a knowledge element ke from KE, the learner performance degree in ke is defined by the function* LP_L.

$$LP_L : \begin{cases} KE \longrightarrow \{0,1,2,3\} \\ ke \longmapsto lp \in \{0,1,2,3\} \end{cases}$$

The prerequisites and the learning objectives of the MOOC associated to the learner performance degrees are modeled respectively by the vectors $V_{P,Learner}$ and $V_{LO,Learner}$. A quite similar modeling process is proposed in [15]. In (Fig. 3) we represent the evolution of the knowledge elements in the learner profile during the MOOC. Before starting the MOOC, the learner performance degrees in MOOC prerequisites are stored in $V_{P,Learner}$. During the MOOC and after each section, $V_{LO,Learner}$ is updated with the new learner performance degrees acquired with the learning objectives provided in this section.

The learning style refers to the way a learner receives and processes information [6]. In the literature, many profiles are defined to analyze learners learning styles like Kolb [10] and Felder and Silverman [6]. In our work, we use the frequently used, Index of Learning Style (ILS) questionnaire [16]. It was developed by Felder and Soloman to identify learning styles based on Felder and Silverman Learning style Model (FSLSM) [6]. The FSLSM classifies learning styles along to four dimensions which are active/reflective, sensorial/intuitive, visual/verbal and sequential/global. In our work we also use the patterns introduced by [5] to identify the type of learning resources to be provided to the learner based on his answers to the ILS questionnaire. For example, sensing learners prefer to get more examples and exercises [5]. We model the learning style by using the term $S_{learner}$.

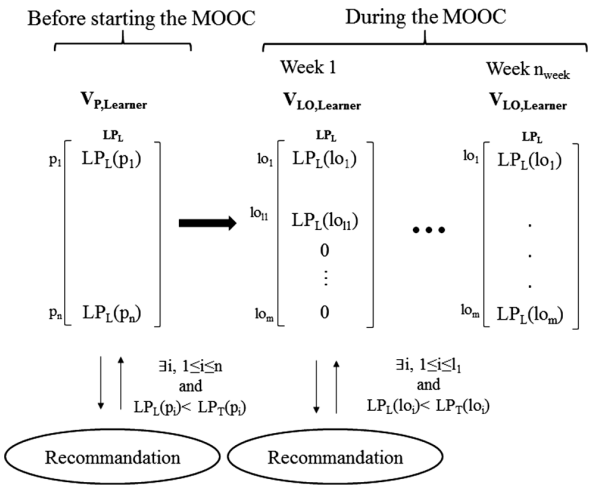


Fig. 3. The evolution of the knowledge elements in the profile of the learner during the course (from [8]).

The languages known by the learner are modeled by using a list $L_{learner}$.

Generation of the Learner Profile. The data required to generate the profile of a specific learner are collected with different methods. The learner performance degrees on MOOC prerequisites are determined by browsing his knowledge base storing the knowledge elements acquired by the learner previously on the same MOOC platform. If no significant evidence is collected this way, because the learner is a new user of the platform for example, then we ask the learner a few questions about the prerequisites in order to evaluate his performance degrees.

Each MOOC section ends with a quiz. The learner results on those quiz are used to compute the learner performance degrees on MOOC learning objectives. For the learning style of the learner, we use his answers to the questionnaire ILS [16] to deduce his learning style. We ask him also about the languages he knows.

4.3 PreSelection PROCESS

As indicated in (Fig. 3), the recommendation process is triggered for a learner L at two different kind of steps of the MOOC.

Before Starting the MOOC. Let $LP_L(p)$ the performance degree of a learner L in $p \in P$, the recommendation process is triggered if:

$$LP_L(p) = V_{p,L}(p) < LP_T(p) = V_{p,MOOC}(p)$$

During the MOOC. At the end of the kth section of the MOOC, let a learner L who acquires $LP_L(lo)$ in $lo \in LO_k$, the recommendation process is triggered when:

$$LP_L(lo) = V_{LO,L}(lo) < LP_T(lo) = V_{LO,MOOC}(lo)$$

Where $V_{LO,MOOC}$ and $V_{LO,L}$ considered are those updated after the kth section.

The aim of this module is to select a set of candidate OERs dealing with the knowledge element for which the recommendation process has been triggered. In order to find these resources, the system performs a keyword search in metadata stored in external accessible repositories of OERs metadata [7]. The metadata used in this search is "the description of the resource" introducing the subject and the global idea of the resource.

For keywords, the first keyword is the knowledge element which is the subject of the recommendation. The objective of the search is, therefore, to select resources with descriptions containing this knowledge element. However, we must also solve the problems of synonymy and polysemy that may arise in our search.

Polysemy. The same knowledge element can be expressed differently. For example "conditional statements" in computer science are also "if-then-else".

Synonymy. The same expression can be used to represent different knowledge elements that belong to different domains. For example the notion of "graph" is used in a variety of disciplines as "computer science", "mathematics", etc.

To solve synonymy problems, we used a second keyword, in addition to the knowledge element, which is the domain of the MOOC. We introduced also a module of synonyms detection[1] to address synonymy problems. This module is based on DBpedia[2] structured data that has been extracted from Wikipedia. Some possible synonyms of the knowledge element and the MOOC domain are inferred using this module that will increase the number of selected resources.

Overall, we select descriptions including the knowledge element and the MOOC domain or at least one of their synonyms.

In a formal way, let OER be the set of Open educational resources, $R \in OER$, $ke \in KE$, $Meta_R$ is the set of the metadata of R where $\text{Meta}_R = \{\text{Descript}_R, \text{Lang}_R, \text{Prereq}_R, \ldots\}$, $Syn_{D_{mooc}}$ the set of the synonyms of the domain of the MOOC, D_{mooc} and Syn_{ke} is the set of the synonyms of the ke generated by our module of synonyms detection.

$$(R \text{ is dealing with } ke) \equiv ((ke \in \text{Descript}_R) \text{ OR } (\exists i : Syn_{ke}[i] \in \text{Descript}_R)) \text{ AND}$$
$$((D_{mooc} \in \text{Descript}_R) \text{ OR } (\exists j : Syn_{D_{mooc}}[j] \in \text{Descript}_R)).$$

To select the candidate resources, our PreSelection process requests external repositories providing SPARQL endpoints with an initial query. In order to manage the diversity of metadata schemas employed by these repositories, we use classes and properties as defined in the Learning Object Ontology of Mapping (LOOM) introduced in [7].

4.4 Refinement Process

The PreSelection process returns as result an initial set E of OER and as we explained in the previous part the PreSelection is based on keywords. This initial set E is transmitted to this refinement process which applies selection and sorting operations according to several criteria. These operations are performed in three stages.

- Selection by constraints.
- Selection by semantic similarity.
- Sorting by options.

Selection by Constraints. In this first step, we select a subset E' of E which contains only the resources that respect certain constraints. The constraints are firm criteria that must be respected by the resources recommended to the learner. In other words, these are criteria without which it will be difficult for the learner to understand the content of the resource or it will impact the MOOC follow up. The three constraints we consider are the following. The first one is: "Resources must not require prerequisites not assimilated by the learner" **(C1)**. The second constraint is: "Resources must be presented in a language known by the learner" **(C2)**. The third constraint is: "The resource has to bring a performance level which is greater than or equal to the level defined in the MOOC" **(C3)**.

[1] https://davidallenfox.wordpress.com/2013/09/05/generating-synonyms/.
[2] http://wiki.dbpedia.org/.

Constraint C1 Violation. Let $R \in E$, R doesn't respect the constraint C1 if:

$$\exists ke \in \text{Prereq}_R : (LP_L(ke) = 0) \vee (\exists i, LP_L(\text{Syn}_{ke}[i]) = 0)$$

In order to identify the performance degree $LP_L(ke)$ of the learner in the prerequisite ke of the resource R, we start with browsing his knowledge base looking for the information. Otherwise, we ask the learner some questions to deduce his $LP_L(ke)$.

Constraint C2 Violation. Let $R \in E$, R doesn't respect the constraint C2 if:

$$(\text{Lang}_R \notin \text{L}_{learner})$$

Where $\text{L}_{learner}$ is the set of the languages known by the learner and Lang_R the metadata presenting the language of the resource.

Constraint C3 Violation. As we did previously for the learner and the MOOC, each resource $R \in E$ is modeled by a vector $V_{KE,R}$ that represents the performance degrees acquired in each ke of the MOOC after following this resource.

Definition 3 (Resource Performance Degree): *The performance degree acquired by the learner with regard to a specific knowledge element ke, from KE, after following the resource R is defined by the function* LP_R.

$$LP_R : \begin{cases} KE \longrightarrow \{0,1,2,3\} \\ ke \longmapsto lp \in \{0,1,2,3\} \end{cases}$$

As shown in (Fig. 3), the recommendation process is triggered when $LP_T(ke) > LP_L(ke)$. Indeed, the goal of the recommendation of the resource R to the learner is the improvement of his level of performance in the knowledge element ke.

So, Let $R \in E$, R doesn't respect the constraint C3:

$$LP_R(ke) < LP_T(ke).$$

As the OERs metadata do not specify the performance degrees of OREs, we need to estimate them from learners who have already used the resources. For a new OER we use two performance degrees 0: a resource doesn't deal with the knowledge element and 1: the resource deals with the knowledge element. Then once the resources are studied by the learners, we plan to collect the results from them according to their responses to the knowledge test once they have followed a recommended resource.

Definition 4 (A Knowledge Element derived from a Resource):

$$(ke \textit{ is provided by } R) \equiv (\text{V}_{KE,R}(ke) \neq 0) \equiv$$
$$\{ke \in \text{Descript}(R) \vee \exists i : \text{Syn}_{ke}[i] \in \text{Descript}(R)\}.$$

Selection by Semantic Similarity. Once resources respecting the defined constraints have been identified, we select resources which are close to the initial query (the query defined in the PreSearch module). For that purpose, we calculate the similarity between selected resources and the initial query terms (IQT). The IQT are the terms used in the initial query: the knowledge element, the domain of the MOOC and their synonyms generated by our module of synonyms detection. We denote IQT as a set.

$$IQT = \{\mathrm{T}_1, ..., \mathrm{T}_{nt}\}$$

Where T_{nt} is one of the initial query terms and nt is the total number of terms used in the initial query.

We start with using Term Frequency Inverse Document Frequency (TF-IDF) [4] to identify the importance of IQT inside the selected resources descriptions. Each selected resource R is represented by a vector V_R.

$$\mathrm{V}_R = (\mathrm{V}_{R_{T_1}}, ..., \mathrm{V}_{R_{T_{nt}}})$$

Where $\mathrm{V}_{R_{T_1}}$ is the TF-IDF value of the term $operatornameT_1$ in the description of the resource R.

The IQT is also represented by a vector V_{IQT}.

$$\mathrm{V}_{IQT} = (\mathrm{V}_{IQT_{T_1}}, ..., \mathrm{V}_{IQT_{T_{nt}}})$$

Where $\mathrm{V}_{IQT_{T_1}}$ is the TF-IDF value of the term $\mathrm{V}_{R_{T_1}}$ in the descriptions of all selected resources.

Then a cosine measure is employed to compute the similarity between each resource vector V_R and the initial query vector V_{IQT}. As result each selected resource R is characterized by one measure which is his cosine measure, CosSim(R). The resource R with higher value of CosSim(R), is the closest to the initial query.

At the end of this step, the result is the set E' of resources ordered by their semantic similarity with the IQT. We select the subset E'' of the first n ones to be the input of the next step. We take a limited number of resources because the objective of our recommendations is to help the learner to improve his knowledge level in a certain knowledge element by following at least one resource rather than recommending a large number of resources. The number n is defined arbitrarily and in our case study, n has been fixed to 4 but the teacher can change its value. We define the relation of preference ($\geq$).

Definition 5 (Preference Relation $\geq$): *Given two resources $R_1 \in OER$ and $R_2 \in OER$:*

$$R_1 \geq R_2 \textit{ means } R_1 \textit{ is at least as good as } R_2.$$

In this first step $R_1 \geq R_2 \Leftrightarrow \mathrm{CosSim}(R_2) \geq \mathrm{CosSim}(R_1)$.

Sorting by Options. The last step is to classify resources based on options. The options are other criteria defined to reflect the adaptation to some characteristics of the MOOC and the learner. However, these options are not mandatory criteria as the constraints used in the first step. In other words, recommended resources may not be consistent with all options, but they are presented to the learner in an order depending on how much they satisfy the options. Let Op the set of options and n_{op} the total number of options.

Definition 6 (Score Function): *For each $op_i \in OP$, where $1 \leq i \leq n_{op}$, we define the score function U_i:*

$$U_i : \begin{cases} E'' \longrightarrow [0,1] \\ R_j \longmapsto a_i^j \end{cases} \tag{1}$$

U_i assigns a score a_i^j, between 0 and 1, to each candidate resource R_j depending on how much it satisfies the op_i. The scores a_i^j are calculated differently depending on the options op_i type.

For each option, the resource score represents its option satisfaction percent. Then we consider each option as a fuzzy set and the score of each resource as its membership degree to this set. We represent each resource R_j by a vector S_{R_j} whose components are the values of its score for each option.

$$S_{R_j} = (a_1^j, a_2^j, ..., a_2^{n_{op}})$$

The ideal resource id has a vector S_{id} whose components are equal to 1. This means that the resource meets all the options at 100%.

A weight value $p_i \in 1, 2, 3$ is assigned to each option in order to characterize its importance (1: less important, 2: important and 3: very important).

Initially all options have the same importance ($p_i = 3$) but we give the teacher the possibility to change weights values of options defined to reflect coherence with the MOOC. In MORS, we defined these options: "Recommended resources should respect the learning style of the learner." (Op_1) and "Recommended resources should have a 'typical Learning Time' similar or bellow the mean effort needed to assimilate a MOOC resource, as defined by the teacher." (Op_2).

For Op_1, we define the corresponding score function U_1 as below:

$$U_1(R) = \begin{cases} 1 \text{ if } \mathrm{Typ}_R \in RT_L \\ 0 \text{ else.} \end{cases}$$

where Typ_R is the type of the resource $R \in E''$ and RT_L is the set of resources types corresponding to the learner L learning style.

Concerning op_2, we define the corresponding score function U_2 as below:

$$U_2(R) = \begin{cases} 1 & \text{if } LT_R \leq ME_{W/R} \\ (\varepsilon + ME_{W/R} - LT_R)/\varepsilon & \text{elseif } LT_R \\ & \in [ME_{W/R}, ME_{W/R} + \varepsilon] \\ 0 & \text{else } LT_R \geq ME_{W/R} + \varepsilon \end{cases}$$

where $R \in E''$, $ME_{W/R}$ (Mean Effort section) corresponds to the quotient of the section effort defined by the teacher and the number of the section resources and LT_R corresponds to the value of the metadata 'typical Learning Time' for R. The value ε is defined arbitrarily and has been fixed to ME_W in our case study.

To rank candidate resources, we use the Chebyshev distance to compute the distance between the ideal resource and each resource to recommend. The smaller the distance is the better the resource is. This distance is defined as below:

$$\mathrm{DCH}_{R_j,id} = max_{i \in n_{op}} \lambda_i |V_{R_j}[i] - V_{id}[i]|$$

where λ_i is defined as below:

$$\lambda_i = p_i / (\mathrm{Sup}_{R_j \in E''}(V_{R_j}[i]) - \mathrm{Inf}_{R_j \in E^*}(V_{R_j}[i]))$$

where E^* is a subset from E'' of candidate resources that not have a maximal satisfaction degree for any option.

In conclusion,

$$R_1 \geq R_2 \Leftrightarrow \mathrm{DCH}_{R_2,id} \geq \mathrm{DCH}_{R_1,id}$$

5 Implementation

In order to implement our solution, we choose edX as the MOOC platform. First, it is an open platform which is widely used. Then, this choice allows us to offer our solution to the vast community of OpenEdX users and hope to replicate experiments about personalization, then gather more data about its efficiency. Furthermore, the documentation of Open edX is well detailed and the community is very active. Finally, the platform is characterized by a modular architecture thanks to XBlocks [11] that we will detail later.

The XBlock is a component architecture developed in 2013 by edX, which allows developers to create independent course components (xBlocks). These components can be combined together to make an online course [11]. The advantage of XBlocks is that they are deployable. The code that you write can be deployed in any instance of the edX Platform or other XBlock runtime application[3]. We found also that there is a recent focus on using these XBlocks to add personalization in MOOCs, for example the work [12] where a recommender XBblock was created in order to recommend resources for remediation in a MOOC. Once developed, each XBlock can be installed and added by the MOOC creator, in the appropriate unit of the appropriate section of his MOOC[4]. In fact, Open edX organized the courses in a hierarchy of sections, sub-sections and units, where the unit is the smallest component in the MOOC.

[3] https://open.edx.org/about-open-edx.

[4] http://edx.readthedocs.org/projects/open-edx-building-and-running-a-course/en/latest/.

For these reasons, we use XBlocks to implement our solution in edX. Three XBlocks have been implemented.

An XBlock to Generate the MOOC Profile and the Static Part of the Learner Profile. This XBlock is developed to be added at the first unit of the MOOC first section. It is responsible for collecting information about the learner and the MOOC: learner languages, learner learning style, MOOC knowledge elements and corresponding performance degrees (see Fig. 4).

An XBlock to Compute Recommendation at the Beginning of the MOOC. This XBlock checks the performance degree of the learner in the prerequisites of the MOOC. In order to ensure this, it starts with determining whether this prerequisite is stored in the knowledge base of the learner. In case the prerequisite is not found in the knowledge base, the XBlock asses the knowledge level of the learner in the MOOC prerequisites by asking him some questions (an example for the prerequisite "structure data" (Fig. 5)). Then if he doesn't answer the questions correctly, a set of OERs links are recommended to him. These links are ranked by descending order by satisfaction of the criteria defined at the previous section (Fig. 6).

An XBlock to Compute Recommendation after Each MOOC Section. A third XBlock is developed to be added at the end of each section. This XBlock computes recommendations of OERs to the learner based on his answers to the quiz presented at the end of the section. These OERs links are presented to the learner sorted based on the criteria we defined in the previous section.

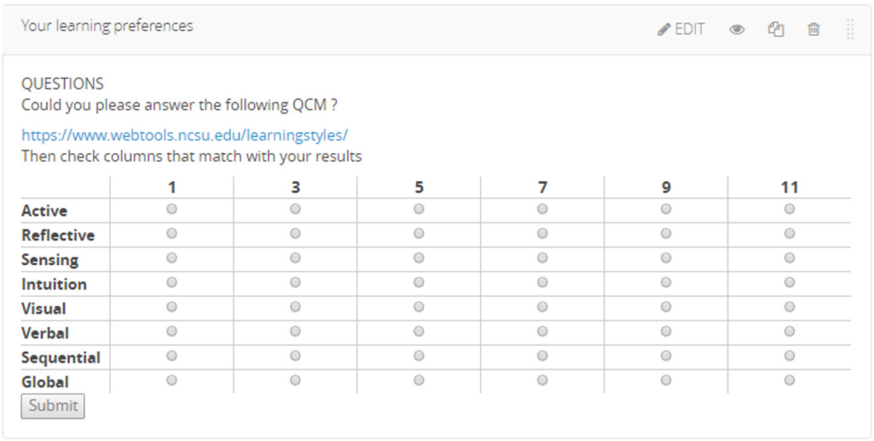


Fig. 4. Interface to collect information about the learning preferences of the learner (from [8]).

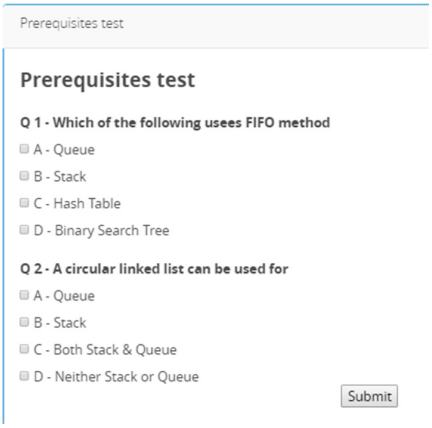


Fig. 5. Interface for testing the assimilation of the prerequisites (example "Data structure") (from [8]).

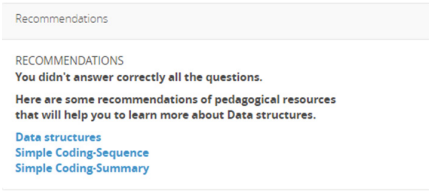


Fig. 6. Interface for recommended resources (from [8]).

6 Evaluation

Our recommender system MORS presents some specific characteristics that come firstly from the open and the massive nature of MOOCs and secondly from OERs referenced in external and dynamic repositories. For this reason, it is difficult to reuse evaluation protocols mentioned in the literature for classic recommender systems and it is important to introduce a new evaluation protocol.

This section presents a set of experiments conducted to evaluate our solution and discuss the results. The purpose of the evaluation is to assess our algorithm of recommendation and the relevance of the recommended resources. The recommended resources have to be adequate with the criteria defined previously, to exhibit some characteristics consistent with both the learner and the MOOC. The scalability and the versatility of OERs repositories means that OERs and their descriptions change dynamically, so there is no "best OERs", only best OERs at some time point. Given OERs vary over time, during the evaluation process, we are not interested in assessing whether our system recommends all the "good" resources, but rather in assessing whether the retrieved resources are "good" resources.

6.1 Evaluation Process

In order to evaluate our system, we chose the MOOC[5] "Design a Database with UML" in the domain of computer science, from the platform OpenClassrooms. This MOOC is composed of three sections. Each section represents a set of pedagogical resources. An assessment quiz is presented at the end of the section to

[5] https://openclassrooms.com/courses/faites-une-base-de-donnees-avec-uml.

evaluate the assimilation of its learning objectives by the learner. In our evaluation, we were interested in a learning objective of each section: "Relationnel model" presented in the first section, "Database management system (DMS)" presented in the second section and "SQL" presented in the third section.

Our evaluation process is based on two questionnaires. The first one is dedicated to experts and the second one is intended to learners. The questionnaire for experts has been sent by e-mail to four teachers which are experts in the domain of database. We had several constraints to respect for learners. The MOOC can be followed by a large number of learners with various profiles. To collect a large number of various users profiles without having to wait until they subscribe and follow the MOOC from the beginning to the end, we used the website Foule Factory. It is a micro-service platform which offers the possibility to ask the crowd to do some tasks as answering questions or finding data. It also allows recovery of rapid results. 117 Foule Factory respondants answered our questionnaire.

In the questionnaire dedicated to teachers, we present the REL selected by the initial query which are expected to provide the three learning objectives of the MOOC as well as some questions to assess these resources. Concerning the questionnaire for learners, we start with assessing the knowledge level of each learner in each learning objective. Therefore, we invite the learner to answer a set of questions about the learning objective, selected from the assessment quiz of the corresponding section. In this context, we chose three questions to evaluate his knowledge in the notion of "relational model", three questions to evaluate his knowledge in the notion of SMD and two questions to evaluate his knowledge in the notion of "SQL". For each set of questions, if the learner answers correctly, he is redirected to the questions about the learning objective of the next section. Otherwise, we consider that the learner has not sufficiently assimilated the corresponding notion and we present to him the resources selected by the initial query as well as some questions to evaluate these resources.

Adequacy of Recommendations with the MOOC and the Learner Profiles

A set of questions asked to Foule Factory respondents and to the teachers had the objective to assess the adequacy of selected OER with the criteria fixed, with regard to our approach, to express the recommendations adaptation with the learner and the MOOC profiles.

Application of Semantic Similarity Measure. The first criterion to evaluate is the selection of the closest resources to the initial query by applying the semantic similarity measure. In this context, for each learning objective: "relational model", "DMS" and "SQL", we present the initial set selected using the initial query to the teacher. For each resource, a closed question is presented to the teacher asking whether this resource is relevant or not. Relevance in this context refers to the fact that studying the resource allows the learner to acquire knowledge on the notion.

The synthesis of teachers replies will help us to assess the interest applying the semantic similarity measure on REL descriptions is interesting and whether it allows eliminating non relevant resources selected by the initial query and keeping the most relevant resources.

Adequacy of Recommended Resources with the MOOC Profile. In order to assess the adequacy of the OER recommended by our solution with some specificities of the MOOC, the teachers are invited to rate the relevance of the recommended OER in accordance with the criteria fixed in our approach: the granularity, the learning time and the provided performance degree.

For each resource that provides the learning objective, according to the teacher, he is invited to answer three questions:

1. A closed question asking the teacher to rate the resource on a scale of 1 to 5 according to how much its granularity is more or less adequate with the MOOC content.
2. A closed question asking the teacher to rate the resource on a scale of 1 to 5 according to how much its learning time is more or less adequate with MOOC specificities.
3. A closed question asking the teacher to rate the resource on a scale of 1 to 5 according to how much the knowledge level provided by the resource in the notion in question is more or less adequate with the level supposed to be provided in the MOOC.

Adequacy of Recommended Resources with the Learner Profile. The assessment of the adequacy of the resources recommended by our solution with some characteristics of the learner is conducted according to this protocol. As a first step, we ask Foule Factory respondents to rate the pertinence of the OER in accordance with the criteria fixed in our approach: the learning style and the knowledge of the learner. For each resource, they are invited to answer two questions:

1. A closed question asking the respondent to rate the resource on a scale of 1 to 5 according to how much the resource is more or less pleasant to follow and matches with his learning habits.
2. A closed question asking the respondent to rate the resource on a scale of 1 to 5 according to how much the resource is more or less easy to follow.

In a second step, we ask the respondents to choose one resource to follow from the list of OER presented to them for each notion. Our aim is to examine the learner's choice to compare it with his characteristics and then to detect what makes him choose one resource over than other. Learners are invited to answer two questions.

1. A closed question asking them to tick the resource they choose to study.
2. An open question asking them to justify choosing this resource and abandoning the others.

Evolution of learners Knowledge after Recommendations

Another set of questions proposed to Foule Factory respondents aims to evaluate the effect of the recommended resources on the evolution of learner knowledge in general and on the evolution of his knowledge in the learning objective of the MOOC.

For this purpose, learners are invited, firstly to answer a closed question asking him whether the recommended resources allowed him to acquire new knowledge. Secondly, the same questions selected from the evaluation quiz and asked to evaluate his knowledge level in each notion, are addressed to the learner for a second time after he studied at least one recommended resources for each notion. The purpose is to study and compare his answers before and after recommendations.

Overall evaluation of the Approach

A last set of questions asked to learners and teachers concerns assessing globally our solution. As experts, after consulting the recommended resource for each notion, the teachers are invited to express their views on the interest of recommendations, the method we use to propose recommendations and more precisely the importance of the criteria used to adapt the resources to the specificities of the MOOC and to propose other criteria they find interesting.

The teachers questionnaire contains 4 questions dedicated for that purpose.

1. An open question asking the teacher's view on the idea of recommending external resources for a learner who is attending a MOOC.
2. A closed question presenting the different criteria used in our solution to the teacher and asking him to rate them according to their importance on a scale from 1 to 3 (not very important, important, very important).
3. An open question inviting the teachers to propose other criteria to take into account when choosing resources to recommend, in order to respect the specificities of the initial course and to facilitate the integration of the resources in its content.
4. An open question asking teachers to propose other suggestions.

As playing learner role, Foule Factory respondents are also invited to express their views on recommending external resources, after consulting the recommended resources and studying some of them.

The learners questionnaire contains 3 questions dedicated for this purpose.

1. An open question asking the respondents to suppose they are attending an online course and express their views about recommending to them other resources when they don't answer correctly to the evaluation test of a certain course's section. The respondents are invited to choose between: (1) It is a good idea but I will study some of them; (2) It is a good idea but I don't have the time to study them; (3) It is not a good idea, because I don't want to disperse myself with studying external resources; (4) other point of view.

2. A closed question asking respondents to suppose they are attending an online course and to precise their behavior when they don't understand some of its notions. Do they repeat the course or look for other online courses ?
3. For the learners who prefer looking for other courses, another closed question asks them to precise whether the recommended resources allow them to save time compared to the time they spend to look for external resources by themselves.

6.2 Evaluation Results

In this chapter we summarize the responses of teachers and learners to the questions presented in the evaluation questionnaires.

Overall evaluation of the Approach

Experts Opinions. After consulting the resources selected by our system, all experts agreed that recommending external resources, during a MOOC, is a good idea. Regarding the criteria defined in our solution to take into account the specificities of the MOOC in the recommended resources, we obtain these results. All experts assigned the maximum score of 3 to the criterion relating to the level of knowledge provided by the resource. All experts agreed on the importance to recommend resources allowing the learner to acquire the knowledge level supposed to be acquired by attending the MOOC. Three of the experts assigned the maximum score of 3 to the criterion relating to the learning time. According to them, it is very important to take into account the learning time of the recommended resources. The fourth expert assigned the score of 2 to this criterion. He considers that it is an important criterion but still less important than the one relating to the knowledge level.

Two experts assigned the maximum score to criterion relating to the granularity of the resource. The other two experts assigned the score 2 to this criterion. For them, the fact that the granularity of the recommended resources is adequate with the internal resources of the MOOC is important, but it is less important than the adequacy of the knowledge level and the learning time.

Learners Opinions. After consulting the resources selected by our solution, 60% of Foule Factory respondents endorse the recommendation of external pedagogical resources, when they didn't correctly answer the evaluation test presented in a course they are attending. Another 25.7% of the respondents consider that it is a good idea but they have a problem with the time they will spend to learn additional resources. The lack of additional time available to study the recommended resources, confirms the importance of the learning time criterion taken into account in our solution. The last 14.3% of respondents don't consider the recommendation of external resources as a good idea and they do not want to disperse themselves by studying external resources in addition to the course they are attending.

On the other hand, 55.2% of the respondents prefer to search for other online courses when they are taking a course and they don't understand some of its notions. The remaining respondents prefer to repeat the same course. Among those who prefer to search for other online courses, 72.4% find that the recommended resources allow them to save time, compared to the time they would have spent looking for resources on the web.

Evolution of Learners Knowledge after Recommendations

After consulting the recommended resources, 81.6% of Foule Factory respondents find that those recommended for the notion "relational model" allowed them to acquire new knowledge. 69.5% find that those recommended for the notion "DBMS" allowed them to acquire new knowledge about this notion and 60.5% find that those recommended for the notion "SQL" helped them to learn new knowledge.

Based on the respondents answers to the same quiz presented before and after recommendations, the Table 1 resumes the percentages of correct answers. By examining the results, we note an improvement in all the answers after the recommendations except for one question dealing with the notion of "Relational Model" Q_2 for which the percentage of correct answers decreased. To understand the causes of this decrease, we started with studying the resources we recommend to the respondents to help them to acquire the notion of "Relational Model". Among the four recommended resources, there is R_1 that does not contain the information which is the subject of the question Q_2. There are also two resources R_2 and R_3 that include the information and the resource R_4 that includes the information but that is very voluminous with 37 modules and that require at least 3 hours to learn its content.

Table 1. Answers to the evaluation quiz before and after recommendations.

Recommandations	Relationnal model			DBMS			SQL	
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
Before	48%	42%	41%	31%	51%	24%	42%	16%
After	76%	34%	52%	41%	68%	28%	43%	20%

Then we selected the 15 respondents who answered correctly the question Q_2 before the recommendations and incorrectly after the recommendations. By looking the resources attended by these respondents, we found that 10 of them have opted to attend the resource R_1 not dealing with the information processed by the question Q_2 and 3 of them choose the voluminous resource R_4. So, a possible explanation for these results is that the respondents didn't find the answer in the chosen resource or they didn't follow it until the end and they replied randomly to the question.

It is also important to note that two of the three respondents who opted to attend R_4 left some comments about its length: *"many pdf documents to open"* and *"too long"*. Hence the importance of the criteria relating to the granularity and the learning time of the recommend resources, defined in our solution and the importance of describing these information in the metadata of the OER to improve the recommendation's results.

Adequacy of recommendations with both the MOOC and the Learner Profiles

Application of Semantic Similarity Measure. For each MOOC's notion, the fourth experts ticked the resources allowing to acquire knowledge about this notion, from the set of all the resources selected by the initial query of our MORS system (the query defined in the PreSearch module). To assess the benefit of using the semantic similarity measure, we define two measures of precision. The first one is carried out on the resources selected by the initial query to assess whether the keyword search adopted in this query allows selection of relevant resources dealing with of knowledge element (subject of the recommendation). The second measure of precision aims to assess whether the refinement of the initial set of resources by using the semantic similarity measure increases the precision rate. Our objective is to have a precision rate closer to 100% after applying the semantic similarity on the descriptions of the resources selected by the initial query.

As explained previously, our goal is not to select all the relevant resources that exist in the external repositories but to be assured that the limited resources recommended to learners are relevant. In this context, the relevance is that the resource allow the acquisition of the notion which was not well mastered by the leaner and whose lack of knowledge triggered the recommendation process. For this reason, we define to measures of precision adapted to our goals:

- A first precision rate Precision_{ad1}. Precision_{ad1} represents the percentage of relevant resources among all the resources selected by the initial query.

$$\text{Precision}_{ad1} = \frac{|R_{RT}|}{|R_{IQ}|}$$

  Where R_{RT} represents the resources ticked by experts as relevant resources and R_{IQ} represents the set of the resources selected by the initial query of our system MORS.
- A second precision rate Precision_{ad1}. Precision_{ad2} represents the percentage of relevant resources among all the resources selected after the refinement using semantic similarity.

$$\text{Precision}_{ad1} = \frac{|R_{RT}|}{|R_{SS}|}$$

  Where R_{RT} represents the resources ticked by experts as relevant resources and R_{IQ} represents the set of the resources selected after applying semantic similarity.

By reviewing the results, we notice a significant increase in precision rates for resources related to the notion "Relational Model" and "DMS". For the notion "SQL", an increase is detected depending on the answers of two experts (expert 1 and expert 3) whereas a decrease is detected depending on the answers of the two others (expert 2 and expert 4). To understand this decrease, we studied the answers of the fourth experts on the relevance of the resources selected after applying the semantic similarity. Among the four selected resources, there is a resource which is considered as relevant by the expert 1 and 3 and no relevant by the expert 2 and 4. This is a course on database with a section introducing the notion "SQL", without giving details. A possible explanation of the difference between experts' opinions is that presenting not detailed information about the notion is considered, by some experts, sufficient to acquire some basic knowledge about the notion and not sufficient by the others.

In our case, the objective of recommendations is to help the learner to acquire a knowledge level allowing him to attend the MOOC until the end. From that comes the importance of the constraint defined in our solution to just retain resources which provide a knowledge level in a specific knowledge element (pre-requisite or learning objective) upper or equal to the knowledge level defined in the MOOC. Unfortunately this information doesn't exist in the metadata of OER but recovering it later from learners who consult the recommendations may improve the results.

Adequacy of Recommendations with the MOOC Profile. In this step, we calculated the averages of scores given by the experts to each resource according to its satisfaction to the criteria related to the granularity and the learning time and the knowledge level.

It is important to note that in this step we are interested in the resources of the final set which is recommended to the learner (after applying the selection by constraints and the selection by semantic similarity).

Concerning the criterion related to the learning time of the resource, we obtain these results. For the notion "Relational Model", the resource with the worst note, represents a voluminous course with 37 modules and 3 hours at least to learn its content. The resource which receives the best note is a pdf of 84 pages in the form of slides where each page some lines. Therefore, we can deduce that the resource's volume (the number of its pages in this case) cannot give the exact information about the duration needed to assimilate its content.

Having the exact information about the learning time of the resource could support its recommendation by our approach. Unfortunately, information about learning times, in the majority of cases, is not included in the OER descriptions.

Concerning the notion "DBMS", the two best notes were assigned to resources presented as HTML pages. Although these courses are long, the notes can be explained by the fact that each of them offers a well presented summary that allows learners to go directly to a section dealing with a specific notion. The worst rating was assigned to a resource in the form of a quiz. A quiz presents questions and answers to learners and in most cases, it is used for the assessment

of knowledge. Therefore, to answer a quiz, the learner is, in many cases, obliged to consult other courses which will increase the duration necessary to master the knowledge.

For the notion "SQL", the best note was assigned to a resource presented as a set of short videos of 2 to 4 min. A quiz resource is considered as the resource that requires the longer learning time.

Based on the experts responses on the relevance of OERs according to the criterion of the learning time, we can notice that two learning durations are to be taken into consideration.

The first duration concerns that necessary to assimilate the total content of the resource, and the second duration concerns that necessary to acquire knowledge about a specific notion, by consulting the resource. We can also notice that the way in which the resource is presented influences the duration to be spent by the learner to master its content.

In some cases, we noticed that experts have difficulty to rate resources criterion-by-criterion and their scores are influenced by several criteria at once. Such is the case of the criterion of granularity and knowledge level. For example, for the notion "DBMS", the same resource received the worst notes about its satisfaction to the criterion of granularity and knowledge level. The same resource received also the best note for its satisfaction to the criterion about granularity and knowledge level.

Adequacy of Recommendations with the Learner Profile. A large proportion of respondents, between 40% and 90%, considered the recommended resources as difficult resources in relation to their knowledge. Some of them left comments to express this difficulty: *"quite complex"*, *"very technical"* and *"I do not have the necessary bases"*.

This highlights the constraint defined in our approach that the prerequisites of recommended resources must be mastered by the learner.

As regards the criterion about learner's learning style, we calculated the average values of the scores assigned by respondents. After reviewing the results, we noticed that the resources presented in the form of videos and HTML pages with a properly presented menu to access resources' sections, are the most appreciated. Long resources are less appreciated. The worst notes were assigned to quizzes. This can be explained by the fact that a quiz does not explain the notion especially for beginner learners.

By studying respondents' learning styles based on their answers to the (ILS) questionnaire [16], we found that 29% of respondents are visual rather than verbal and 16% are 100% visual. This can explain the fact that the resources properly presented and including videos and images received better notes. We also found that 36% of respondents are sensorial rather than intuitive and 18% are 100% sensorial. This can explain the fact that resources containing exercises and applications were more appreciated than those based solely on theory.

7 Conclusions

In this article, we have presented our recommender system MORS. It dynamically recommends external OERs during a MOOC when a lack of knowledge of the learner is detected. To this end, we use performance degrees to qualify the difficulty of the MOOC knowledge elements and also to measure the acquisition of these elements by the learner. MORS adapts the results according to the progress of the MOOC and the knowledge acquired by the learner.

The resources to be recommended by MORS, at the beginning of the MOOC or at the end of each section, are calculated so that they are adapted to some characteristics of the learner and of the MOOC he is attending, at the moment of the recommendation. Therefore, our system consists of two processes responsible for generating and updating the learner profile and the MOOC profile throughout the MOOC, by extracting the necessary information. Based on these two profiles, if a lack of knowledge is detected for a certain knowledge element, the calculation of the recommendation is triggered. It starts with the PreSelection process that requests the repositories of OERs descriptions to select an initial set of OERs dealing with the knowledge element. Then, the refinement process generates the final set of OERs to be recommended to the learner. To his end, the refinement process performs selection and ranking operations based on mandatory and optional criteria. These criteria are defined to take into account both some characteristics of the learner and some specificities of the MOOC.

The first assessment of our system showed the importance of the criteria used in our recommendation algorithm to satisfy some learner and MOOC characteristics. This also allowed us to have an idea about the criteria priority according to the experts and to retrieve other proposals of criteria like the difficulty and the creation's date of the resource to enhance our solution. The evaluation also showed that in most cases, the resources recommended to the learners allowed them to acquire better knowledge in the notions of the MOOC. It is also necessary to perform another qualitative assessment to understand some of the experts answers that were not clear and easy to interpret. It is important to note that our solution uses the OERs metadata stored in accessible repositories, hence depends on the availability and the quality of these metadata. Therefore, in some cases, we have not been able to verify the fulfillment of certain criteria by the recommended OER because of the lack of metadata information, which affects the quality of recommendations.

Our short-term objective in the intermediate future is to improve our solution in the light of the results obtained from the evaluation, for example by adding other criteria in the recommended resources and assigning weights values to the optional criteria based on the experts answers. We also seek to integrate our recommender system in an existing MOOC in order to assess how it will be working under real conditions. This integration will allow us to get better feedback on the quality of recommended OERs and how much they support learners during the MOOC. We will be able also to deduce the performance degrees provided by the OERs based on learners responses to the MOOC quiz, after attending the recommended OERs.

References

1. Agrawal, A., Venkatraman, J., Leonard, S., Paepcke, A.: YouEDU: addressing confusion in MOOC discussion forums by recommending instructional video clips (2015)
2. Alario-Hoyos, C., Leony, D., Estévez-Ayres, I., Pérez-Sanagustín, M., Gutiérrez-Rojas, I., Kloos, C.D.: Adaptive planner for facilitating the management of tasks in MOOCs. In: V Congreso Internacional sobre Calidad y Accesibilidad de la Formación Virtual, CAFVIR, pp. 517–522 (2014)
3. Bansal, N.: Adaptive recommendation system for MOOC. Indian Institute of Technology, pp. 1–40 (2013)
4. Chowdhury, G.G.: Introduction to Modern Information Retrieval. Facet Publishing, London (2010)
5. Fasihuddin, H.A., Skinner, G.D., Athauda, R.I.: Personalizing open learning environments through the adaptation to learning styles. In: ICITA (2014)
6. Felder, R.M., Silverman, L.K., et al.: Learning and teaching styles in engineering education. Eng. Educ. **78**(7), 674–681 (1988)
7. Hajri, H., Bourda, Y., Popineau, F.: Querying repositories of OER descriptions: the challenge of educational metadata schemas diversity. In: Conole, G., Klobučar, T., Rensing, C., Konert, J., Lavoué, É. (eds.) EC-TEL 2015. LNCS, vol. 9307, pp. 582–586. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24258-3_64
8. Hajri, H., Bourda, Y., Popineau, P.: A system to recommend open educational resources during an online course. In: 10th International Conference on Computer Supported Education, pp. 99–109 (2018)
9. Imran, H., Belghis-Zadeh, M., Chang, T.-W., Graf, S., et al.: PLORS: a personalized learning object recommender system. Vietnam J. Comput. Sci. **3**(1), 3–13 (2016)
10. Kolb, A.Y.: The Kolb learning style inventory-version 3.1 2005 technical specifications, vol. 200, p. 72. Hay Resource Direct, Boston (2005)
11. Kolukuluri, S.: XBlock-courseware component architecture. Ph.D. thesis, Indian Institute of Technology, Bombay, Mumbai (2014)
12. Li, S.-W.D., Mitros, P.: Learnersourced recommendations for remediation. In: 2015 IEEE 15th International Conference on Advanced Learning Technologies (ICALT), pp. 411–412. IEEE (2015)
13. Maran, V., de Oliveira, J.P.M., Pietrobon, R., Augustin, I.: Ontology network definition for motivational interviewing learning driven by semantic context-awareness. In: 2015 IEEE 28th International Symposium on Computer-Based Medical Systems (CBMS), pp. 264–269. IEEE (2015)
14. Paquette, G., Mariño, O., Rogozan, D., Léonard, M.: Competency-based personalization for massive online learning. Smart Learn. Environ. **2**(1), 4 (2015)
15. Sahebi, S., Lin, Y.-R., Brusilovsky, P.: Tensor factorization for student modeling and performance prediction in unstructured domain. In: Proceedings of the 9th International Conference on Educational Data Mining, pp. 502–506. IEDMS (2016)
16. Soloman, B.A., Felder, R.M.: Index of learning styles questionnaire (1999). Accessed 26 Mar 2003
17. Sunar, A.S., Abdullah, N.A., White, S., Davis, H.C.: Personalisation of MOOCs: the state of the art (2015)

université PARIS-SACLAY

ÉCOLE DOCTORALE
énergétiques, matériaux
et géosciences (SMEMAG)

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies
de l'information et de
la communication (STIC)

ÉCOLE DOCTORALE
Sciences du sport, de la

Titre : Approche Logique de la Personnalisation dans les Environnements Informatiques pour l'Apprentissage Humain

Mots clés : Programmation logique, adaptation à l'utilisateur, environnements informatiques pour l'apprentissage humain

Résumé :Nous présentons ici un ensemble de travaux autour de l'adaptation à l'apprenant dans les environnements informatiques pour l'apprentissage humain (EIAH). Ces travaux s'appuient sur une approche classique de l'intelligence artificielle, fondée essentiellement sur la logique et une approche agent. Nous développons plusieurs arguments sur le bienfondé du calcul des situations pour piloter ces EIAH. Nous proposons une méthode de transformation des programmes logiques pour adapter le comportement de l'agent à l'utilisateur, ou bien pour donner une personnalité à l'agent. Nous discutons également la possibilité d'intégrer cette approche dans une plateforme de MOOC, pour accompagner les apprenants et pour leur fournir des recommandations personnalisées.

Title : Logical Approach of Personalization in Technology Enhanced Learning

Keywords : Logic programming, user adaptation, technology enhanced learning

Abstract : We present here a body of works dealing with learner adaptation in Technology Enhanced Learning (TLE) platforms. These works are based on a classical approach of artificial intelligence, essentially grounded on logic and an agent approach. We develop several arguments on the appropriateness of situation calculus to drive these platforms. We propose a method of transforming logic programs to adapt the agent's behavior to the user, or to give a personality to the agent. We also discuss the possibility of integrating this approach in a MOOC platform, to accompany learners and to provide them with personalized recommendations.