Vers l'analyse de ressources d'apprentissage de la programmation à l'aide d'un référentiel de types de tâches

Sébastien Jolivet, Eva Dechaux, Anne-Claire Gobard, Patrick Wang

Actes du colloque Didapro 10 sur la Didactique de l'informatique et des STIC
Louvain-La-Neuve, Belgique, 30 janvier au 1er février 2024

Vers l'analyse de ressources d'apprentissage de la programmation à l'aide d'un référentiel de types de tâches

Sébastien Jolivet[1][0000−0003−3915−8465], Eva Dechaux[2][0000−0001−8579−1398], Anne-Claire Gobard[3][0000−0002−2866−084X], and Patrick Wang[4][0000−0003−3117−8189]

[1] IUFE & TECFA, Université de Genève, Suisse
sebastien.jolivet@unige.ch
[2] IUFE, Université de Genève, Suisse
eva.dechaux@unige.ch
[3] Lycée Kastler, Académie de Versailles, France
anne-clair.gobard@ac-versailles.fr
[4] Haute école pédagogique du canton de Vaud, Lausanne, Suisse
patrick.wang@hepl.ch

Résumé Dans cette contribution nous abordons la problématique de la description didactique de ressources d'apprentissage de la programmation. Nous nous concentrons sur une compréhension fine des savoirs en jeu dans ces ressources. À cette fin, nous définissons un référentiel de types de tâches, puis l'exploitons pour décrire des problèmes de programmation en illustrant le processus sur deux exemples. Dans la conclusion, nous proposons quelques perspectives pour obtenir, sur cette première base, une description plus complète des ressources.

Keywords: Apprentissage de la programmation · Description de ressource · Référentiel de types de tâches.

1 Introduction

L'enseignement de l'informatique, comme discipline scolaire, s'est généralisé ces dernières années. Ceci entraîne la production de nombreuses ressources d'enseignements. Elles peuvent être en format papier (brochures, manuels, cahiers d'exercices) ou numérique (plateformes en ligne, EIAH, etc.). Avec le partage et la diffusion des ressources, la question n'est pas de savoir s'il existe une ressource qui aborde telle ou telle notion, de telle ou telle manière, mais bien de pouvoir identifier les ressources adéquates à un projet didactique donné. La description des ressources selon des critères de nature bibliothécaire est standardisée – voir par exemple le standard international IEEE LOM (Learning Object Metadata) dont la place centrale est mise en évidence dans [13] (p. 94). Dans [10] les auteurs montrent les limites de différentes approches pour proposer une description didactique des ressources d'apprentissage. Or une telle description est un enjeu important pour aider un agent, humain ou logiciel, à choisir les bonnes ressources

pour un projet didactique (enseignant qui prépare sa classe, EIAH avec de l'apprentissage adaptatif, etc.). Dans cet article nous apportons une contribution partielle à la description didactique de ressources : 1) en présentant un référentiel permettant de décrire les savoir-faire, et indirectement les savoirs, mobilisés dans une ressource ; 2) en illustrant, à l'aide de deux exemples, son exploitation pour décrire les types de tâches à mobiliser lors de la construction d'une solution correcte pour des exercices de programmation.

L'utilisation d'un référentiel permet, au-delà de contribuer à la description didactique de la ressource, de répondre à un problème important lié à la description de ressources, qui est qu'elle ne doit pas dépendre de l'annotateur.

Dans [11] et [12] nous avons présenté un premier référentiel de types de tâches mobilisés dans les premiers apprentissages de la programmation et l'avons utilisé pour étudier trois EIAH d'apprentissage de Python. Ce premier référentiel présente certaines limites, en particulier en termes de couverture des types de tâches liés au travail sur la conception d'algorithmes et l'implémentation de programmes. Nous ne reprenons pas le détail de la méthode mise en œuvre pour le construire, signalons simplement qu'elle est en particulier basée sur une étude de divers moyens d'enseignement pour le secondaire.

Dans cette contribution (Sect. 2), nous enrichissons le référentiel et précisons les fondements de sa construction. nous commençons par présenter le référentiel enrichi utilisés, ainsi que les fondements de sa construction. La Sect. 3 explique son utilisation pour décrire des ressources de type exercice en nous appuyant sur deux problèmes fréquemment proposés lors de l'apprentissage de la programmation au secondaire : le problème du *juste prix* et le jeu *pierre, feuille, ciseaux*. Dans chaque section nous pointons les apports et les limites des éléments présentés.

2 Référentiel de type de tâches de la programmation

Dans cette section nous commençons par identifier différents travaux relatifs à la description de l'activité de programmation, puis nous expliquons les spécificités et les fondements de notre référentiel. Enfin, nous insistons sur les principales nouveautés par rapport à la version présentée dans [11]

2.1 Travaux existants

Comprendre et décrire ce que signifie *programmer* est interrogé depuis plus de 50 ans, dans [7] les auteurs proposent une perspective historique. Ils identifient deux grands courants qui ont orienté les motivations et finalités de ce questionnement. Le premier est une approche psychologique de l'activité de programmation, en particulier pour identifier un "bon développeur" pour l'industrie, le second est centré sur l'apprentissage et l'enseignement de la programmation.

Nous nous intéressons plutôt à l'approche psychologique. Dans [14], les auteures situent l'activité de programmation dans le domaine plus large de la

conception (*design*). La programmation y est décrite au travers de grands domaines (compréhension du problème, conception, codage, maintenance) qui organisent les tâches à réaliser, mais ces tâches ne sont pas explicitement identifiées.

Plusieurs travaux visent également à répertorier les éléments constitutifs d'un langage de programmation particulier (e.g., [3,18]) ou partagés par plusieurs langages (e.g., [6,15,17]) afin d'en construire une représentation sous forme d'ontologies. Ces travaux mettent en évidence les objets de savoirs que les apprenants peuvent rencontrer. Le référentiel présenté dans la suite de cet article se distingue de ces travaux en explicitant les manipulations possibles de ces objets de savoir dans le cadre de la résolution d'exercices de programmation.

2.2 Délimitations et fondements du référentiel

Dans ce travail, nous nous situons dans le paradigme de la programmation impérative. Nous considérons qu'un algorithme (ou un programme) est une séquence d'instructions permettant la réalisation d'une tâche t ; t pouvant être de nature et d'ampleur très diverses (e.g., calculer la somme de trois entiers, gérer le décollage d'une fusée, trier une liste, etc.). Nous considérons également qu'un algorithme s'écrit dans un registre autre que celui du langage de programmation (e.g., pseudo code, logigramme, langage naturel) tandis qu'un programme est nécessairement écrit avec un langage de programmation. Cette distinction est motivée par les différents moyens existants pour concevoir un algorithme ou implémenter un programme et l'effet de ces moyens sur l'activité induite. En effet, l'utilisation d'un environnement de développement fournit des moyens spécifiques pour réaliser certaines tâches (par exemple l'exécution, éventuellement pas à pas, d'un programme ou l'affichage de messages en console pour comprendre son fonctionnement). Enfin, le référentiel ne privilégie aucun langage de programmation, ce qui implique que nous ne définissons pas de types de tâches basés sur des caractéristiques syntaxiques (e.g., l'opérateur ternaire `?` en Javascript) ou sémantiques (e.g., la compréhension des listes en Python) différentes d'un langage à l'autre.

Notre travail se place dans une approche anthropologique visant à décrire l'activité humaine mise en œuvre lorsque l'apprenant doit réaliser une tâche de programmation ainsi que ses fondements théoriques. Nous nous appuyons sur le modèle praxéologique développé dans le cadre de la théorie anthropologique du didactique [2], que nous ne détaillons pas dans cet article car non nécessaire pour sa compréhension. Dans ce modèle, l'activité humaine est décrite à l'aide d'un quadruplet constitué de deux blocs, le bloc praxique avec un type de tâches et le moyen de réaliser ce type de tâches (appelé technique dans le modèle) et le bloc du logos qui est constitué des éléments du savoir qui permettent de justifier, organiser, guider la technique, et sur lequel nous ne reviendrons pas dans la suite de cet article. Un des intérêts de choisir ce cadre est qu'il permet de décrire toutes les activités humaines liées à la programmation (modélisation, travail algorithmique, codage, etc.) à l'aide du même cadre et identifie les liens entre savoir-faire et savoirs.

2.3 Structuration et présentation du référentiel

Pour construire le référentiel, nous nous sommes tout d'abord appuyés sur un état de l'art relatif à la question "qu'est ce que programmer ?" (voir par exemple [8,9,16,14]). La synthèse que nous en retenons est qu'il y a des activités qui portent sur d'une part les algorithmes et d'autre part les programmes (conception et maintenance). C'est cette partie du référentiel que nous développons dans cette contribution.

D'autre part, ces activités mobilisent notamment des éléments fondamentaux que sont les variables et les structures de contrôle (boucles bornées et non bornées, instructions conditionnelles). Une seconde partie du référentiel, détaillée dans [11], précise les types de tâches spécifiques à ces éléments.

Pour des raisons de place nous ne présentons que des extraits [5] du référentiel en nous concentrant sur les algorithmes et les programmes. La seconde partie du référentiel est toutefois aussi exploitée dans la section 3.

Pour organiser les principaux types de tâches relatifs à l'activité sur les algorithmes (*types de tâches relatifs aux algorithmes*, Fig. 1) et les programmes (*types de tâches relatifs aux programmes*, Fig. 2) nous nous inspirons des travaux de Pennington et Grabowski [14]. Les auteures distinguent deux activités fondamentales, les tâches de *création d'un programme (composition of a program)* et les tâches de *compréhension d'un programme (comprehension of a program)*, que nous utilisons comme premier niveau de structuration avec les verbes *Produire* et *Analyser*. Dans les types de tâches relatifs aux programme nous ajoutons la catégorie *Mettre au point un programme* qui correspond à l'activité *Maintenance* identifiée dans [14].

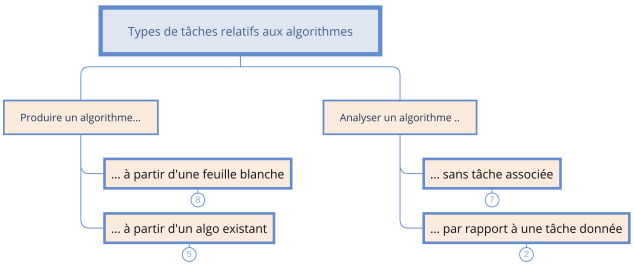

Figure 1. Classification des types de tâches relatifs aux algorithmes.

La catégorie *Produire* est relative au cas où le type de tâches vise à la production d'un algorithme ou d'un programme. Nous distinguons deux situations : 1) on ne dispose d'aucune base de travail, 2) un algorithme ou un programme est déjà existant et il s'agit de le faire évoluer.

5. L'intégralité des éléments disponibles est accessible à ce lien `https://link.infini.fr/ref_prog`

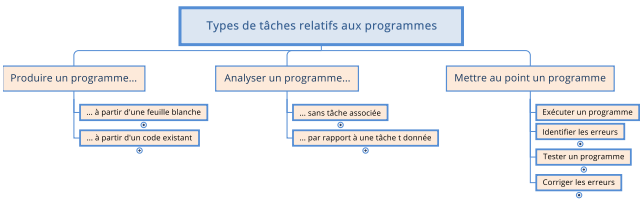

Figure 2. Classification des types de tâches relatifs aux programmes.

La catégorie *Analyser* est aussi scindée en deux cas selon ce qui guide l'analyse. Soit elle porte sur l'objet considéré de manière indépendante de toute tâche à réaliser, par exemple déterminer la complexité algorithmique d'un algorithme ne nécessite pas de lier cet algorithme à la tâche qu'il réalise. Soit la tâche t que doit réaliser l'algorithme ou le programme sert de critère pour mener l'analyse, comme c'est le cas par exemple quand il s'agit de prouver la correction d'un algorithme.

La catégorie *Mettre au point un programme* regroupe différents types de tâches relatifs à la situation où l'on dispose d'un programme non fonctionnel et qu'il s'agit de le faire fonctionner (tester, déboguer, etc.).

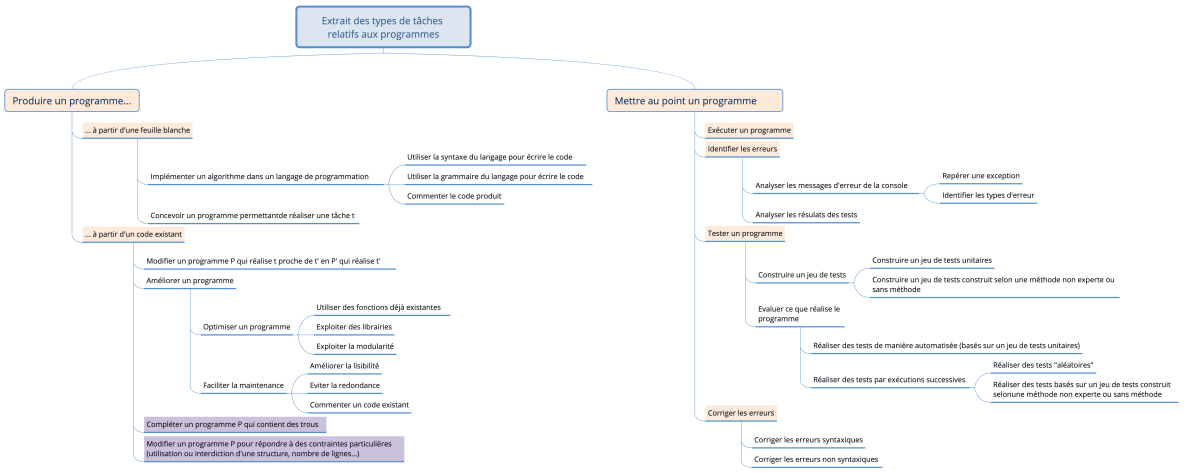

Figure 3. Principaux types de tâches relatifs aux programmes.

Pour être réalisés, chacun de ces types de tâches nécessite de mettre en œuvre d'autres types de tâches. Nous en présentons quelques-uns en Fig. 3, relatifs aux catégories *Produire un programme* et *Mettre au point un programme*. Ils sont organisés à l'aide de nouveaux types de tâches *Identifier les erreurs ; Tester un programme ; Corriger les erreurs*, pour lesquels on précise à nouveau différents types de tâches permettant de les réaliser.

2.4 Synthèse sur le référentiel : apports et limites

Les éléments présentés dans la Sect. 2.3, ainsi que ceux décrits dans [11] , permettent d'avoir un référentiel qui couvre à la fois les types de tâches relatifs aux objets algorithmes et programmes et ceux relatifs aux variables et structures de contrôle.

Il est structuré selon la nature de l'activité à réaliser (produire à partir de rien ou avec une base ; analyser de manière autonome ou relativement à une tâche à réaliser ; mettre au point). Les niveaux inférieurs permettent d'identifier des types de tâches nécessaires à la réalisation de ces types de tâches de plus haut niveau.

Le nombre de types de tâches identifiés est déjà conséquent. Il est aussi nettement plus précis que ce que l'on peut rencontrer dans les curriculums ou dans les représentations ontologiques signalées dans la Sect. 2.1. Il reste cependant possible de raffiner encore ce travail sans remettre en cause la structuration en place. Le niveau de finesse visé dépend essentiellement de l'exploitation souhaitée du référentiel. Par exemple :

- ajouter des types de tâches permettant de mettre en œuvre dans un langage spécifique un type de tâches déjà identifié. Ainsi on pourrait définir le type de tâches *Indenter un ensemble d'instructions* comme ingrédient permettant de réaliser le type de tâches *Déterminer les instructions à exécuter à chaque itération* dans le cas de Python.
- préciser certains ingrédients des techniques permettant de réaliser un type de tâches. Ainsi, on peut ajouter *Dresser un tableau de valeurs des variables* comme moyen, non unique, de réaliser *Déterminer la valeur d'une ou plusieurs variables à l'issue d'une séquence d'instructions*. Ce type de tâches, lui-même, pourra être réalisé de diverses manières en fonction des outils à disposition (e.g., papier-crayon ou IDE) et ainsi donner lieu à la définition de nouveaux types de tâches.

Enfin, le travail d'analyse des moyens d'enseignement nous a amené à identifier certains types de tâches qui n'ont pas de raison d'être si on se positionne du point de vue de l'activité du développeur, mais qui ont une existence comme types de tâches didactiques. Ils sont identifiés en violet dans notre référentiel, par exemple le type de tâches *Compléter un programme à trous* (Fig. 3).

Comme évoqué en Sect. 2.2 nous appuyons notre travail sur l'approche praxéologique. Cela signifie que pour chaque type de tâches il y a une praxéologie qui permet de lier le type de tâches et la technique permettant de le réaliser aux éléments du logos les justifiant. Ainsi ce référentiel, construit et présenté autour des

savoir-faire, permet aussi d'identifier les savoirs. La conclusion est l'occasion de dessiner d'autres perspectives sur les utilisations et évolutions de ce référentiel.

3 Description de la résolution de deux problèmes

Dans cette section nous revenons sur notre problématique de description didactique de ressources d'enseignement de la programmation. Un premier élément, nécessaire mais non suffisant comme expliqué dans [10] , est l'identification des savoir-faire et savoirs mobilisables pour réaliser les tâches prescrites.

Nous nous concentrons sur cet aspect et montrons en quoi le référentiel contribue à répondre à cet objectif. La démarche se déroule en deux étapes. Tout d'abord il s'agit d'identifier les types de tâches situés aux niveaux *algorithme* et *programme* qui sont mobilisés pour produire un programme permettant de réaliser la tâche t visée. Dans un deuxième temps, à partir d'une ou plusieurs solutions du problème, nous identifions les types de tâches relatifs aux variables et structures de contrôle. Nous illustrons maintenant cette démarche sur deux problèmes qui peuvent être rencontrés lors de l'apprentissage de la programmation au secondaire : les jeux *Pierre - Feuille - Ciseaux* et *Devine le juste prix*.

3.1 Types de tâches du niveau algorithmes et programmes

Nous considérons que, quel que soit le problème à résoudre (i.e. la tâche t à faire réaliser par le programme visé), l'ensemble [6] de types de tâches suivant est potentiellement mobilisé : *Concevoir un algorithme réalisant une tache t* ; *Concevoir un programme réalisant une tache t* ; *Exécuter un programme* ; *Tester un programme*. La réalisation de chacun de ces types de tâches va mobiliser au moins une partie des types de tâches identifiées dans le référentiel (par exemple *Modéliser* ; *Choisir une structure de contrôle* ; etc.). Le fait que ces types de tâches soient effectivement mobilisés ou non peut être déterminé à partir d'une analyse de l'énoncé permettant la dévolution du problème à l'apprenant. Nous n'abordons pas cette étape supplémentaire dans ce document. Nous illustrons maintenant la deuxième étape en nous basant sur une solution "classique".

3.2 Pierre - Feuille - Ciseaux

L'énoncé considéré pour mener l'analyse de ce problème est le suivant : *écrire un programme qui permet de jouer une manche de* Pierre - Feuille - Ciseaux *contre un ordinateur qui joue de manière aléatoire. Le joueur doit savoir si le résultat est gagné, nul ou perdu.* Pas de gestion des saisies erronées attendue. Un extrait du code, en Python, d'une solution possible à ce problème est :

6. Nous utilisons le terme *ensemble* à dessein compte tenu qu'il n'y a pas de chronologie systématique entre ces types de tâches et qu'ils peuvent être mobilisés plusieurs fois dans une démarche, partiellement, itérative.

```
import random
choix = int(input( "PIERRE (1) ou FEUILLE (2) ou CISEAUX (3)? "))
hasard = random.randint(1,3)
if choix == hasard :
    print("Match nul !")
elif choix == 1 and hasard == 3:
    print("Victoire !")
# Autres branches absentes pour des raisons de longueur d'article
else :
    print("Perdu :-(")
```

Cette solution permet d'identifier la mobilisation des types de tâches suivants : *Déclarer une variable* ; *Affecter une valeur à une variable* ; *Utiliser une variable dans une expression* ; *Concevoir une instruction conditionnelle.*

3.3 Devine le juste prix

L'énoncé considéré pour mener l'analyse de ce problème est le suivant : *écrire un programme qui permet à l'utilisateur de deviner un nombre entier entre 1 et 100, choisi aléatoirement par l'ordinateur. Il fait des propositions et indique si le nombre est plus grand, plus petit ou affiche gagné si le nombre est trouvé. La partie s'arrête dès que c'est gagné.* Il n'est pas demandé que le programme gère des saisies erronées ou compte le nombre d'essais. Un extrait du code, en Python, d'une solution possible à ce problème est :

```
import random
devine = 0
prix = random.randint(1,100)
while prix != devine:
    devine = int(input("Devinez le prix : "))
    if devine > prix :
        print("C'est moins !")
    elif devine < prix :
        print("C'est plus !")
    else :
        print("Bravo, le prix est bien", devine)
```

Cette solution permet d'identifier la mobilisation des types de tâches suivants : *Déclarer une variable* ; *Affecter une valeur à une variable* ; *Utiliser une variable dans une expression* ; *Concevoir une instruction conditionnelle* ; *Concevoir une boucle non bornée.*

3.4 Synthèse sur la description de ressources : apports et limites

Nous venons d'illustrer l'utilisation du référentiel pour identifier les savoir-faire mobilisables pour résoudre un problème donné. Tout d'abord en établissant un premier ensemble de types de tâches permettant d'aller de l'analyse du problème à résoudre à la production d'un programme puis à sa validation, ou sa correction

s'il ne réalise pas la tâche visée. Certains de ces types de tâches sont nécessairement précédés par d'autres (on ne produit pas un programme avant d'avoir réalisé un travail sur l'algorithme), d'autres vont nécessiter de remobiliser des types de tâches déjà utilisés au moins une fois précédemment. Par exemple, le type de tâches *Corriger les erreurs non syntaxiques* va exploiter des types de tâches pouvant relever des catégories *Analyser un programme* et/ou *Produire un programme à partir d'un code existant*, *Analyser un programme* ayant possiblement déjà été utilisé lors de la réalisation du type de tâches *Chercher un programme réalisant une tâche t' proche de la tâche t.*

Puis, sur la base d'une solution au problème, nous utilisons à nouveau le référentiel pour identifier un second ensemble de types de tâches à mobiliser, qui sont, cette fois, relatifs aux structures de contrôle et variables qui sont mobilisées.

Ceci permet de proposer pour chaque problème une étude des savoirs-faire en jeu et, en remontant au niveau des praxéologies, d'identifier les savoirs mobilisés. Par exemple, par rapport au type de tâches *Concevoir une boucle non bornée*, on va avoir des savoirs tels que *la définition de la valeur de vérité d'une expression* ou *le fait qu'une boucle non bornée est constituée d'un test d'arrêt et d'un corps.*

En travaillant sur un corpus de ressources, cette analyse des ressources peut servir à évaluer la couverture d'un ensemble de savoirs ou types de tâches visés, comme cela a été fait dans [12] où il est montré que des catégories entières de types de tâches sont absentes des EIAH analysés.

La description de ressources, telle que présentée ici, présente trois limites significatives :

- La première est intrinsèque au niveau de précision du référentiel exploité. Par exemple, dans les types de tâches relatifs aux instructions conditionnelles, nous identifions le type de tâches *Déterminer les différentes branches* mais notre référentiel ne nous permet pas de distinguer s'il y a 2 ou 10 branches dans la solution analysée. Ou encore, nous identifions en Sect. 3.3 les types de tâches *Concevoir une IC* ; *Concevoir une boucle non bornée* mais notre description ne rend pas compte du fait qu'elles sont imbriquées dans cette solution. Si on estime que ce type de précision présente un intérêt pour la qualité de la description des exercices, au regard de l'usage souhaité, il est possible de préciser le référentiel en reprenant par exemple la démarche présentée dans [10] qui exploite les générateurs de types de tâches définis dans T4-TEL [1].
- La deuxième est liée au fait que nous n'avons pris en compte qu'une seule solution du problème pour la deuxième phase de la description. Face à cette limite on peut choisir de prendre en compte plusieurs solutions et d'avoir des ensembles de types de tâches (et de praxéologies) qui seront spécifiques à chaque solution. L'étude des différences entre ces ensembles peut, en fixant un contexte d'utilisation de la ressource donnée, déterminer leur viabilité.
- La troisième concerne l'absence de prise en compte de l'énoncé utilisé pour communiquer le problème à l'apprenant sous forme d'exercice. Cet énoncé peut contenir des indications particulières (par exemple sur les

variables à déclarer, ce qui reviendrait à la prise en charge partielle du type de tâches *Modéliser*) ou des questions intermédiaires (par exemple donner une fonction à définir, ce qui reviendrait à une prise en charge partielle du type de tâches *Modulariser*). Cette prise en compte peut se réaliser lors d'une deuxième étude permettant de raffiner les types de tâches identifiés, par exemple en les caractérisant avec des indications du type *identifié dans l'énoncé*, *(partiellement) pris en charge par l'énoncé*, etc.

In fine, les choix à réaliser sur les limites identifiées ci-dessus sont essentiellement liés aux besoins de description des ressources. Par exemple, s'il s'agit de savoir si un code donné est pertinent comme exemple pour illustrer une notion, il n'est pas nécessaire d'envisager tous les codes solutions du problème. S'il s'agit de comparer des exercices entre eux il pourra être nécessaire de prendre en considération les éléments apportés par les énoncés et pas uniquement le problème à résoudre. S'il s'agit, pour un agent humain ou logiciel, de recommander des exercices à un apprenant, le niveau de description des types de tâches et les informations associées seront à adapter à la connaissance de l'apprenant (l'institution où il apprend ; son rapport aux types de tâches étudiés ; etc.).

4 Conclusion

Dans cette contribution nous avons présenté un référentiel de types de tâches d'apprentissage de la programmation en nous centrant sur les aspects liés aux algorithmes et aux programmes. Il enrichit significativement celui présenté dans [11] . La diversité des types de tâches identifiés, sans viser l'exhaustivité, montre l'ampleur de ce que veut dire *apprendre à programmer* et de ce que l'apprenant doit travailler. Et, par contraste, cela peut aussi mettre en évidence certains types de tâches qui ne sont pas traités lors de l'enseignement et laissés *de facto* à la charge complète de l'élève [7].

Le deuxième apport est une esquisse de méthode de description de problèmes de programmation. Les premières limites et perspectives associées à ce travail sont précisées en Sect. 3.4.

Du point de vue du référentiel, divers travaux sont engagés, ou en voie de l'être, pour répondre à certaines limites identifiées en Sect. 2. En particulier, améliorer la couverture avec l'intégration des types de tâches relatifs aux fonctions et aux listes et raffiner en spécifiant certains sous-types de tâches. Un autre axe est la construction de praxéologies complètes en liant types de tâches et techniques avec le logos. Une perspective liée à cette axe est la construction d'une ontologie comme évoqué dans [5] dans un autre domaine.

Pour ce qui est de la description des ressources, une suite envisageable concerne bien évidemment l'exploitation du référentiel étendu pour décrire une plus grande

7. Nous n'affirmons pas ici qu'il faut impérativement travailler tous ces types de tâches, simplement que la plupart d'entre eux vont sans doute être rencontrés à un moment ou un autre par les apprenants de la programmation.

diversité de ressources. Un autre axe consisterait à mettre en relation cette description avec d'autres caractéristiques comme la complexité d'une ressource [4], le niveau d'étayage proposé, ou son adéquation à une intention didactique. Cet axe offre d'autres perspectives d'utilisation de ce référentiel et de cette méthode de description qui pourraient aider un enseignant ou un EIAH dans leurs choix de ressources. Ce sujet a été partiellement exploré dans [10] autour de ressources en mathématiques, mais il s'agit maintenant de prendre en compte les spécificités des ressources pour l'enseignement de la programmation.

Références

1. Chaachoua, H., Bessot, A., Romo, A., Castela, C. : Developments and functionalities in the praxeological model. In : Bosch, M., Chevallard, Y., Javier Garcia, F., Monaghan, J. (eds.) Working with the anthropological theory of the didactic : A comprehensive casebook, pp. 41–60. New perspectives on research in mathematics education - ERME Series, Routledge, routledge edn. (2019)
2. Chevallard, Y. : Concepts fondamentaux de la didactique : perspectives apportées par une approche anthropologique. Recherche en didactique des mathématiques **12**(1), 83–121 (1992)
3. Diatta, B., Basse, A., Ouya, S. : PasOnto : Ontology for learning Pascal programming language. In : 2019 IEEE Global Engineering Education Conference (EDUCON). pp. 749–754. IEEE (2019)
4. Feitelson, D.G. : From Code Complexity Metrics to Program Comprehension. Communications of the ACM **66**(5), 52–61 (May 2023). https://doi.org/10.1145/3546576
5. Grugeon-Allys, B., Jolivet, S., Lesnes, E., Luengo, V., Yessad, A. : Mindmath : didactique des mathématiques et intelligence artificielle dans un EIAH. In : Vandebrouck, F., Emprin, F., Ouvrier-Buffet, C., Vivier, L. (eds.) Nouvelles perspectives en didactique des mathématiques : preuve modélisation et technologies numériques, vol. Volume des ateliers, actes de EE21, pp. 133–152. IREM de Paris - Université de Paris (2023)
6. Grévisse, C., Botev, J., Rothkugel, S. : An Extensible and Lightweight Modular Ontology for Programming Education. In : Solano, A., Ordoñez, H. (eds.) Advances in Computing, vol. 735, pp. 358–371. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-66562-7_26
7. Guzdial, M., Du Boulay, B. : The History of Computing Education Research. In : Fincher, S.A., Robins, A.V. (eds.) The Cambridge Handbook of Computing Education Research, pp. 11–39. Cambridge University Press, 1 edn. (Feb 2019). https://doi.org/10.1017/9781108654555.002
8. Hermans, F., Aldewereld, M. : Programming is Writing is Programming. In : Companion Proceedings of the 1st International Conference on the Art, Science, and Engineering of Programming. pp. 1–8. Programming '17, Association for Computing Machinery, New York, NY, USA (Apr 2017). https://doi.org/10.1145/3079368.3079413
9. Hoc, J.M. : Quelques remarques sur l'analyse du travail et la formation de l'analyste-programmeur. Bulletin de Psychologie **28**, 357–359 (1974)

10. Jolivet, S., Chaachoua, H., Desmoulins, C. : Modèle de description didactique d'exercices de mathématiques. Recherche en Didactique des Mathématiques **42**(1), 53–102 (2022)
11. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In : Actes du colloque EIAH2023 : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain. Brest (2023)
12. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Description et analyse de trois EIAH d'apprentissage de Python. In : Actes de l'atelier APIMU : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain. Brest (2023)
13. Loiseau, M. : Élaboration d'un modèle pour une base de textes indexée pédagogiquement pour l'enseignement des langues. Thèse de doctorat, Université Stendhal - Grenoble III, Grenoble (Dec 2009)
14. Pennington, N., Grabowski, B. : The tasks of programming. In : Psychology of programming, pp. 45–62. Elsevier (1990)
15. Pierrakeas, C., Solomou, G., Kameas, A. : An Ontology-Based Approach in Learning Programming Languages. In : 2012 16th Panhellenic Conference on Informatics. pp. 393–398. IEEE, Piraeus, Greece (Oct 2012). https://doi.org/10.1109/PCi.2012.78
16. Rogalski, J., Samurçay, R. : Acquisition of Programming Knowledge and Skills. In : Psychology of Programming, pp. 157–174. Elsevier (1990). https://doi.org/10.1016/B978-0-12-350772-3.50015-X
17. Shishehchi, S., Mat Zin, N.A., Abu Seman, E.A. : Ontology-Based Recommender System for a Learning Sequence in Programming Languages. International Journal of Emerging Technologies in Learning (iJET) **16**(12), 123 (Jun 2021). https://doi.org/10.3991/ijet.v16i12.21451
18. Sosnovsky, S., Gavrilova, T. : Development of educational ontology for C-programming. International Journal "Information Theories & Applications" **13**, 303–308 (2006), publisher : Institute of Information Theories and Applications FOI ITHEA