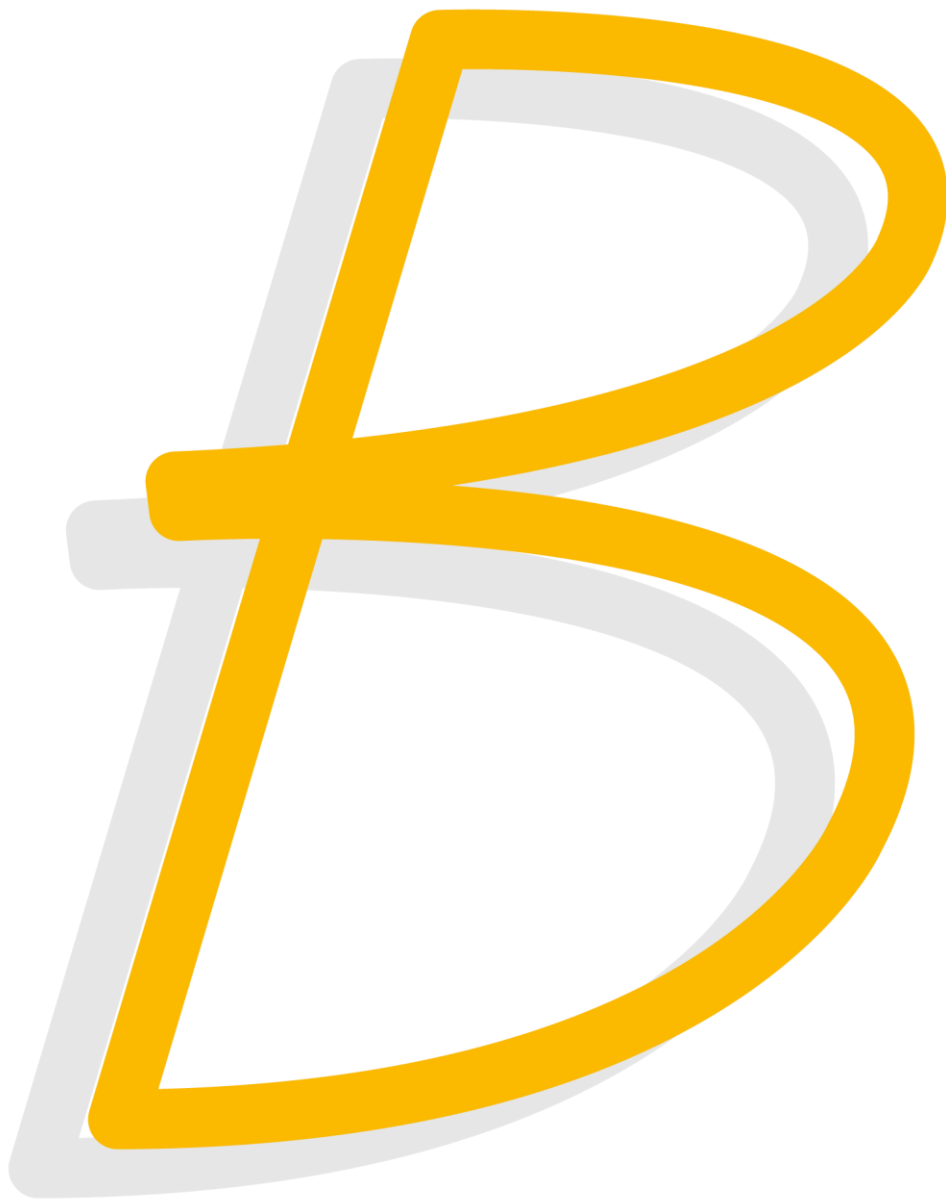


Atelier B Manuel Utilisateur

Atelier B
Manuel Utilisateur
Version 4.7

- Développement logiciel
- Modélisation système



Suivi des révisions

Édition	Date	Auteur	Commentaires
4.7	01/07/2021	M. Colange	Ajout des contraintes sécuritaires exportées, et mises à jour mineures.

Préambule

Ce document a pour but de présenter les différentes fonctionnalités offertes par l'Atelier B et d'initier progressivement le lecteur à son utilisation dans le cadre de projets de modélisation formelle. Il est structuré en trois parties principales :

- **Prise en main de l'Atelier B**

Présentation synthétique des principales fonctionnalités.

- **L'Atelier B en profondeur**

Présentation étendue de toutes les fonctionnalités.

- **Annexes techniques**

Description de fonctionnalités avancées.

Pour une présentation de la méthode B, nous renvoyons le lecteur à plusieurs références de qualité (voir §5.2 Bibliographie):

- *The B-Book : Assigning programs to meanings*
- *The B-Method : an introduction*
- *Program Development by Refinement*
- *Spécification formelle avec B*

Pour une présentation de la modélisation de système avec le B événementiel, nous renvoyons le lecteur à :

- *Modeling in Event-B : system and software engineering*

Il est à noter que le langage B supporté par l'Atelier B diffère en partie de celui décrit dans le *B-Book*. Par ailleurs certains exemples contenus dans ce document ne correspondent pas exactement à la syntaxe supportée par l'Atelier B. Nous invitons le lecteur à se reporter au Manuel de Référence du langage B.

De manière similaire, le langage Event-B supporté par l'Atelier B diffère sensiblement de celui présenté dans *Modeling in Event-B*. Nous invitons le lecteur à se reporter au Manuel de référence du langage B système

TABLE OF CONTENTS

1	INTRODUCTION Présentation, histoire	5
1.1	L'ATELIER B EN QUELQUES MOTS	6
1.2	UNE BREVE HISTOIRE DE L'ATELIER B	7
2	PRISE EN MAIN Installation, types de projet, interface, modélisation	11
2.1	INSTALLATION	12
2.2	LES TYPES DE PROJET	14
2.3	L'INTERFACE GRAPHIQUE	15
2.4	DEVELOPPER UN LOGICIEL	19
	2.4.1 Créer un projet	20
	2.4.2 Ouvrir un projet	21
	2.4.3 Ajouter un composant	21
	2.4.4 Editer un composant	21
	2.4.5 Contrôler le typage	21
	2.4.6 Générer les obligations de preuve	21
	2.4.7 Prouver automatiquement	22
	2.4.8 Prouver interactivement	23
	2.4.9 Contrôler les implémentations	23
	2.4.10	
	Traduire les implémentations	23
	2.4.11	
	Importer un projet	25
	2.4.12	
	Archiver un projet	25
2.5	MODELISER UN SYSTEME	26
3	L'ATELIER B EN PROFONDEUR	28
3.1	GESTION DE PROJET	29
	3.1.1 Paramétrage de l'Atelier B	29
	3.1.2 Gestion des utilisateurs	29
	3.1.3 Gestion des bibliothèques	29
	3.1.4 Gestion des répertoires de fichiers de définitions	29
	3.1.5 Générateur d'obligations de preuve	30
	3.1.6 Paramétrage de INT	30
	3.1.7 Vérification du typage des commandes de preuve	30
	3.1.8 Importation et exportation de projets	31
	3.1.9 Localisation de l'interface graphique	31
3.2	MODELISATION	31
	3.2.1 Vérification des règles de codage	31
	3.2.2 Raffinement automatique	32
	3.2.3 RECORDS	32
3.3	PREUVE	32
	3.3.1 Normalisation des prédicats	32
	3.3.2 GOP 4.1 et plus anciens	33

3.3.3	Preuve avec Pmm	34
3.3.4	Preuve interactive	34
3.3.5	Preuve par règles (version maintenance)	35
3.3.6	Le support du multi-cœur	35
3.3.7	Le serveur de preuve	35
3.3.8	Utilisation de prouveurs externes avec Why3	37
3.4	GENERATION DE CODE	37
3.4.1	Traducteur C	37
3.4.2	Traducteur HIA (version maintenance)	38
3.5	LANGAGE B SUPPORTE PAR L'ATELIER B	38
3.5.1	Support des entiers de grande taille	38
3.5.2	Support expérimental des nombres réels et flottants	38
3.5.3	Support des nombres littéraux en hexadécimal	39
3.6	LANGAGE EVENT-B SUPPORTE PAR L'ATELIER B	40
4	Safety Related Application Conditions	
	Erreur ! Signet non défini.	
5	ANNEXES TECHNIQUES Mode batch, commandes preuve interactive, obligations de preuves	43
5.1	MODE BATCH	44
5.2	LISTE DES COMMANDE INTERACTIVES DE PREUVE	45
5.3	OBLIGATIONS DE PREUVE B	45
5.4	OBLIGATIONS DE PREUVE EVENT-B	45
5.5	GENERATEUR D'OBLIGATIONS DE PREUVE 4.2+	46
6	Références Glossaire, bibliographie, liens utiles	47
6.1	GLOSSAIRE	48
6.2	BIBLIOGRAPHIE	48

01.

1 INTRODUCTION PRESENTATION, HISTOIRE

1.1 L'ATELIER B EN QUELQUES MOTS

Développé par la société ClearSy¹, l'Atelier B est l'outil industriel qui permet une utilisation opérationnelle de la méthode formelle B pour des développements de logiciels prouvés sans défaut. B est une méthode de spécification formelle qui autorise l'expression rigoureuse de propriétés et la description du comportement d'un logiciel, de manière modulaire et progressive, les spécifications pouvant être abstraites. Il est alors possible de prouver de manière automatisée que :

- ces propriétés sont cohérentes et non contradictoires.
- la spécification des fonctions du logiciel vérifie ces propriétés

La garantie que ces propriétés sont respectées au fur et à mesure des étapes de conception est apportée par la preuve mathématique. Le logiciel obtenu est alors considéré sans défaut. L'Atelier B est disponible en deux versions:

- une version communautaire accessible et utilisable par tous. Elle est complètement fonctionnelle et est fournie sans support avec une périodicité d'environ 2 ans. Elle inclut un générateur de code C.
- une version maintenue accessible aux possesseurs d'un contrat de maintenance. Elle est mise à jour une à deux fois par année. Elle inclut toutes les fonctionnalités de la version communautaire ainsi que les traducteurs Ada, C++ et les outils de preuve de règles.
- Une version CSSP accessible aux possesseurs de starter kits SK0 ou SK1

Fonctionnalité	Atelier B 4.2 Community Edition	Atelier B 4.2 Maintenance Edition
Environnement de développement	✓	✓
Support projet langage B	✓	✓
Support projet langage Event-B	✓	✓
Support projet validation de données	✓	✓
Editeur de modèles B et Event-B	✓	✓
Raffineur Automatique	✓	✓
Vérificateur de type	✓	✓
Générateur d'obligations de preuve	✓	✓
Prouveur Automatique	✓	✓
Prouveur interactif	✓	✓
Prouveur de prédicats	✓	✓
Traducteur C C4B	✓	✓
Traducteur Ada (MacOS, Linux)		✓
Traducteur High Integrity Ada (MacOS, Linux)		✓
Traducteur C++ (MacOS, Linux)		✓
Outil de validation de règles mathématiques		✓

- FIGURE 1.1 -
Comparaison des fonctionnalités des deux versions de l'Atelier B

L'Atelier B est notamment utilisé pour :

- le développement logiciel d'automatismes sécuritaires ferroviaires
- la certification² de composants microélectroniques avec B événementiel
-

1- <http://www.clearsy.com>

2- selon la norme Critères Communs (2.2 et 3.1)

Ces deux usages donnent lieu à deux types de projet dans l'Atelier B : développement

logiciel, modélisation de système et validation formelle de données.

1.2 UNE BREVE HISTOIRE DE L'ATELIER B

GENESE

L'Atelier B est à l'origine un outil interne Alstom permettant de mettre en œuvre la méthode B pour le développement de logiciels sécuritaires. Dans le cadre de la création du premier métro parisien sans conducteur, le consortium RATP/SNCF/INRETS organise l'industrialisation de l'outil afin qu'il soit qualifié et utilisé pour le développement des automatismes sécuritaires de la ligne 14. Les personnels-clé en charge des travaux de réingénierie et de développement réalisés par une PME d'Aix en Provence, créeront ClearSy en 2001 qui deviendra propriétaire intégrale de l'Atelier B et s'occupera des développements de l'outil et de la diffusion de la méthode.

TECHNOLOGIE

Le premier Atelier B commercial, en version 3.2, est né en 1994. Il est développé sous Unix (Sun Solaris, HP/UX et Linux) et utilise X-Motif pour son interface graphique, ce qui restera vrai jusqu'à la version 3.7 en 2008. La version majeure, utilisée pour Météor et mise en service en 1998, est la 3.6. En 2009, la version 4.0 apporte deux innovations :

- l'utilisation du framework Qt qui permet un portage sous Windows et MacOS (Solaris et HP/UX sont abandonnés),
- la mise à disposition d'une version communautaire gratuite et fonctionnelle en plus de la version maintenance.

Une majorité des outils au cœur de l'Atelier B originel sont développés en langage de théorie, langage proche de Prolog inventé par Jean-Raymond Abrial et permettant une manipulation aisée de modèles B. La plupart de ces outils ont été aujourd'hui redéveloppés en C++/Qt, à l'exception des prouveurs.

GENERATION DE CODE

L'Atelier B 3.2 est équipé de 3 traducteurs (C/C++, Ada, High Integrity Ada) génériques. Le traducteur C/C++ sera remplacé en 2011 par un traducteur C, ComenC, produisant un code plus lisible mais ne supportant pas l'instanciation multiple de machines B pour limiter l'usage de pointeurs. La technologie sous-jacente étant devenue obsolète, un nouveau générateur de code C, C4B, sera ensuite développé à partir de Coco/R.

Pour les besoins de l'automatisation de la ligne 3, un générateur de code Ladder, ciblant des automates programmables, sera développé mais il ne sera pas ni diffusé ni commercialisé. Il sera ensuite remplacé par un générateur de code Instruction List, ciblant lui aussi des automates programmables et utilisé pour le métro de Paris L13 et Sao Paulo L2 & L9.

Dans le cadre d'un projet de R&D industriel, un générateur de code VHDL, B4SYN, sera développé en partenariat avec STMicroelectronics. Il n'est à ce jour ni diffusé ni commercialisé.

Pour les besoins de l'automatisation de la ligne 15 du métro de Sao Paulo et de Stockholm Citybanan, un générateur de code binaire PIC32 (format Hex) a été développé. Il n'est actuellement ni diffusé ni commercialisé.

Il existe enfin un traducteur vers MISP32, appelé b32, utilisé dans la double chaîne de compilation de la ClearSy Safety Platform.

OUTILLAGE COMPLEMENTAIRE

Un outil de raffinement automatique, BART, similaire à celui développé et utilisé par Siemens a été développé et intégré à l'Atelier B en 2006. Il a été utilisé en 2007 pour

développer le logiciel de pilotage de la navette automatique de l'aéroport Charles de Gaulle (180 kloc) et plus récemment pour un compilateur xml pour l'ingénierie du métro de Lille (300 kloc).

L'animateur de modèle, PredicateB, initialement développé pour la plateforme Rodin en 2007, a été reconçu et développé en Qt/C++ pour être intégré à des logiciels d'ingénierie propriétaires (outils de validation formelle de données ferroviaires, simulateur d'algorithmes de pilotage de métro). Il n'est actuellement pas diffusé.

Depuis sa version 4.2, l'AtelierB intègre à titre expérimental un nouveau générateur d'obligations de preuve, baptisé NGOP. Il a pour vocation de remplacer le générateur d'obligations de preuve historique, et propose de nouvelles fonctionnalités, notamment la traçabilité entre obligations de preuve et éléments du modèle B, et la possibilité de s'interfacer avec des prouveurs externes, tels que proB ou des prouveurs SMT.

L'atelierB propose également (en édition Maintenance) un outil de preuve de règles de preuve, nommé OPR.

- La distribution des tâches du générateur d'obligations de preuve et du prouveur sur plusieurs cœurs et sur plusieurs ordinateurs

Tâches					
Projet	Composant	Action	Statut	Messages	Serveur
DCGe	crc_i		En cours	End of Proof	localhost-2
DCGe	donnees_i		En cours	clause Initialisation - Prouvé 2, Non Prouvé 0, Essayé 2/39, Fin estim...	localhost-4
DCGe	itf		En cours	End of Proof	localhost
DCGe	operateurs		En cours	clause WellDefinednessProperties - Prouvé 1, Non Prouvé 0, Essayé 1...	localhost-3

Figure 1: exécution de la preuve d'un composant en parallèle sur 4 cœurs

- Le prouveur interactif évolué (choix de la branche de preuve suivante, copier/coller dans l'arbre de preuve, automatisation d'enchainements de certaines commandes, lien entre les obligations de preuve et la modélisation)
- L'éditeur intégré avec coloration syntaxique, navigation, complétion, mise en évidence des erreurs, intégration de la preuve avec mise en évidence des lignes de modèles prouvées ou non)

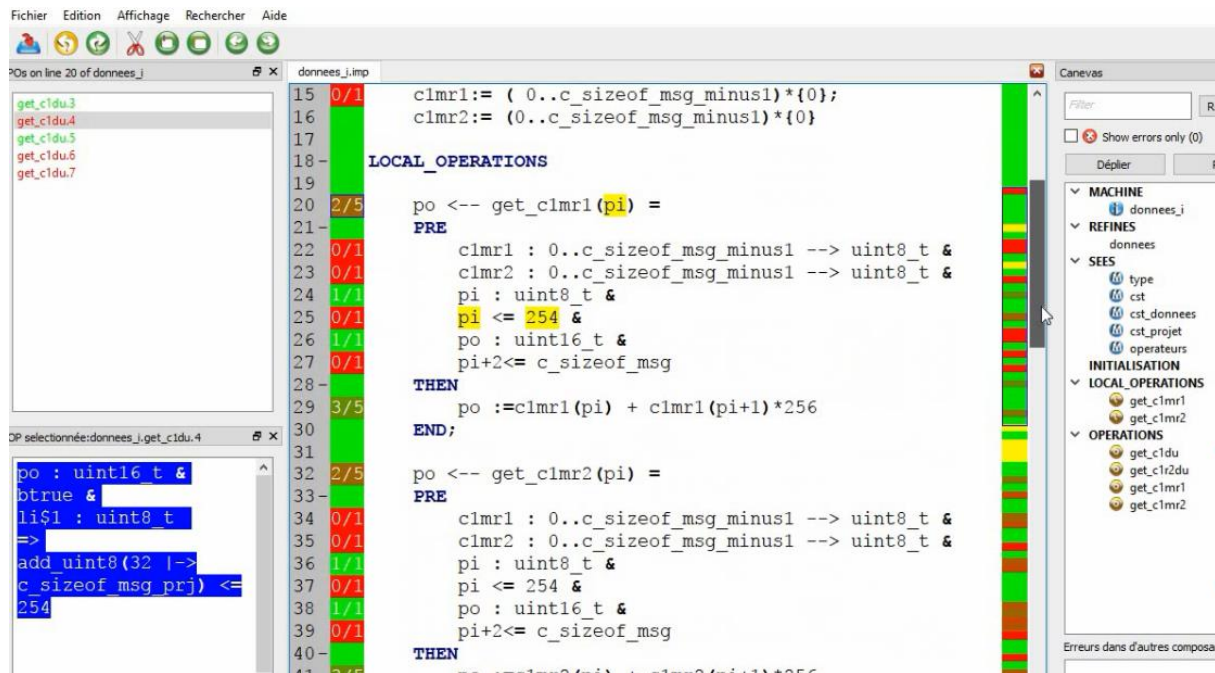
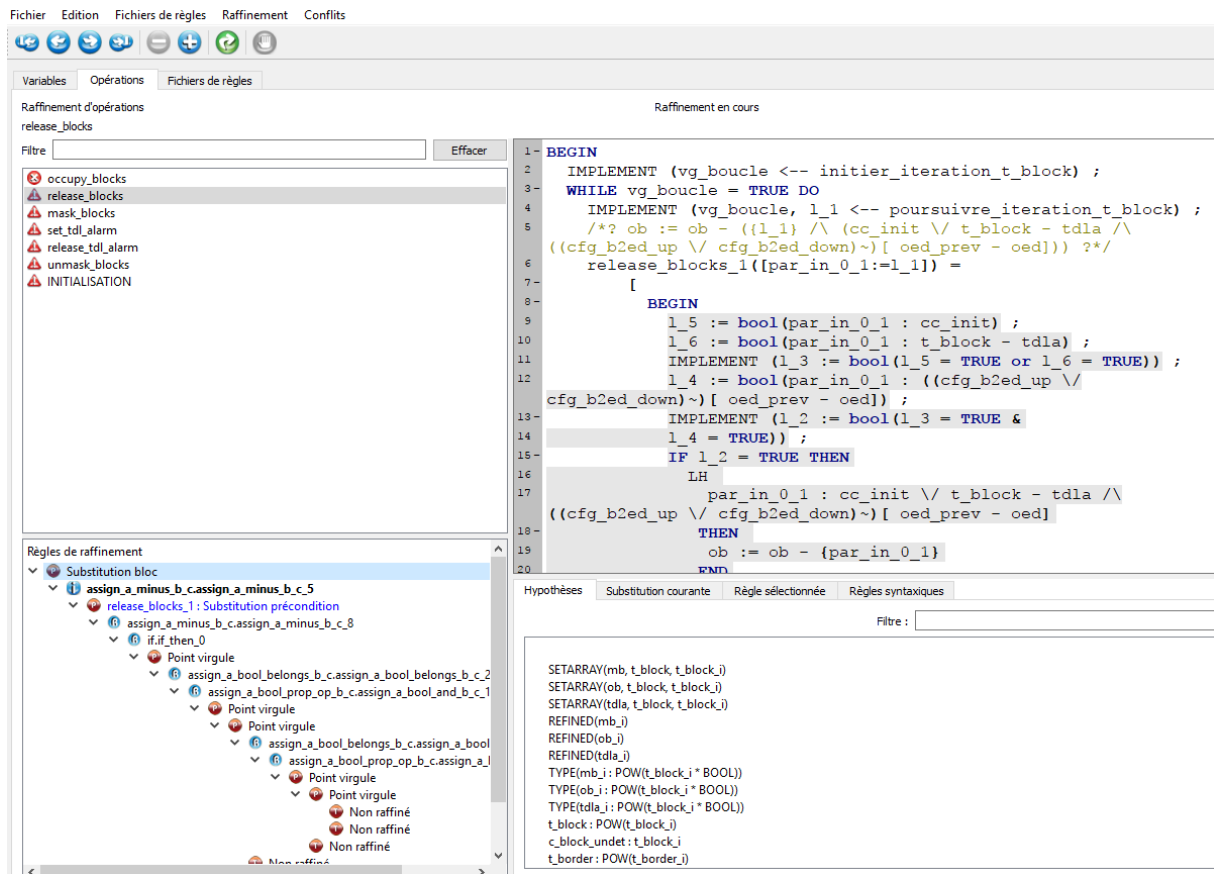


Figure 2: éditeur de modèle intégrant l'information de preuve (nombre d'obligations de preuve et taux de preuve par ligne

- Les outils de preuve de règles, qui permettent de vérifier les règles mathématiques ajoutées dans un cadre industriel
- Le raffinement automatique avec l'ajout de l'outil BART supportant un formalisme proche de celui d'Edith B et un mode de fonctionnement similaire.



- FIGURE 2.1.1 -
interface de raffinement interactif avec BART

- La bonne définition des modèles. Les obligations de preuve de bonne définition sont gérées directement au travers du projet B.
- L'Interfaçage avec des outils externes au travers de points d'extension qui permettent d'ajouter des fonctionnalités (menus) à l'Atelier B sans recompilation.
- Le nouveau GOP qui permet de rendre accessible l'information de traçabilité entre un modèle et ses obligations de preuve. Par ailleurs, les obligations de preuve sont décrites sous la forme d'un fichier texte structuré, ce qui permet un découplage entre le moteur de génération et les obligations de preuve théoriques (avec la possibilité de produire d'autres obligations de preuve si nécessaire). Les deux GOP (version historique 3.6.4, version 4.4.2) peuvent être utilisés indifféremment sur un projet. Toutefois il est à noter que les obligations de preuve générées diffèrent en nombre et en forme, entre ces deux versions. Il est aussi à noter que la plupart des fonctionnalités présentées ci-dessus nécessitent l'utilisation du nouveau GOP.

NOUVEAUX PARADIGMES

Pour des besoins fonctionnels et de performances non atteints par la plateforme Rodin en 2007, le support du B événementiel est introduit afin de pouvoir compléter des travaux de modélisation de composants micro-électroniques. En 2013, ce support permettait de démontrer la sûreté des principes de signalisation de la ligne 7 du métro de New York.

02.

2 PRISE EN MAIN

INSTALLATION, TYPES DE PROJET, INTERFACE, MODELISATION

2.1 INSTALLATION

L'Atelier B se télécharge depuis le site <http://www.atelierb.eu>.

La version **Community** se télécharge depuis la section Téléchargements. Les plateformes supportées sont Windows, Linux Debian 64 bits, Linux Debian et MacOS. Seule la dernière version est disponible, sans support. La version Community est mise à jour environ tous les 2 ans.

La version **Maintenance** s'installe de la même manière que la version Community mais nécessite un accès à un espace dédié aux possesseurs d'un contrat de maintenance. La version Maintenance est mise à jour jusqu'à 2 fois par an.

N'importe quel ordinateur moderne fonctionnant sous Windows, Linus ou MacOS est capable d'exécuter l'Atelier B.

Il est conseillé de disposer d'au moins 2 Go de mémoire RAM. Toutefois, l'espace mémoire réellement nécessaire est fortement dépendant de la taille des développements effectués avec l'Atelier B.

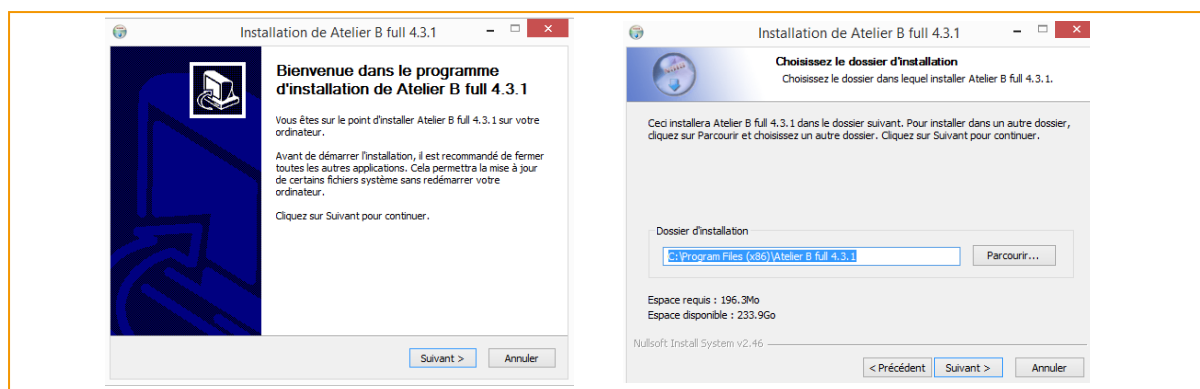
Pour installer l'Atelier B, vous avez besoin d'environ 250 Mo d'espace disque.

L'espace disque occupé par un projet développé avec l'Atelier B est dépendant de la taille des fichiers sources B. L'espace disque occupé par les fichiers générés automatiquement par l'Atelier B est égal à environ 25 fois la taille de l'espace disque de tous les fichiers sources B.

Enfin, dans le cadre de la mise en œuvre du serveur de preuves (voir §3.3.7), il est nécessaire d'installer un client SSH comme OpenSSH ou la distribution Putty.

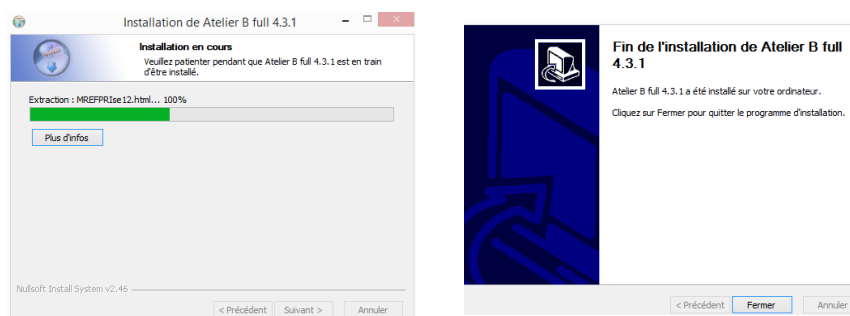
INSTALLATION DE LA VERSION WINDOWS

Une fois l'installateur téléchargé, il suffit de l'exécuter, puis d'indiquer le répertoire d'installation. Le choix par défaut est le répertoire c:\Program Files (X86).



- FIGURE 2.1.1 -
Ecran d'accueil de l'installateur et choix du répertoire d'installation

Si une version antérieure de l'Atelier B était installée, il est inutile de la désinstaller au préalable car chaque version de l'Atelier B s'installe dans un répertoire différent dont le nom se termine par le numéro de version.



- FIGURE 2.1.2 -
Installation et écran final

Par ailleurs, il conviendra d'éviter d'installer l'Atelier B sur un compte dont le nom contient des caractères spéciaux et accentués.

INSTALLATION DE LA VERSION LINUX

Une fois le paquet téléchargé (au format rpm ou au format deb, 64 bits), il suffit de l'installer :

- `dpkg -i atelierb-<full ou free>-<version>.deb` pour la version Debian
- `rpm -i atelierb-<full ou free>-<version>.rpm` pour la version Redhat (version spécifique)

L'Atelier B s'installe dans le répertoire `/opt/atelierb-<version>`.

Pour lancer l'Atelier B, il faut alors taper la commande

`/opt/atelierb-<version>/startAB` pour la version 64 bits
`/opt/atelierb-<version>/startAB` pour la version 32 bits

```
tlecomte@tlecomte-VirtualBox:~$ sudo dpkg -i /mnt/SHARE/atelierb-full-4.3.1-linux_x64.deb
Selecting previously unselected package atelierb-full-4.3.
(Reading database ... 172127 files and directories currently installed.)
Preparing to unpack .../atelierb-full-4.3.1-linux_x64.deb ...
Unpacking atelierb-full-4.3 (4.3.1) ...
Setting up atelierb-full-4.3 (4.3.1) ...
```

Il est recommandé de désinstaller la version courante de l'Atelier B avant d'installer une nouvelle version, si le numéro de version mineur ou majeure change. S'il s'agit d'une version de maintenance (installation de la version 4.3.1 par-dessus la version 4.3), il n'est pas nécessaire de procéder à la désinstallation. Pour la désinstallation, on utilisera la commande :

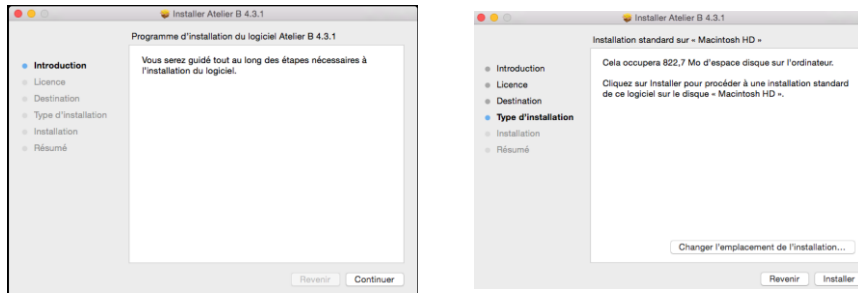
- `dpkg -r atelierb-<full ou free>-<version>` pour la version Debian
- `rpm -e atelierb-<full ou free>-<version>` pour la version Redhat (version spécifique)

INSTALLATION DE LA VERSION ZIPPEE

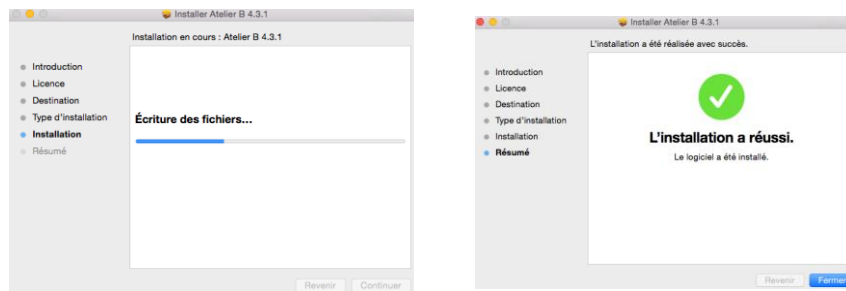
L'atelierB est également distribué sous forme d'archive contenant les exécutables Linux. Il suffit d'extraire le contenu de l'archive à l'emplacement de son choix.

INSTALLATION DE LA VERSION MACOS

Une fois l'installateur téléchargé, il suffit de l'exécuter, puis d'indiquer le répertoire d'installation.



- FIGURE 2.1.5 -
Ecran d'accueil de l'installateur et choix du répertoire d'installation



- FIGURE 2.1.6 -
Installation et écran final

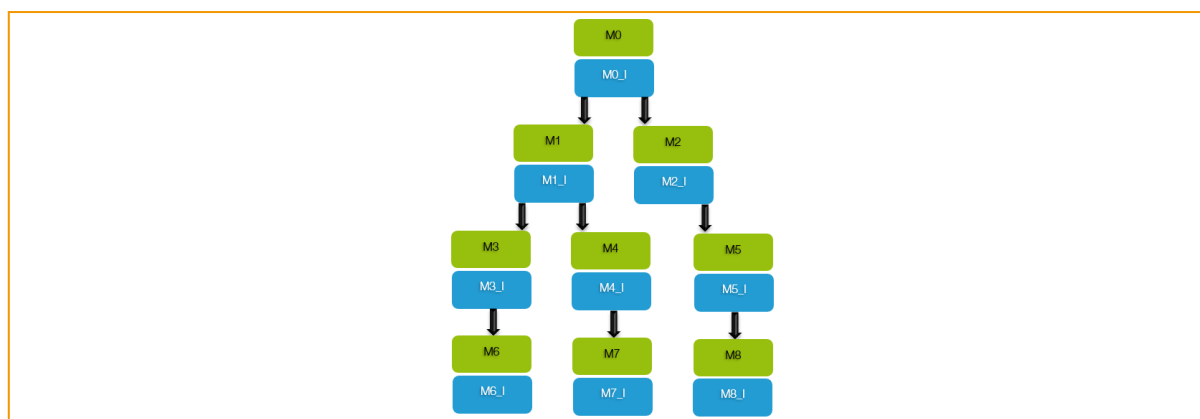
2.2 LES TYPES DE PROJET

L'Atelier B permet de créer 2 types de projet :

- **Développement logiciel.** Un projet de développement logiciel permet de construire le modèle d'un logiciel séquentiel. Le modèle contient la spécification et la conception des fonctions réalisées par le logiciel, spécification et conception étant distinctes. La conception peut alors faire apparaître le séquençement de fonctions provenant de composants importés, spécifiés et conçus de la même manière.

Le graphe de dépendance d'un projet logiciel est un arbre dont la racine est le composant de plus haut niveau (voir figure 2.2.1). Le logiciel est obtenu en traduisant chaque composant de type implémentation en code source.

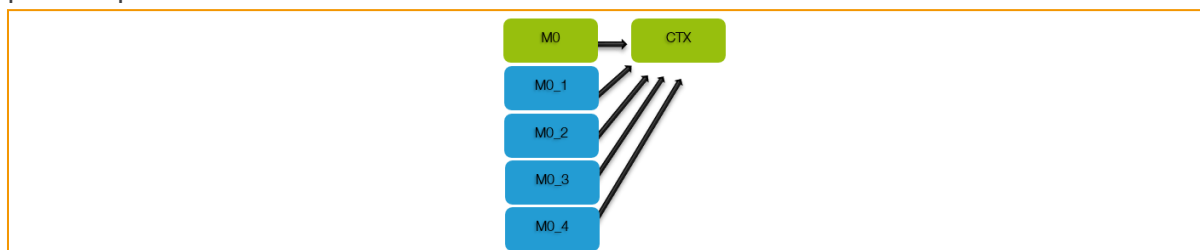
Le cycle de développement d'un projet logiciel est détaillé en figure 2.4.1. Il est composé de deux branches : l'une pour la génération de code, l'autre pour la preuve du modèle. Il est possible de générer du code sans que le modèle soit prouvé mais il n'y a alors aucune garantie que le logiciel soit sans défaut. Il est par ailleurs possible d'utiliser des spécifications muettes (une opération spécifiée par skip) : la portée de la preuve s'en trouve dans ce cas considérablement réduite, le recours à cette technique doit alors être justifié et encadré.



- FIGURE 2.2.1 -

Exemple d'arborescence d'un projet B logiciel. Les composants verts sont les spécifications (MACHINE), les composants bleus représentent les implémentations (IMPLEMENTATION), les flèches figurent les liens d'importation (IMPORTS).

- **Modélisation système.** Un projet de modélisation système permet de construire le modèle d'un système, au sens le plus large du terme (par exemple un avion, un passage à niveau, un processeur, une procédure administrative). Le modèle représente les variables d'état du système et les événements atomiques qui modifient, l'un après l'autre, son état. Ce modèle implique une « exécution » asynchrone et non-déterministe de ses événements. N'importe quel système peut alors être modélisé, sa spécification peut être équipée de propriétés puis être prouvée qu'elle correspond à ces propriétés. Le graphe de dépendance d'un projet de modélisation système est une colonne de raffinement qui contient une machine abstraite à la racine et éventuellement une succession de raffinements. Un modèle système ne contient pas d'implantation.



- FIGURE 2.2.2 -

Exemple d'arborescence d'un projet Event-B. Les composants verts sont les spécifications (SYSTEM), les composants bleus représentent les raffinements (REFINEMENT), les flèches figurent les liens de visibilité (SEES).

2.3 L'INTERFACE GRAPHIQUE

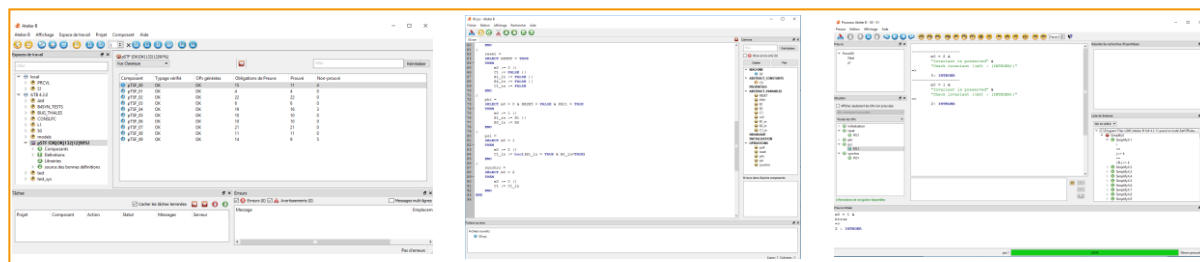
L'Atelier B est utilisable en :

- **Mode interactif** : au travers d'une interface graphique conçue pour pouvoir appliquer de manière efficace la méthode B, c'est-à-dire, spécifier, raffiner, prouver et générer du code. Le présent document couvre majoritairement ce mode d'utilisation.
- **Mode programmé** : reposant sur un langage de commande. Ce mode permet une utilisation

scriptée et est décrit en annexe (voir §4.1).

Le mode interactif offre des fonctionnalités nécessaires au développement de modèles formels :

- La gestion de projet, au travers de la vue composants ;
- L'édition des modèles, au travers de la vue éditeur ;
- La démonstration mathématique des modèles, au travers de la vue preuve.



- FIGURE 2.3.1 -

Les trois principales vues de l'Atelier B (de gauche à droite) : la vue composants, la vue éditeur et la vue preuve.

VUE COMPOSANTS

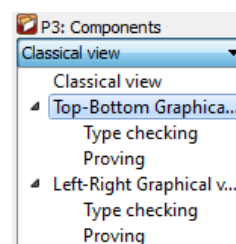
La vue composants de l'Atelier B liste les différents composants d'un projet, leur état (vérification du typage, obligations de preuve générées, capacité à être traduit en code source) ainsi que le nombre d'obligations de preuve prouvées et à prouver. Cette vue est centrale et permet d'avoir une vue d'ensemble du projet et des tâches en cours d'exécution.

Composant	Typage vérifié	OPs générées	Obligations de Preuve	Prouvé	Non-prouvé	BO Vérifié
dcd_cst	OK	2	0	0	2	-
dcd_cst_crc	OK	0	0	0	0	-
dcd_cst_donnees	OK	0	0	0	0	-
dcd_cst_type	OK	3	0	3	-	-
dcd_cst_type_i	OK	8	0	8	-	-
dcd_donnees	OK	1	0	1	-	-
dcd_donnees_i	OK	5	0	5	OK	-
dcd_id	OK	2	0	2	-	-
dcd_message_coeur_distant	OK	0	0	0	-	-
dcd_message_coeur_distant_i	OK	14	0	14	OK	-
dcd_operateurs	OK	0	0	0	-	-
dcd_type	OK	0	0	0	-	-
dcd_util	OK	8	0	8	-	-
dcd_util_i	OK	32	0	32	OK	-
dcd_verif_ALU	OK	4	0	4	-	-
dcd_verif_ALU_1	OK	5	0	5	-	-
dcd_verif_ALU_1_2r	OK	153	0	153	-	-
dcd_verif_ALU_1_3r	OK	109	0	109	-	-

- FIGURE 2.3.2 -

Vue composants (classique) d'un projet logiciel. Les composants sont classés par ordre alphabétique. Les spécifications (Machine) sont précédées de l'icône M. Les implémentations sont précédées de l'icône I.

La vue composants se décline en plusieurs type de représentation : tabulaire (classique) ou graphiques (afin de pouvoir apprécier certaines valeurs d'un seul coup d'œil). Le choix de la vue affichée est déterminé par le menu ci-contre. On notera que la vue graphique peut être paramétrée pour un affichage du « haut vers le bas » ou de « la gauche vers la droite ».

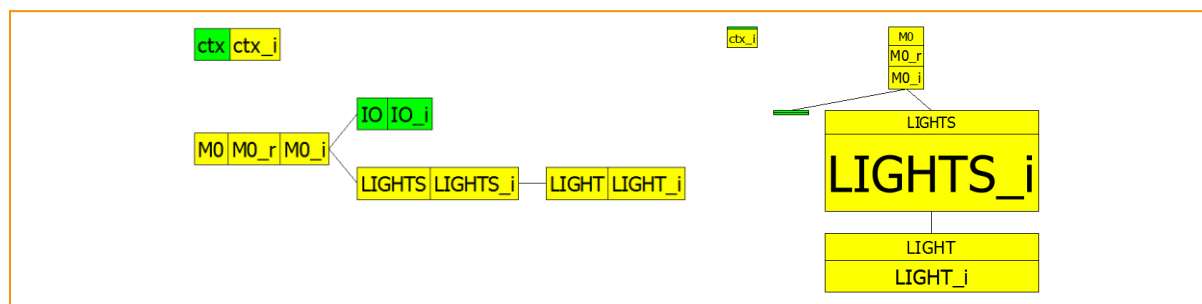


La vue graphique simple permet d'afficher l'état de preuve d'un projet. Les composants sont

affichés selon leur ordre de dépendance.
Leur couleur indique leur état de preuve :

- **vert** : un taux de preuve de 100%,
- **un dégradé du jaune au rouge** pour des taux de preuve intermédiaires.

Les vues « typecheck » et « proving » permettent respectivement de dimensionner les boîtes des composants en fonction du nombre d'obligations de preuve. Plus une boîte est volumineuse, plus le nombre d'obligations de preuve est important. Combiné avec la colorisation en fonction du taux de preuve, il devient possible d'apprécier rapidement l'effort de preuve restant à mener.

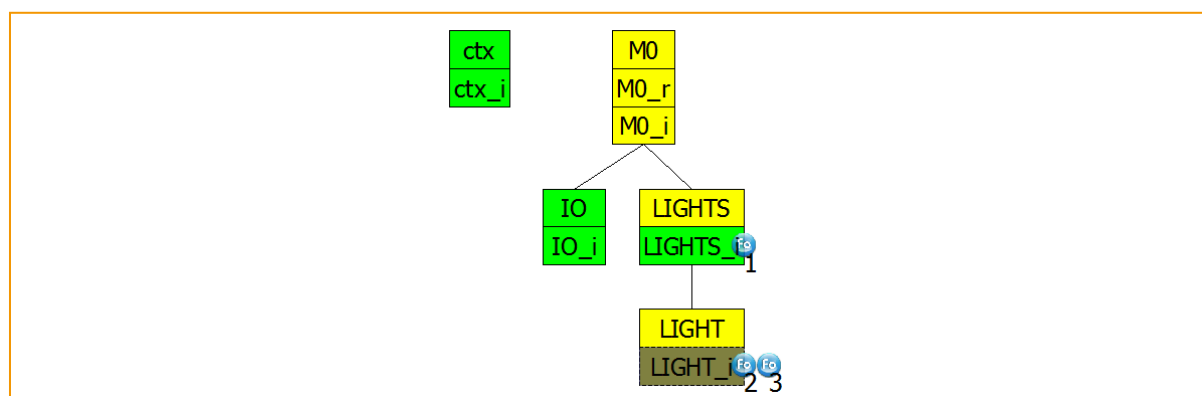


- FIGURE 2.3.3 -

Vue graphique simple (gauche) et vue « typecheck/proving » (droite). La coloration indique le taux de preuve (un vert franc indique un composant complètement prouvé) et la taille (droite) indique en proportion le nombre d'obligations de preuve.

La vue graphique permet enfin de représenter la liste des tâches associées aux composants. Il est en effet possible de lancer plusieurs tâches en parallèle sur un même composant. Il faut pour cela :

- **avoir modifié la ressource** « maximum running tasks » avec une valeur supérieure à 1 (menu préférences / installation) ;
- sélectionner un ou plusieurs composants et **actionner plusieurs fois les boutons** PO, TC, F0 ou F1.



- FIGURE 2.3.3 -

La vue graphique sera mise à jour en fonction des tâches en cours d'exécution.

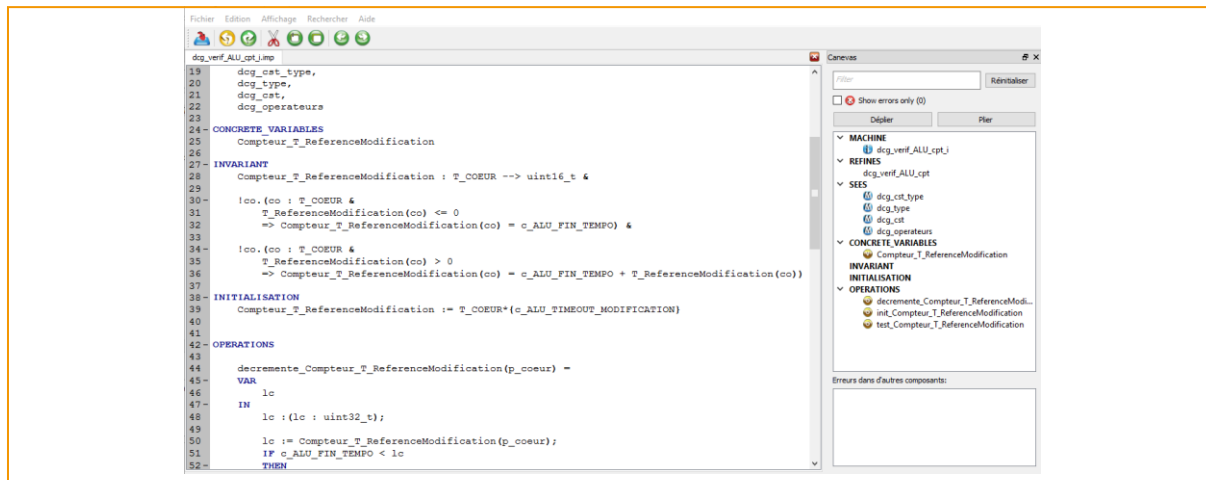
La liste des tâches en cours s'affichera dans la liste des tâches. La vue graphique sera mise à jour en conséquence. Lorsqu'il y a plus de tâches demandées que de tâches exécutables, celles-là seront mises en attente et auront un numéro d'ordre qui apparaîtra sur la vue graphique.

Si la case à cocher « Cacher les tâches terminées » est cochée, seules les tâches en cours d'exécution sont affichées. Sinon l'historique (le contenu de la fenêtre « tâches ») est affiché.

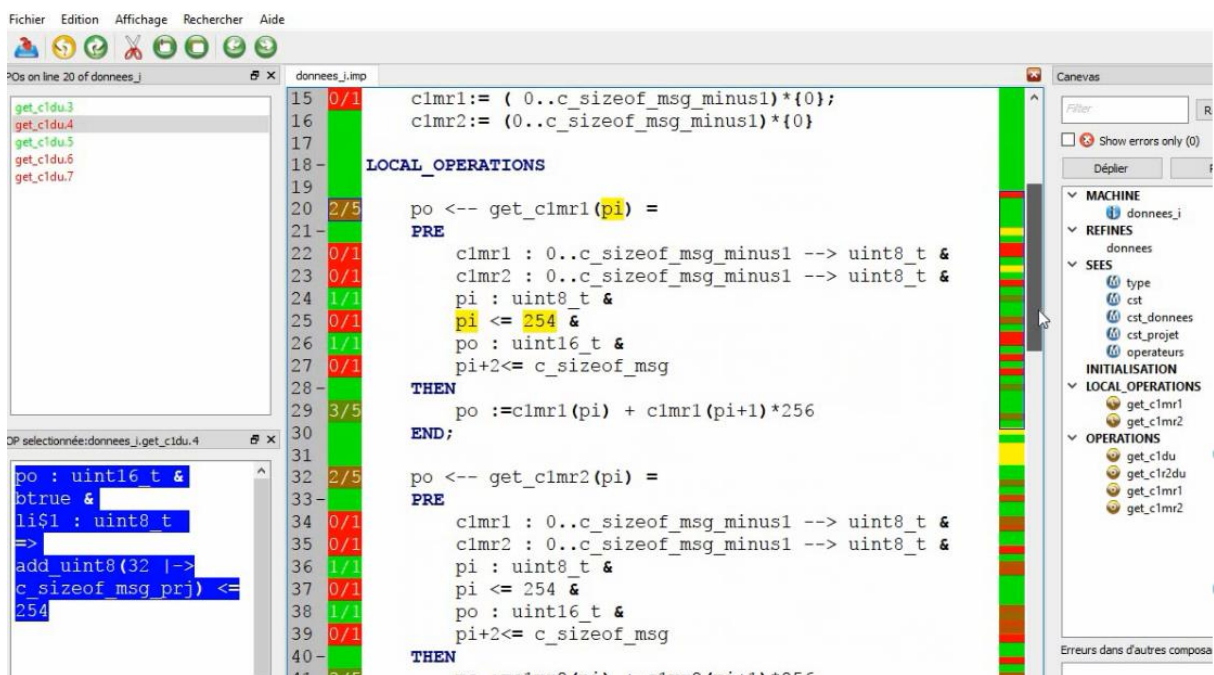
VUE EDETEUR

Cette vue permet de saisir :

- des modèles formels (composant B),
- des règles et des démonstrations mathématiques pour la phase de preuve (fichiers pmm),
- des règles de raffinement pour la phase de raffinement automatique (fichiers rmf).

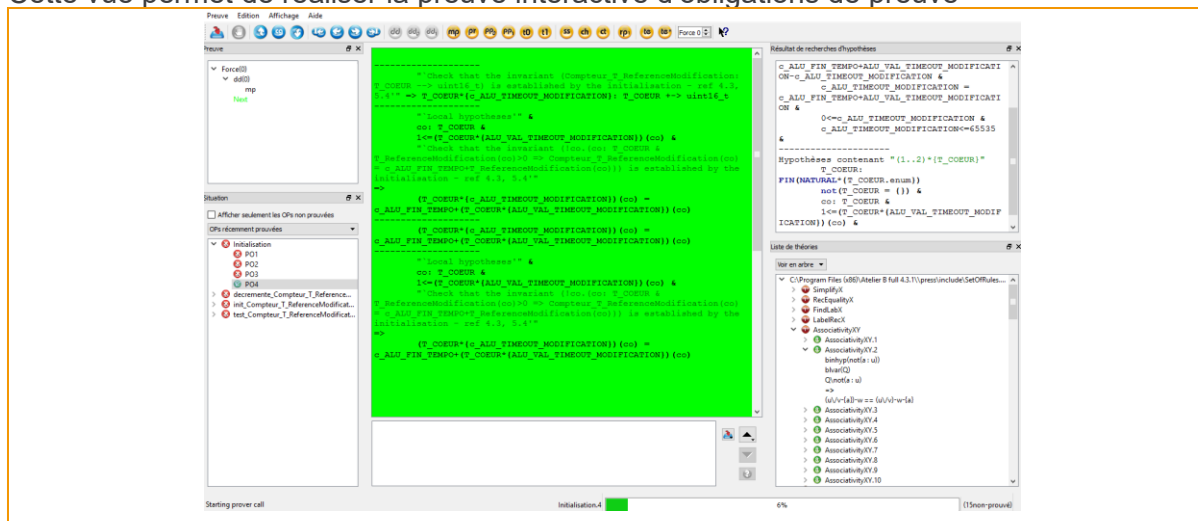


L'édition de composants B s'accompagne de services tels que l'affichage de la structure du modèle (outline), la navigation, la complétion, l'affichage et la localisation des erreurs, l'indentation automatique, la correction orthographique des commentaires, intégration de la preuve avec mise en évidence des lignes de modèles prouvées ou non)



VUE PREUVE

Cette vue permet de réaliser la preuve interactive d'obligations de preuve



2.4 DEVELOPPER UN LOGICIEL

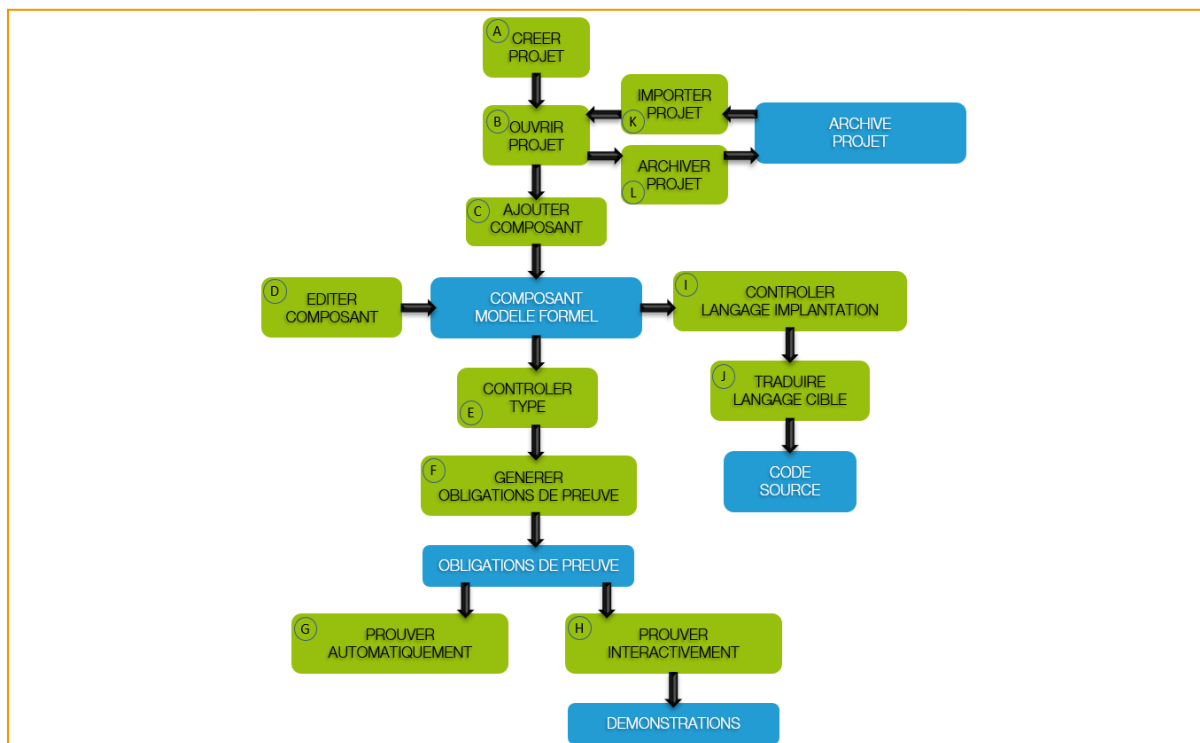
Le développement d'un logiciel avec B a pour objectif de fournir un logiciel prouvé par rapport à ses spécifications. La figure 2.4.1 présente le séquençement des activités à mener (les actions sont représentées par des rectangles verts, les données par des rectangles bleus).

La **structuration** d'un projet et la **création du contenu** (modèles) sont réalisées par les étapes A, B, C, D, K et L.

L'activité de **preuve** est couverte par les étapes E, F, G et H.

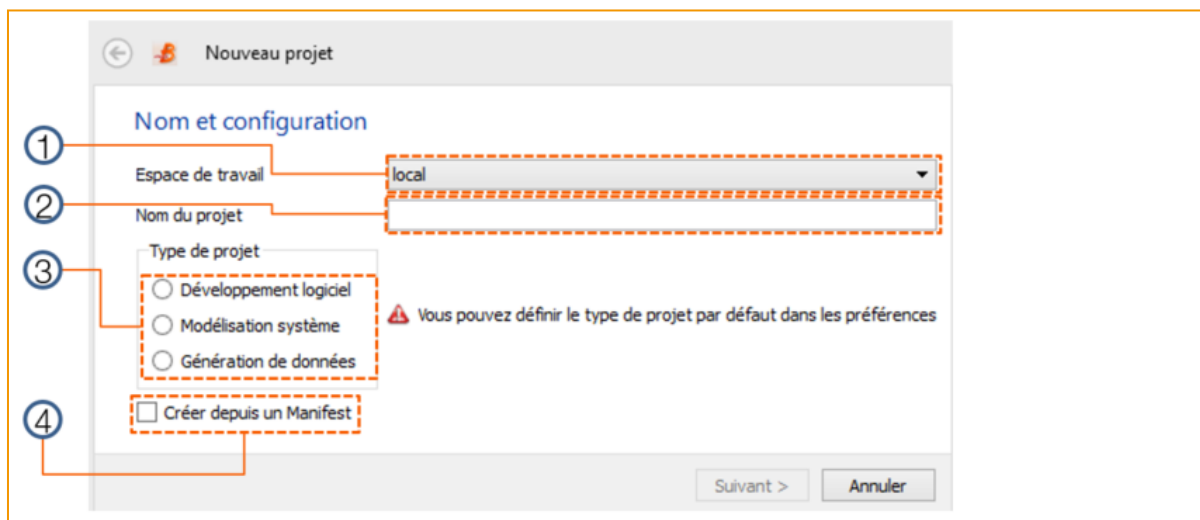
L'activité de **génération de code** à partir des modèles formels est couverte par les étapes I et J.

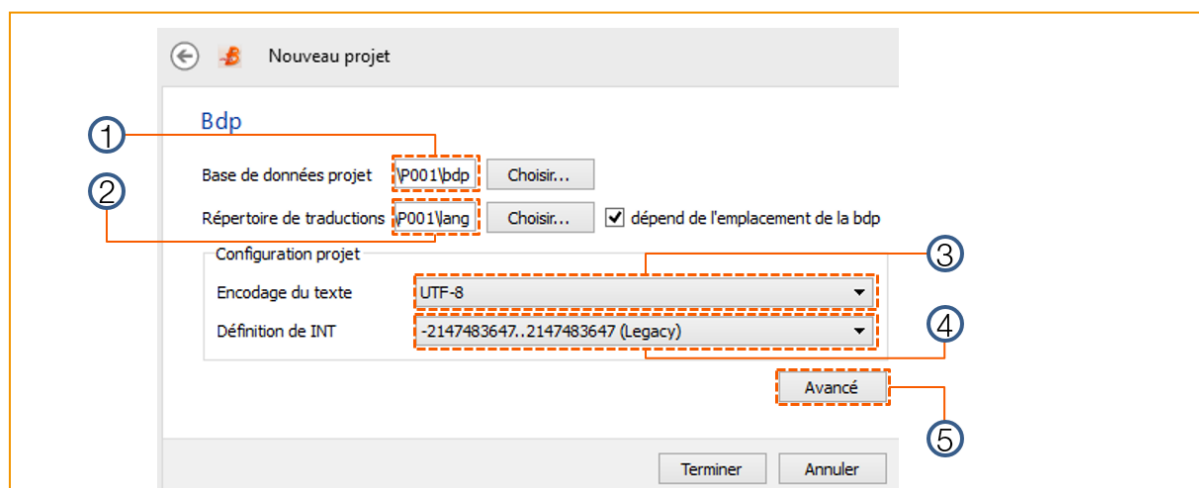
Il n'est pas nécessaire de compléter l'activité de preuve pour pouvoir générer du code. On perd par contre tout l'intérêt de B en menant pas la preuve à son terme.



- FIGURE 2.4.1 -
Le cycle de développement d'un projet de développement logiciel

2.4.1 Créer un projet





2.4.2 Ouvrir un projet

L'ouverture d'un projet se fait simplement par double click sur le projet dans l'arborescence des projets (à gauche de la fenêtre principale). Une fenêtre invite alors l'utilisateur à renseigner le type de projet, son nom et son emplacement sur le disque.

2.4.3 Ajouter un composant

L'ajout d'un nouveau composant se fait par clic-droit sur le projet et en sélectionnant « Nouveau composant » dans le menu contextuel. Une fenêtre invite alors l'utilisateur à renseigner le type du composant (machine, raffinement, implémentation), son nom, ainsi que son emplacement sur le disque. Cette fenêtre présente également un aperçu du contenu du nouveau composant. Dans le cas d'un raffinement ou d'une implémentation, il est possible de sélectionner les opérations de l'abstraction dont on souhaite reprendre le prototype.

Il est également possible d'ajouter un composant déjà existant sur le disque : par clic-droit sur le projet, et en sélectionnant « Ajouter composant ». Une fenêtre invite alors l'utilisateur à sélectionner l'emplacement du fichier à importer.

2.4.4 Editer un composant

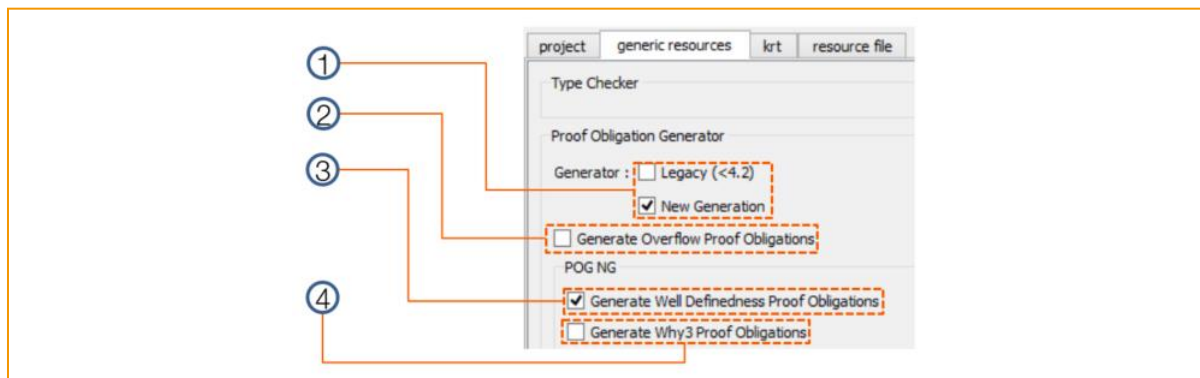
L'édition d'un composant se fait en double-cliquant sur un composant dans l'arborescence projet ou dans la vue centrale du projet. L'éditeur B s'ouvre alors. Cet éditeur propose une coloration syntaxique pour le langage B, ainsi qu'une vue synthétique de la structure du composant.

2.4.5 Contrôler le typage

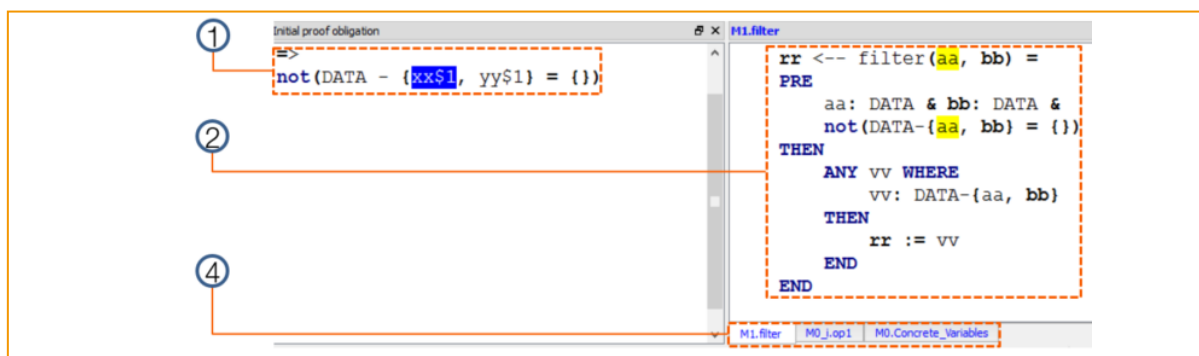
Le contrôle de typage (TC) vérifie la cohérence de typage des composants du modèle (cohérence interne, par exemple entre le type déclaré d'une variable et le type des valeurs qu'on lui affecte et cohérence externe, par exemple entre une variable et son raffinement). Le contrôleur de type effectue également des contrôles de visibilité (détaillés dans le manuel de référence du B).

2.4.6 Générer les obligations de preuve

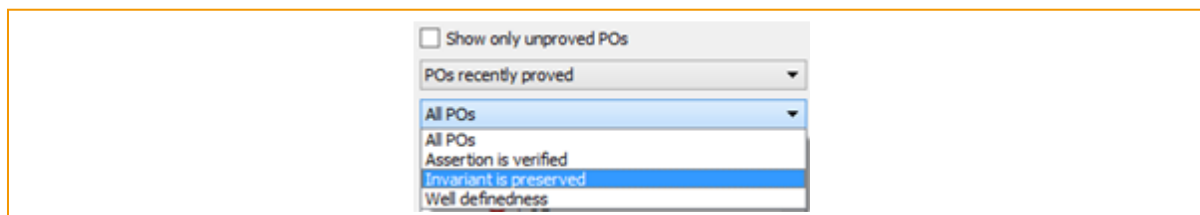
Le générateur d'obligations de preuve (GOP) calcule automatiquement ce qui doit être démontré dans un modèle, en fonction de la théorie de B (voir annexe : les obligations de preuve B et Event-B) et du contenu du modèle.



Le GOP assure la traçabilité des obligations de preuve (OP), au travers d'une interface graphique intégrée permettant d'associer modèle et preuve. Pour chaque obligation de preuve, les portions de modèles en rapport sont affichées dans un onglet à droite de celle-ci. Les expressions contenues dans les OPs sont liées avec le code, lors du surlignage d'une expression de l'obligation de preuve, avec l'affichage et surlignage du code correspondant.



Les OP peuvent être filtrées par type dans l'interface de preuve interactive, grâce à un menu déroulant dynamique qui ne contient que la liste des types d'obligations de preuve du composant.



2.4.7 Prouver automatiquement

Le prouveur automatique dispose de plusieurs stratégies de preuve, appelée « Forces ». Ces stratégies sont des heuristiques de recherche de preuve qui permettent de prouver un nombre important de PO (de l'ordre de 70 à 80% sur la plupart des projets industriels). Ces forces sont hiérarchisées en fonction de leur puissance : plus une force est puissante, plus elle prouve de PO automatiquement, mais plus elle consomme de temps et de mémoire.

Il est fortement recommandé de commencer par les forces les plus faibles (Force rapide, puis Force 0, puis Force 1) puis de progresser vers les forces les plus puissantes. En pratique les Forces 2 et 3 sont quasiment inutilisées, car il est rare qu'elle prouve plus de PO que la Force 1 en un temps raisonnable.

2.4.8 Prouver interactivement

Les PO n'ayant pu être prouvées automatiquement doivent être prouvées interactivement. On ouvre le prouveur interactif par clic-droit sur un composant, puis en sélectionnant « Preuve interactif » dans le sous-menu « Preuve ».

Le prouveur interactif et son interface font l'objet d'une documentation dédiée.

2.4.9 Contrôler les implémentations

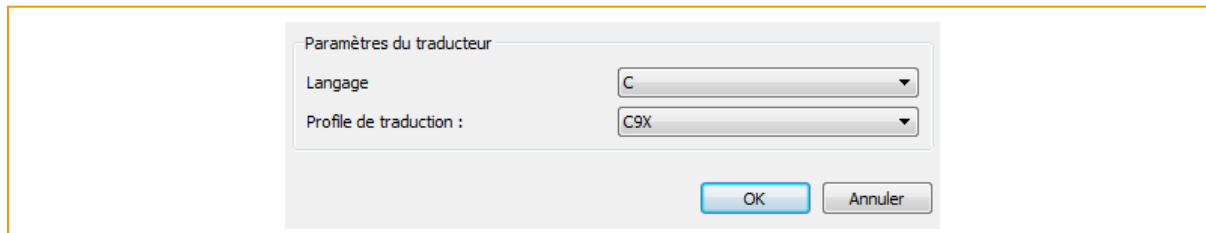
Les traducteurs ne peuvent traduire qu'un sous-ensemble déterministe du B, appelé le B0. Avant de traduire les implémentations, il faut vérifier que les implémentations n'utilisent que du B0 bien formé. Cette vérification, également appelée « B0Check » se fait en sélectionnant « B0 Check » dans le menu contextuel ouvert après un clic droit sur un composant.

2.4.10 Traduire les implémentations

Le générateur de code C C4B a été développé sur la base du compilateur B.

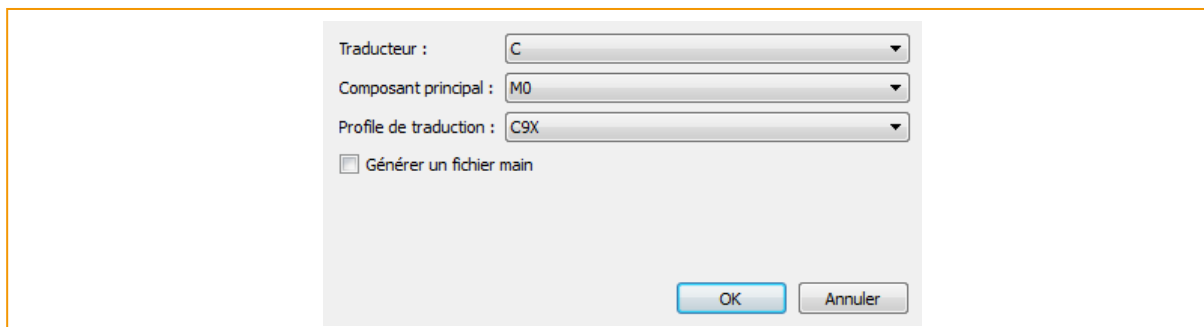
Le générateur de code peut être utilisé :

- en mode **composant**, pour traduire en C une implémentation. Il faut définir le profil de traduction parmi C9X, LIGHT, PROJECT. Les fichiers .c et .h de l'implémentation sélectionnée sont générés dans le répertoire <project translate directory>/c.



- en mode **projet**, pour traduire en C le projet complet. Il faut définir :
 - le nom du composant de plus haut niveau, à choisir dans la liste des implémentations du projet. Il doit s'agir d'un composant ne contenant qu'une seule opération sans paramètre.
 - le profil de traduction : C9X, LIGHT ou PROJECT.
 - Si un cmakefile doit être généré afin de faciliter la compilation du projet.

Les fichiers .c et .h des implémentations du projet, ainsi que le fichier Cmakelist.txt sont générés dans le répertoire <project translate directory>/c.



Le profil Project permet de choisir plus finement comment traduire chaque élément, par l'intermédiaire de ressources positionnées dans le fichier AtelierB.

Elément	Profil C9X	Profil Light	Profil Project
boolean type	bool	unsigned char	user defined
boolean true literal	true	1	user defined
boolean false literal	false	0	user defined
integer type	int32_t	long	user defined
added headers	<stdint.h> <stdbool.h>	/	user defined

- TABLE 2.4.10 -

Les profils de traduction permettent de choisir la traduction de certains éléments

Exemple : pour le projet ci-dessous, on obtient le fichier Cmakelist.txt suivant :

```

graph TD
    ctx[ctx]
    topM[topM]
    topM_i[topM_i]
    M0[M0]
    M0_i[M0_i]
    topM --- topM_i
    M0 --- M0_i
    
```

```

project (P1)

cmake_minimum_required(VERSION 2.4)

SET(P1_SOURCES
    M0_i.c
    topM_i.c
    b_init.c
    b_main.c
)
SET(P1_HEADERS
    M0.h
    ctx.h
    topM.h
    b_init.h
)
add_executable(P1 ${P1_SOURCES} ${P1_HEADERS})
    
```

Le composant b_init.c réalise l'initialisation des différents composants du projet.

Le composant b_main est le composant chapeau : il déclenche l'initialisation des composants (b_init.c) puis les traitements définis dans l'unique opération du composant topM_i. La machine topM doit contenir une seule opération sans paramètre.

C4B ne supporte pas l'intégralité du langage B/B0. Certaines limitations sont listées ci-après.

- Les constantes et les variables de type « tableau de T » doivent être définies avec le type : (0..MAX) --> T où MAX est un littéral entier ou une constante concrète valuée avec un entier.

```

1- IMPLEMENTATION
2   M0_i
3- REFINES
4   M0
5- CONSTANTS
6   C0
7- PROPERTIES
8   C0: NAT
9- VALUES
10  C0 = 12
11- CONCRETE_VARIABLES
12  T1
13- INVARIANT
14  T1: (0..C0) --> INT
15- INITIALISATION
16  T1 := (0..C0) * {0xFF}
17+ OPERATIONS
20  END
21
    
```

1- IMPLEMENTATION

2 M0_i

3- REFINES

4 M0

5- CONSTANTS

6 C0

7- PROPERTIES

8 C0: NAT

9- VALUES

10 C0 = 12

11- CONCRETE_VARIABLES

12 T1

13- INVARIANT

14 T1: (0..C0) --> INT

15- INITIALISATION

16 T1 := (0..C0) * {0xFF}

17+ OPERATIONS

20 END

21

- • Les tableaux de dimension 2 ou supérieure ne sont pas supportés.
- Les constantes concrètes définissant des types tableaux ne sont pas supportés (exemple : `TT = 0..MM --> BOOL` dans la clause `PROPERTIES` avec `TT` constante concrète).
- Les constantes tableau ne sont pas déclarées ou initialisées correctement lorsque les tableaux sont de type `SS --> TT` avec `SS` ensemble abstrait
- Le renommage de machine (multi-instance) et les paramètres de machine abstraite ne sont pas supportés.

2.4.11 Importer un projet

Cette fonction permet la restauration d'un projet géré par l'Atelier B à partir des informations contenues dans une archive créée avec la fonction décrite au paragraphe précédent. Trois options de restauration sont possibles :

- Restauration des fichiers sources B,
- Restauration des fichiers sources B et des fichiers de preuve,
- Restauration de tout le projet.

Une restauration crée toujours un nouveau projet.

Lors de la restauration les informations suivantes sont perdues :

- La liste des bibliothèques du projet,
- La liste des utilisateurs du projet.

Remarque : Vous pouvez restaurer les sources et les fichiers de preuves d'un projet créé avec une version précédente de l'Atelier B. Vous ne pouvez pas effectuer une restauration des sources et des fichiers de preuves si votre archive a été élaborée pour ne contenir que les sources B. L'Atelier B vous avertira que la restauration est impossible.

Un assistant de restauration vous est proposé afin de vous aider dans cette tâche : choisissez tout d'abord un espace de travail dans lequel vous souhaitez restaurer votre projet, puis le chemin d'accès vers l'archive ; ensuite, donnez un nom au nouveau projet, et sélectionnez un répertoire dans lequel restaurer votre projet. Enfin n'oubliez pas de sélectionner le paramètre de restauration voulu avant de cliquer sur Suivant. Si la restauration s'est correctement déroulée, le nouveau projet doit apparaître dans l'espace de travail.

Les composants sont automatiquement attachés au projet, ainsi que les répertoires de définition.

2.4.12 Archiver un projet

Cette fonction permet d'archiver l'ensemble des fichiers constituant un projet géré par l'Atelier B.

L'archive créée est un fichier compressé par la bibliothèque `zlib`, au format `tar`, ayant pour suffixe `.arc`.

Cette fonction peut être utilisée pour :

- Faire la sauvegarde d'un projet,
- Faire une copie d'un projet (pour le transférer sur une autre machine, par exemple)

Il existe trois options pour l'archivage :

- Sauvegarde de l'ensemble des fichiers sources B (`mch`, `ref` et `imp`).
- Sauvegarde de l'ensemble des fichiers sources B et des fichiers de preuve. Avec cette option, l'Atelier B enregistre aussi tous les fichiers nécessaires à la preuve, présents dans la base de données du projet :

- Fichiers po : obligations de preuve
- Fichiers pmi : démonstrations,
- Fichiers pmm : règles utilisateur,
- Fichiers stc : état du composant lors de la génération
- Fichier PatchProver : tactiques utilisateur

- Sauvegarde de tout le projet :
 - Fichiers sources B
 - Fichiers présents dans le répertoire de base de données du projet
 - Fichiers présents dans le répertoire des traductions,

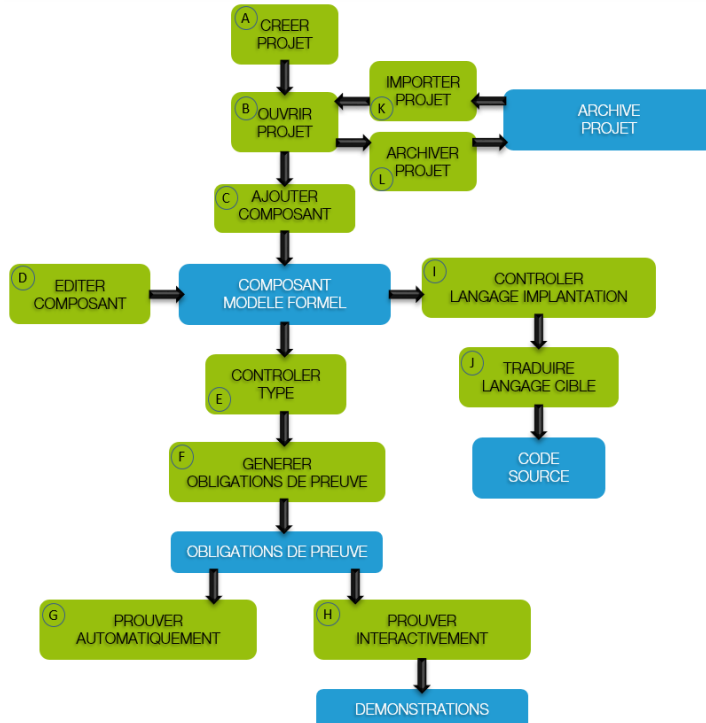
Lorsque l'on archive tout le projet, toutes les informations sont enregistrées. Par conséquent, lorsque le projet sera désarchivé (voir §2.4.11) l'utilisateur le retrouvera dans le même état ; il ne sera donc pas obligé de refaire le contrôle de types, la preuve ...

Afin d'archiver un projet au moyen de l'interface graphique, il suffit, après avoir ouvert l'espace de travail contenant le projet à archiver, de cliquer sur ce dernier, et de sélectionner dans le menu l'option archive. Cela a pour effet d'ouvrir un assistant d'archivage. Sélectionnez le fichier archive de destination, ainsi que le paramètre d'archivage puis cliquez sur le bouton Suivant.

2.5 MODELISER UN SYSTEME

Développé par la société ClearSy¹, l'Atelier B est l'outil industriel qui permet une utilisation Le développement d'un modèle B système a pour objectif de fournir une modélisation prouvée par rapport à ses propriétés.

La figure 2.4.1 présente le séquençement des activités à mener (les actions sont représentées par des rectangles verts, les données par des rectangles bleus).



1 - MACHINE	values(constants,[bind('C0',int(0))]).
2 MO	values(constants,[bind('C0',int(1))]).
3	values(constants,[bind('C0',int(2))]).
4 - CONSTANTS	values(constants,[bind('C0',int(3))]).
5 CO	
6 - PROPERTIES	
7 CO : INT &	
8 CO: 0..4	
9 END	

03.

3 L'ATELIER B EN PROFONDEUR

3.1 GESTION DE PROJET

3.1.1 Paramétrage de l'Atelier B

L'AtelierB est paramétrable à l'aide d'un fichier de ressources, stockées dans un fichier appelé AtelierB. Ce fichier peut exister globalement (dans le répertoire HOME de l'utilisateur), au niveau d'une installation de l'AtelierB (dans le répertoire d'installation), et au niveau d'un projet (dans le répertoire bdp du projet). Les ressources définies dans le fichier projet surchargent les ressources définies aux niveaux supérieurs.

Les ressources et le fichier AtelierB peuvent être édités depuis l'interface graphique de l'AtelierB : clic-droit sur un projet, « propriétés ».

3.1.2 Gestion des utilisateurs

Afin de permettre un contrôle d'accès aux objets gérés par l'Atelier B, il est possible d'ajouter et/ou supprimer des utilisateurs ; par ailleurs, tous les utilisateurs définis sur un projet ont les mêmes droits d'accès en lecture / écriture sur ce dernier. Il est tout de même indispensable d'avoir au moins un utilisateur défini dans la liste.

Cliquez simplement sur les utilisateurs que vous voulez supprimer, puis cliquez sur le bouton remove.

Appuyez sur le bouton Add ... pour faire apparaître une boîte de dialogue permettant la saisie d'un nom d'utilisateur. Si vous souhaitez donner un accès à tout utilisateur du système, vous pouvez spécifier un caractère joker en utilisant le caractère '*'.

3.1.3 Gestion des bibliothèques

Dans le cadre du développement de projets de taille importante, il est essentiel de pouvoir :

- Utiliser des bibliothèques de composants prédéfinis
- Structurer un projet important en plusieurs sous-projets

S'ils le souhaitent, les utilisateurs peuvent lier leurs projets à d'autres projets gérés par l'Atelier. Pour cela, ils disposent de la fonction Add library.

Tout projet qui est accessible peut devenir bibliothèque du projet.

Lorsqu'une bibliothèque est liée à un projet, l'utilisateur du projet peut faire des liens (SEES, IMPORTS, ...) vers les composants de cette bibliothèque.

La fonction Add library vérifie que la bibliothèque à ajouter n'est pas déjà déclarée dans le projet.

Si un composant est défini dans plusieurs bibliothèques du projet, alors le composant qui sera pris en compte est celui défini dans la bibliothèque qui a été ajoutée en premier. En cas de doute sur les composants qui sont pris en compte, il peut être utile d'afficher le graphe de dépendance du projet.

Comme le montre la capture d'écran précédente, la liste des bibliothèques potentielles apparaît dans le cadre Libraries, dans la boîte de droite. Utilisez la souris pour sélectionner le projet à rajouter en tant que bibliothèque puis cliquez sur la flèche dirigée vers la boîte de gauche pour ajouter au projet la bibliothèque sélectionnée.

À l'inverse, sélectionnez la bibliothèque à exclure du projet, puis cliquez sur la flèche dirigée vers la droite, pour exclure une bibliothèque du projet.

3.1.4 Gestion des répertoires de fichiers de définitions

Les fichiers de définitions permettent le regroupement de définitions communes pour plusieurs composants.

Leur description est donnée au chapitre 2.3 du manuel de référence du langage B.

Cette partie décrit les procédures à suivre pour ajouter et supprimer d'un projet B un nouveau répertoire pouvant contenir des fichiers de définitions utilisés par les composants de ce projet.

Toujours sur la page de propriétés des projets (c.f. capture d'écran précédente), dans le cadre `_Definitions directories_`, cliquez sur le bouton `_Add_` pour afficher une boîte de dialogue vous permettant la sélection d'un répertoire.

Inversement, pour supprimer un répertoire de définitions, cliquez sur ce dernier dans la liste prévue à cet effet, et cliquez sur `Remove`.

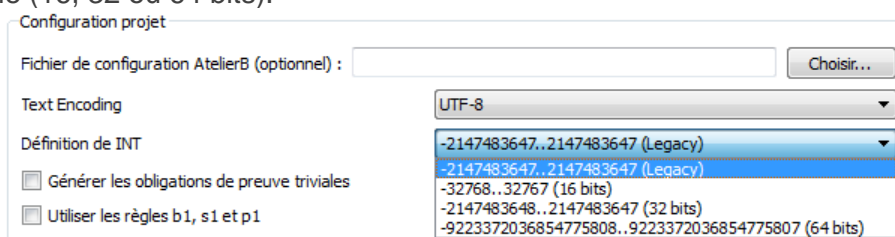
3.1.5 Générateur d'obligations de preuve

Depuis la version 4.2 de l'Atelier B, deux générateurs d'obligations de preuve sont disponibles, la version la plus récente apporte de nouvelles fonctionnalités comme la traçabilité des obligations de preuve.

Ce nouveau générateur produit sensiblement le même nombre d'obligations de preuve que précédemment. Celles-ci peuvent varier toutefois dans leur forme et il est probable qu'un projet prouvé avec une version antérieure de l'Atelier B ne soit plus complètement prouvé, en preuve automatique ou par rejeu, avec la version 4.2. Le nouveau générateur d'obligation de preuve est sélectionné par défaut (New Generation). Pour utiliser l'ancien GOP, il est nécessaire de modifier la configuration des projets et sélectionner « Legacy (<4.2) ».

3.1.6 Paramétrage de INT

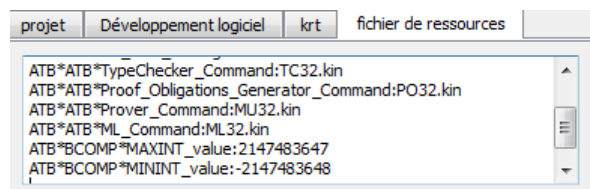
Lors de la création d'un projet, la taille des entiers implémentables est désormais paramétrable (16, 32 ou 64 bits).



L'ensemble des entiers implémentables est défini par $INT = MININT .. MAXINT$. La valeur des constantes prédéfinies `MININT` et `MAXINT` sont ainsi modifiées selon ce choix.

Ce choix est concrétisé dans le fichier de ressources AtelierB du projet, au travers des ressources définissant :

- Les binaires des outils de vérification de type, génération d'obligations de preuves et de preuve (*.kin). Ces fichiers binaires diffèrent selon le nombre de bits utilisés pour les entiers `TC16.kin`, `TC32.kin`, `TC64.kin` par exemple pour le vérificateur de type).
- Les valeurs `MININT_value` et `MAXINT_value`.



Pour modifier ce paramétrage une fois le projet créé, il est nécessaire de :

- modifier les noms des 4 fichiers binaires,
- modifier les valeurs de `MININT_value` et `MAXINT_value`,

en conformité avec le nombre de bits retenus. Il est d'autre part nécessaire de régénérer complètement toutes les obligations de preuve, du fait de la valuation différente de `MININT` et `MAXINT`.

3.1.7 Vérification du typage des commandes de preuve

Les commandes protégées sont : ah, ph, dc, se. La protection consiste à appeler le prouveur de prédicats pp en mode typage pour vérifier que les expressions introduites sont bien typées vis-à-vis du contexte dans lequel elles sont introduites.

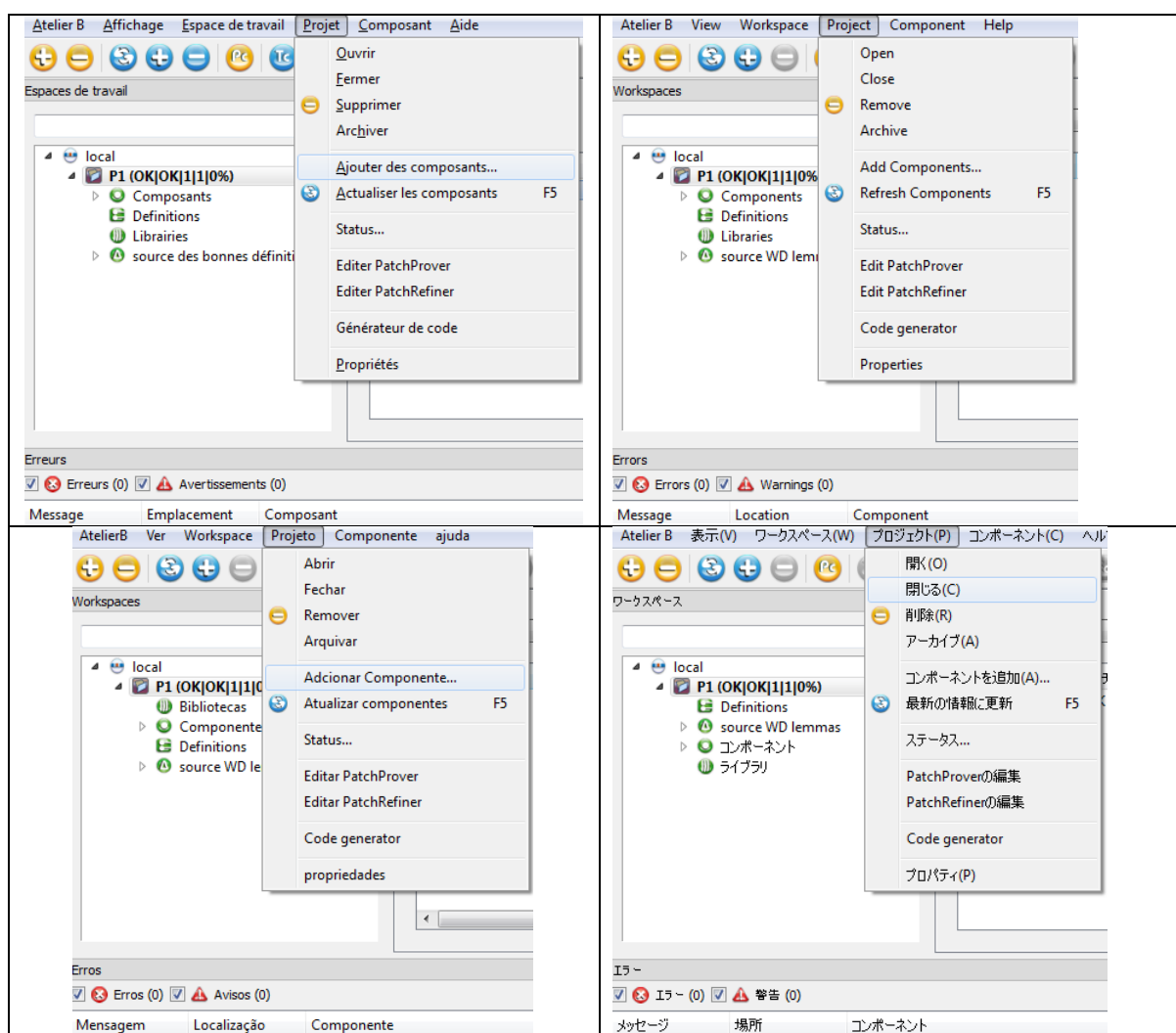
3.1.8 Importation et exportation de projets

3.1.9 Localisation de l'interface graphique

Le choix de la langue dépend de la localisation de l'ordinateur exécutant le logiciel Atelier B. Pour afficher une langue différente, il suffit de :

- modifier le choix dans le menu « Préférences/fenêtre principale/langage »
- modifier la variable d'environnement LANG (fr, en, br, ja, cn) avant de lancer l'Atelier B.

Les langues disponibles sont Anglais, Chinois, Français, Japonais et Portugais (Brésilien).



3.2 MODELISATION

3.2.1 Vérification des règles de codage

3.2.2 Raffinement automatique

3.2.3 RECORDS

Depuis la version 4.2 de l'Atelier B, un nouvel opérateur de mis à jour des records a été introduit lors de la génération des obligations de preuve³. Il permet d'éviter les explosions des expressions et de la mémoire lors de la modification de record. Il corrige aussi les anomalies connues concernant les records, notamment des captures de nom entre les labels des records et les variables du modèle.

```
MACHINE
  REC
  ABSTRACT_CONSTANTS
    tPosHorizontal ,
    tPosition
  PROPERTIES
    tPosHorizontal = struct (
      Latitude : REAL ,
      Longitude : REAL
    ) &
    tPosition = struct (
      Coord : tPosHorizontal ,
      Altitude : REAL
    )
END
```

```
MACHINE
  M2
  SEES REC
  VARIABLES
    yy
  INVARIANT
    yy: tPosition
  INITIALISATION
    yy :: tPosition
  OPERATIONS
    opl =
    BEGIN
      yy := rec(
        Coord: rec(
          Latitude: 0.0,
          Longitude :0.0),
        Altitude: 0.0)
    END
  END
```



```
rec((Coord: rec((Latitude: 0.0), (Longitude: 0.0))), (Altitude: 0.0)): struct((Coord: tPosHorizontal), (Altitude: REAL))
```

exemple d'obligation de preuve générée lors de la valuation d'un record imbriqué - tous les labels doivent être indiqués

3.3 PREUVE

3.3.1 Normalisation des prédicats

De nouveaux principes de normalisation des prédicats ont été définis. Ils sont listés dans les 3 tableaux ci-dessous.

Normalisation but et hypothèses

```
a /= b
a /: b
a <: b
a <<: b
a /<: b
a /<<: b
a <= b (real)
a <= b (float)
a >= b (int)
a >= b (real)
a >= b (float)
a < b (int)
a < b (real)
a < b (float)
a > b (int)
a > b (real)
a > b (float)
a + b (real)
a + b (float)
a - b (real)
a - b (float)
```

Prédicat (expression) normalisé(e)

```
not(a = b)
not(a : b)
a : POW(b)
a : POW(b) & not(a=b)
not(a : POW(b))
a : POW(b) => a=b
a rle b
a <=. B
b <= a
b rle a
b <=. a
a+1 <= b
a rle b & not(a=b)
a <=. b & not(a=b)
b+1 <= a
b rle a & not(b=a)
b <=. a & not(b=a)
a rplus b
a +. b
a rminus b
a -. b
```

<code>a * b (real)</code>	<code>a rmul b</code>
<code>a * b (float)</code>	<code>a *. b</code>
<code>a / b (real)</code>	<code>a rdiv b</code>
<code>a / b (float)</code>	<code>a /. b</code>
<code>a ** b (real)</code>	<code>a rpow b</code>
<code>-a (real)</code>	<code>0.0 rminus a</code>
<code>max(a) (real)</code>	<code>rmax(a)</code>
<code>min(a) (real)</code>	<code>rmin(a)</code>
<code>SIGMA(a).(b c) (real)</code>	<code>rSIGMA(a).(b c)</code>
<code>PI(a).(b c) (real)</code>	<code>rPI(a).(b c)</code>
<code>{a b}</code>	<code>SET(a).(b)</code>
<code>a <=> b</code>	<code>(a => b) & (b => a)</code>
<code>bool(a) = TRUE</code>	<code>A</code>
<code>NAT1</code>	<code>NAT-{0}</code>
<code>NATURAL1</code>	<code>NATURAL-{0}</code>
<code>[]</code>	<code>{}</code>
<code>{a1,...,an}</code>	<code>{a1}\...\{an}</code>
<code>FIN1(a)</code>	<code>FIN(a)-{{}}</code>
<code>POW1(a)</code>	<code>POW(a)-{{}}</code>

normalisation des prédicats et expressions dans le but et hypothèses (remplacement de prédicat/expression par leur forme normalisée)

Normalisation des affectations Prédicat normalisé (lors de la génération des Pos)

<code>a(b) := c</code>	<code>a := a <+ {b -> c}</code>
<code>a'b := c</code>	<code>a := a <<< {b\$8888 = c}</code>

normalisation des affectations (remplacement de prédicat par sa forme normalisée)

Normalisation hypothèses (ajout nouvelles hypothèses) Prédicat normalisé

<code>a : NATURAL</code>	<code>a : INTEGER & 0 <= a</code>
<code>a : b --> c</code>	<code>a : b +-> c & dom(a)=b</code>
<code>a : b >> c</code>	<code>a : b +-> c & a~ : c +-> b</code>
<code>a : b >-> c</code>	<code>a : b +-> c & a~ : c +-> b &</code>
	<code>a : b --> c & dom(a)=b</code>
<code>a : b +->> c</code>	<code>a : b +-> c & ran(a) = b</code>
<code>a : b -->> c</code>	<code>a : b +-> c & ran(a) = b &</code>
	<code>a : b --> c & dom(a)=b &</code>
	<code>a : b +->> c</code>
<code>a : b >>> c</code>	<code>a : b +-> c & ran(a) = b & a~ : c +-></code>
	<code>> b &</code>
<code>a : b >->> c</code>	<code>a : b >> c & a : b +->> c</code>
	<code>a : b +-> c & ran(a) = b &</code>
	<code>a~ : c +-> b & a : b --> c &</code>
<code>a : seq(b)</code>	<code>a : b >> c & a : b +->> c</code>
<code>a : seq1(b)</code>	<code>a : NATURAL-{0} +-> b</code>
	<code>a : seq(b) & a : NATURAL-{0} +-> b &</code>
<code>a : iseq(b)</code>	<code>not(a={})</code>
	<code>a : seq(b) & a : NATURAL-{0} +-> b &</code>
<code>a : iseq1(b)</code>	<code>a~ : b +-> NATURAL-{0}</code>
	<code>a : seq(b) & a : NATURAL-{0} +-> b &</code>
	<code>a~ : b +-> NATURAL-{0} & a : iseq(b) &</code>
	<code>&</code>
<code>a : perm(b)</code>	<code>a : seq1(b) & not(a={})</code>
	<code>a : seq(b) & a : NATURAL-{0} +-> b &</code>
	<code>a~ : b +-> NATURAL-{0} & a : iseq(b) &</code>
	<code>&</code>
	<code>a : seq1(b) & not(a={}) & ran(a) = b</code>

normalisation des prédicats dans les hypothèses (de nouvelles hypothèses sont créées)

3.3.2GOP 4.1 et plus anciens

Il existe deux principales versions du générateur d'obligations de preuve :

- le GOP actuel (version 4.2 et ultérieur)
- le GOP historique (antérieur à la version 4.2)

Le GOP historique permet de générer des obligations de preuve spécifiques au B événementiel :

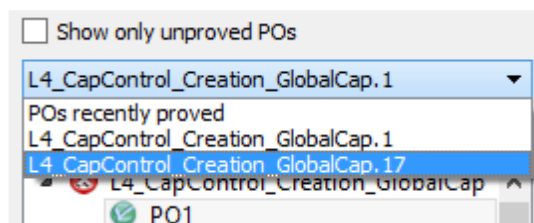
- les lemmes de couverture
- les lemmes d'exclusivité
- les lemmes de non-divergence

Les lemmes de non-divergence consistent à prouver que le variant donné dans le composant (forcément un raffinement) est toujours positif et que chaque nouvel événement le fait décroître. La ressource ATB*POG*Generate_EventB_Non_Divergence_PO indique si les lemmes doivent être générés.

3.3.3 Preuve avec Pmm

3.3.4 Preuve interactive

En preuve interactive, il est possible de revenir à une OP qui a été démontrée de manière interactive et manuellement. Un menu déroulant apparaît avec la liste des obligations de preuve traitées de manière manuelle. L'historique est propre au poste de travail.



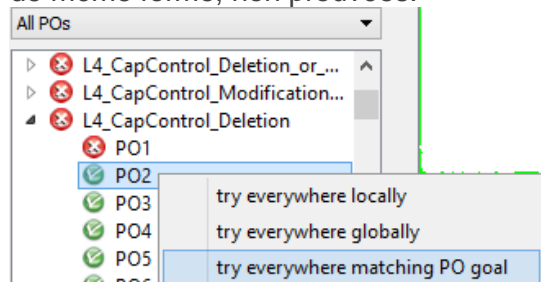
Pour la génération de « User Pass » lors de la sauvegarde d'OP, celle-ci est mise en forme par décomposition de la ligne de commande avec le filtre sur le nom de l'opération d'abord, puis le filtre sur la forme et enfin la démonstration sous la forme d'une liste de commandes.

```
UserPass creation

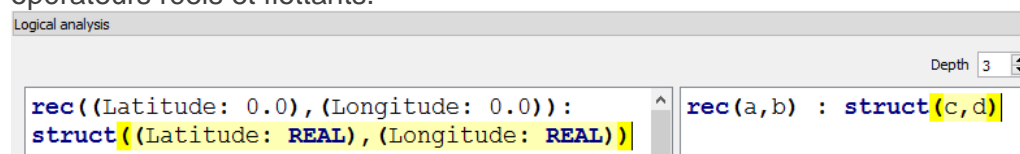
preview

1 operation(Initialisation) &
  Pattern(a : b)
2      & ff(0) & dd(0) & ss & mp
```

La commande TryEverywhere qui permet d'appliquer une démonstration réussie à d'autres obligations de preuve non prouvées dispose d'un nouveau mode d'exécution : il est possible, à partir de la liste des obligations de preuve prouvées (menu contextuel), d'appliquer globalement la démonstration d'une obligation de preuve à toutes les obligations de preuve de même forme, non prouvées.

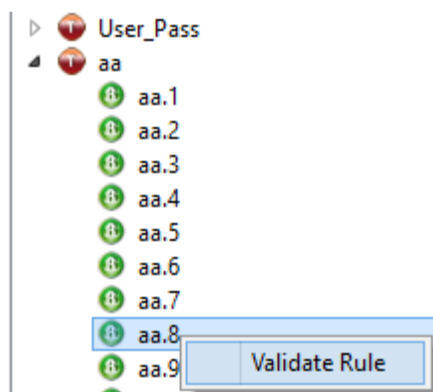


L'analyse logique de formule du prouveur interactif prend en compte les nouveaux opérateurs réels et flottants.



3.3.5 Preuve par règles (version maintenance)

Au sein de l'éditeur de fichier pmm, un menu contextuel accessible dans la vue outline permet de valider une règle.



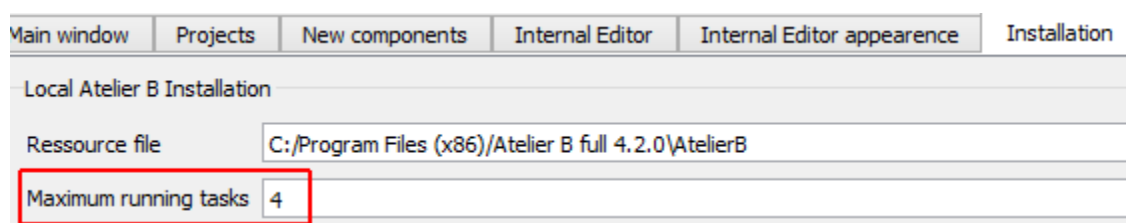
3.3.6 Le support du multi-cœur

3.3.7 Le serveur de preuve

Depuis la version 4.1, il est possible d'exécuter des tâches de preuve en parallèle (voir la Release Notes 4.1.0 pour cette version). Cette fonctionnalité a été étendue avec la version 4.2 grâce à l'utilisation:

- de plusieurs coeurs de la machine locale,
- d'un serveur de preuve distant (uniquement sur Linux).

L'exécution de tâches en parallèle est déterminée par un nombre maximal de tâches strictement supérieur à 1 (voir paramètre "maximum running tasks"). Le nombre de tâches réellement exécutées en parallèle sera toujours inférieur ou égal au nombre de coeurs disponibles. Si ce paramètre vaut 0, alors aucun coeur de la machine locale ne sera utilisé, l'effort de preuve sera alors reporté sur le serveur de preuve distant, s'il existe.



Le nombre de tâches réellement exécutées en parallèle sera toujours inférieur ou égal au nombre de coeurs disponibles. Si ce paramètre vaut 0, alors aucun coeur de la machine locale ne sera utilisé, l'effort de preuve sera alors reporté sur le serveur de preuve distant, s'il existe.

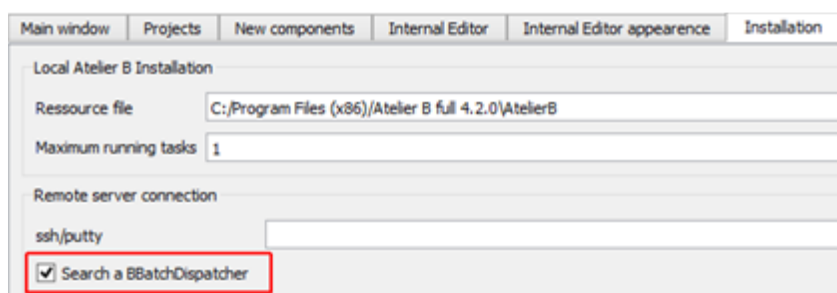
La liste des coeurs disponibles est affichée dans la fenêtre « servers », en commençant par les coeurs de la machine locale. Les coeurs sont nommés « local ::<adresse ip>-<index coeur> » avec <adresse ip> égal à l'adresse IP de la machine exécutant la tâche sur son coeur n° <index coeur>. L'adresse IP de la machine locale est localhost.

Pour déclencher l'exécution de tâches de preuve en parallèle, il suffit de sélectionner au moins un composant d'un projet, de positionner le sélecteur du nombre de tâches (voir Figure 7) à la valeur voulue puis de sélectionner l'action de preuve (F0, F1, etc.).

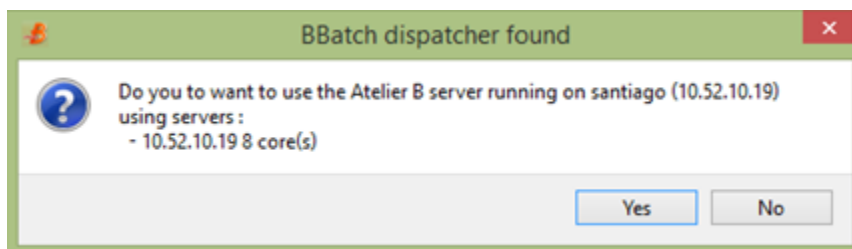


sélecteur du nombre de tâches en parallèle (ici 4)

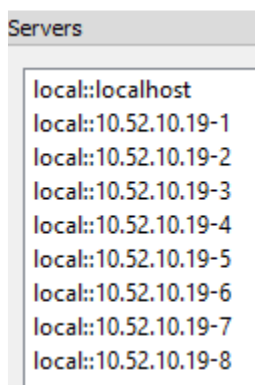
Pour pouvoir se connecter à un serveur de preuve et disposer de bien plus de puissance de calcul, il faut au préalable modifier la configuration de l'Atelier B en autorisant la recherche de serveur de preuve. Pour cela, il convient de cocher la case « search a BBatchDispatcher » dans la page de configuration « installation » de l'Atelier B. Il faudra ensuite redémarrer l'Atelier B.



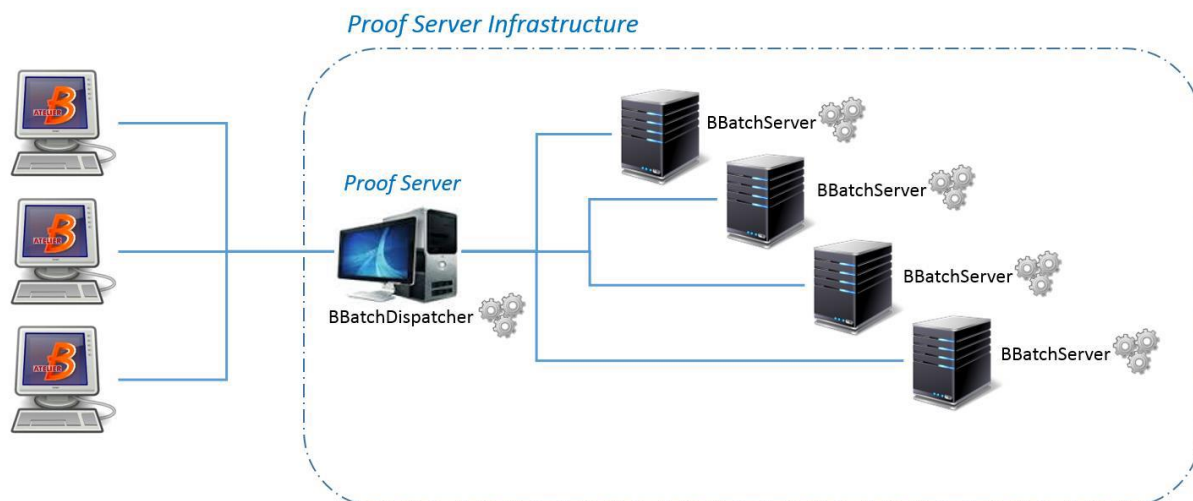
Si au moins un serveur de preuve est actif sur le réseau local, une fenêtre apparaîtra, invitant à se connecter à ce serveur, avec l'indication de son adresse IP et du nombre de coeurs.



En cas de réponse positive, la liste des coeurs disponibles sera étendue avec ceux du serveur de preuve. Les coeurs de la machine locale sont toujours listés avant ceux du serveur de preuve (distant). Lors de l'exécution des tâches en parallèle, la recherche de coeur disponible se fera selon cet ordre.



Un serveur de preuve est un « concentrateur de preuves » : il met en relation des clients Atelier B avec des machines mono ou multi-coeurs réalisant ces preuves. Il n'y pas de contrainte de localisation d'un serveur de preuve qui peut s'exécuter sur n'importe quelle machine Linux.



infrastructure de preuve- l'Atelier B doit être installé sur toutes ces machines

Il est nécessaire d'exécuter (voir Figure 8) :

- Un processus BBatchDispatcher sur le serveur de preuve
- Un processus BBatchServer sur chacune des machines mettant à disposition leurs coeurs pour la réalisation des preuves.

Ces fichiers exécutables se trouvent dans le répertoire d'installation de l'Atelier B.

Pour exécuter un BBatchDispatcher, il convient d'exécuter la ligne de commande:

```
./bbatchdispatcher <hostname> <hostaddress>
```

où <hostname> est le nom de la machine et <hostaddress> son adresse IP.

Un serveur web est alors démarré à l'adresse `http://localhost:<port>/servers.html` (le numéro du port, <port>, est indiqué lors du lancement). Il fournit une interface d'information et de commande (voir Figure 9) qui indique la liste des BBatchServers associés, ainsi que le nombre de coeurs associés.

BBatchDispatcher santiago (10.52.10.19)

Started at ven. nov. 28 14:26:55 2014.

Last broadcast message emitted at ven. nov. 28 15:37:30 2014.

Servers registered:

Address	Alive	Cores	Free Cores	Occupied Cores	Actions
10.52.10.19 yes	8	4	4		<input type="button" value="Reserve"/> <input type="button" value="Release"/>

serveur web du BBatchDispatcher indiquant l'état des BBatchServers associés

Pour exécuter un BBatchServer, il convient d'exécuter la ligne de commande:

```
./bbatchserver <hostaddress> <cores> -d <dispatcheraddress>
```

où <hostaddress> est l'adresse IP de la machine, <cores> le nombre de coeurs disponibles et <dispatcheraddress> l'adresse IP du serveur de preuve exécutant BBatchDispatcher.

3.3.8 Utilisation de prouveurs externes avec Why3

3.4 GENERATION DE CODE

3.4.1 Traducteur C

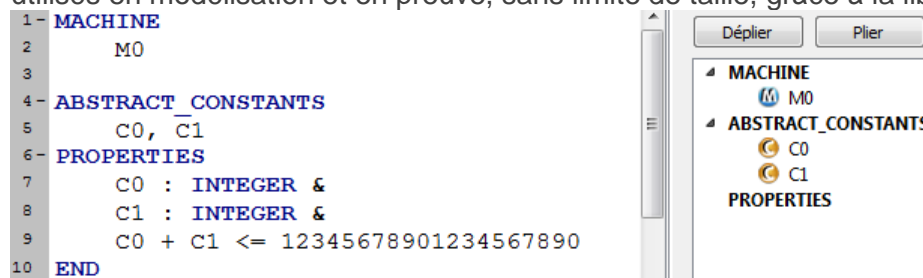
3.4.2 Traducteur HIA (version maintenance)

3.5 LANGAGE B SUPPORTÉ PAR L'ATELIER B

Le langage B supporté par l'Atelier B diffère de celui décrit dans le B-Book.

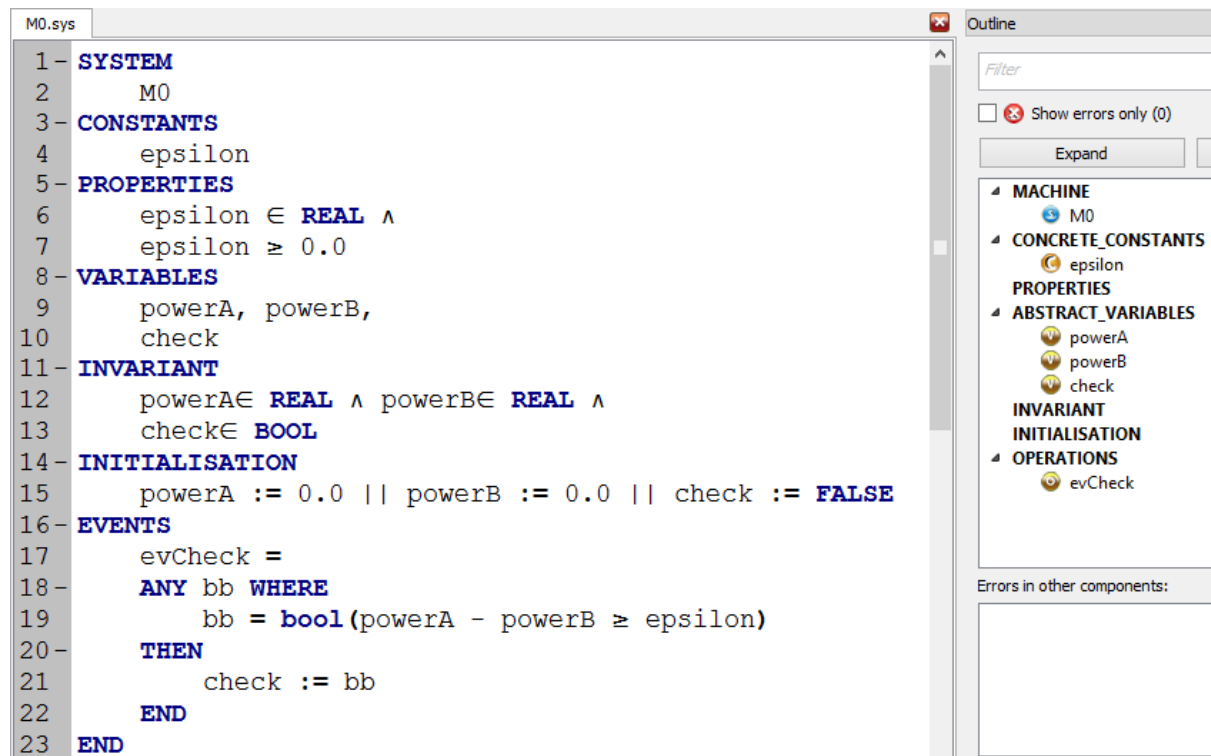
3.5.1 Support des entiers de grande taille

Les entiers littéraux ne sont plus limités à l'ensemble INT. Dorénavant, ils peuvent être utilisés en modélisation et en preuve, sans limite de taille, grâce à la librairie BigInt::GMP.



3.5.2 Support expérimental des nombres réels et flottants

Depuis la version 4.1, il existe un support des nombres réels et nombres flottants (voir la Release Notes 4.1.0 pour cette version). Le type des réels est REAL, celui des flottants est FLOAT. Avec la version 4.2, la gestion des nombres réels et nombres flottants a été améliorée et a subi des modifications.



modèle événementiel incluant des variables réelles

Il y a plusieurs points importants à noter :

- les nombres non entiers sont uniquement gérés par le nouveau GOP ;
- Contrairement à la version 4.1 qui distinguait les opérateurs entiers, réels et flottants, les opérateurs utilisés dans les modèles B sont unifiés (voir Tableau 4, Tableau 5 et Tableau 6)
- Par contre, en phase de preuve, les opérateurs sont distingués car leur sémantique est différente selon leur type. Le langage utilisé n'est pas le même dans le source des composants et dans le prouveur (et donc aussi dans les règles des fichiers .pmm)

Lors de la génération des obligations de preuves, il y a conversion depuis la syntaxe unifiée vers une syntaxe dédiée aux types non-entiers. Le type des opérandes est utilisé pour déterminer quel type d'opérateur utiliser. Il n'y a pas de conversion ni de coercition implicite.

Unifié	Entier	Réels	Flottants
$x \leq y$	$x \leq y$	$x \text{ rle } y$	$x \leq . y$
$x < y$	$x < y$	$x \text{ rlt } y$	$x < . y$
$x \geq y$	$x \geq y$	$x \text{ rge } y$	$x \geq . y$
$x > y$	$x > y$	$x \text{ rgt } y$	$x > . y$
$x + y$	$x + y$	$x \text{ rplus } y$	$x + . Y$
$- x$	$- x$	$0.0 \text{ rminus } x$	$- . X$
$x - y$	$x - y$	$x \text{ rminus } y$	$x <= . y$
$x * y$	$x * y$	$x \text{ rmul } y$	$x * . Y$
x / y	x / y	$x \text{ rdiv } y$	$x / . y$
$x ** y$	$x ** y$	$x \text{ rpow } y$	Invalid
$\min(x)$	$\min(x)$	$\text{rmin}(x)$	Invalid
$\max(x)$	$\max(x)$	$\text{rmax}(x)$	Invalid
$\text{SIGMA}(x) . (y \mid z)$	$\text{SIGMA}(x) . (y \mid z)$	$\text{rSIGMA}(x) . (y \mid z)$	Invalid
$\text{PI}(x) . (y \mid z)$	$\text{PI}(x) . (y \mid z)$	$\text{rPI}(x) . (y \mid z)$	Invalid

conversion des prédicats des modèles (colonne de gauche) pour la preuve, en fonction de leur type

Signification	type	Syntaxe	Type de l'opérande
résultant			
plongement entier dans réel	entier	$\text{real}(x)$	$x : \text{INTEGER}$
$\text{réel } \text{real}(x) : \text{REAL}$			
partie entière	entière	$\text{floor}(x)$	$x : \text{REAL}$
$\text{floor}(x) : \text{INTEGER}$			
partie entière par excès	entière	$\text{ceiling}(x)$	$x : \text{REAL}$
$\text{ceiling}(x) : \text{INTEGER}$			

opérateurs de conversion entier/réel et réel/entier

Les flottants sont considérés comme un type implémentable et ils ne possèdent donc pas des opérateurs de spécification comme min, max, SIGMA et PI.

Il n'y a pas de littéraux flottants, il faut passer par une machine de base.

Il n'y a pas d'opérateur prédéfini pour passer des flottants aux réels, ou des flottant aux entiers (et vice-versa).

La normalisation (dans les obligations de preuves) de la relation d'ordre des réels et des flottants est résumée ci-dessous.

Unifié	Entier	Réels	Flottants
$x < y$	$x+1 \leq y$	$x \text{ rle } y \ \& \ x \neq y$	$x \leq . y \ \& \ x \neq y$
$x \geq y$	$y \leq x$	$y \text{ rle } x$	$y \leq . x$
$x > y$	$y+1 < x$	$y \text{ rle } x \ \& \ y \neq x$	$y \leq . x \ \& \ y \neq x$

normalisation de la relation d'ordre pour les entiers, réels et flottants

3.5.3 Support des nombres littéraux en hexadécimal

Les littéraux hexadécimaux sont utilisables en modélisation et en preuve, en utilisant le préfixe usuel 0x suivi d'une représentation en base 16 (0..9, A..F ou a..f). Les littéraux

hexadécimaux sont de type entier. Combinée au support des entiers de grande taille, cette amélioration permet de manipuler des littéraux hexadécimaux de grande taille.

```
1- MACHINE
2-     ctx
3- CONSTANTS
4-     BASE_MEMORY,
5-     MASK1
6- PROPERTIES
7-     BASE_MEMORY <: NAT &
8-     MASK1 : NAT &
9-     BASE_MEMORY = 0x1FF006FF .. 0x1FF02000| &
10-    MASK1 = 0x1E
11- END
12-
```

3.6 LANGUAGE EVENT-B SUPPORTÉ PAR L'ATELIER B

L'Atelier B v4 propose un support pour la modélisation de système. En effet, il est désormais possible lors de la création d'un projet de spécifier que le projet est destiné à la modélisation de systèmes. Dès lors, dans ce projet, lors de l'ajout de composants on propose d'ajouter des composants système ou des raffinements.

Ce langage est décrit dans le document « Event-B in AtelierB » accessible depuis le menu d'aide, sous-menu « manuels » de l'AtelierB.

04.

4 CONTRAINTES D'APPLICATION SÉCURITAIRES

Cette section liste les conditions sécuritaires d'utilisation de l'AtelierB. Cette liste n'est pas le fruit d'une analyse de sécurité détaillée et ne prétend pas être exhaustive.

- [SRAC1] L'absence d'initialisation conditionnelle dans les modèles B doit être vérifiée manuellement.
- [SRAC2] Le typage des paramètres scalaires de machine doit être vérifié manuellement. Les types autorisés sont les entiers, les booléens et les paramètres ensemblistes de ladite machine.
- [SRAC3] Le fait que les paramètres d'opérations (entrées et sorties) soient deux à deux distincts doit être vérifié manuellement.

Nous signalons également deux écarts connus par rapport au manuel de référence du B. Ces écarts sont des limitations sans impact sécuritaire : ils peuvent causer le rejet par les outils d'un modèle B logiquement valide, mais ne peuvent pas causer l'acceptation par les outils d'un modèle B logiquement invalide. Ils sont donc mentionnés à titre informatif.

- Les variables renommées ne sont pas supportées.
- Les opérations renommées ne sont pas supportées.

05.

5 ANNEXES TECHNIQUES

MODE BATCH, COMMANDES PREUVE INTERACTIVE, OBLIGATIONS DE PREUVES

5.1 MODE BATCH

Le mode batch de l'Atelier B permet son utilisation depuis une console ainsi qu'une utilisation en mode semi-automatique par fichiers de commande. L'interpréteur de commande, bbatch, possède en grande majorité mêmes fonctionnalités que l'interface graphique de l'atelier B.

Pour exécuter l'interpréteur de commandes, il ouvre une fenêtre console, se déplace dans le répertoire contenant les binaires de l'Atelier et tape la commande `./lanceBB`.

Sous environnement Windows, un raccourci est créé par défaut dans le menu démarrer (Raccourci nommé « AtelierB Batch interface »).

Pour quitter l'interface utilisateur mode commande, il faut taper `quit` ou `q`.

Vous pouvez utiliser cette interface de deux manières :

- De manière interactive, ou
- Avec un fichier de commandes.

Un fichier de commandes peut contenir :

Des commentaires : lignes commençant par le caractère `#`,

Des commandes Atelier B : forme longue ou abrégée,

Exemple de fichier de commandes :

```
#-----  
#Ceci est un commentaire  
#-----  
# liste des projets :  
show_projects_list  
# utilisateurs du projet library  
spul library  
#fin du fichier de commandes
```

Pour exécuter un fichier de commandes (sous Linux uniquement), il faut taper l'une des commandes suivantes :

```
lanceBB -i=nom_fichier
```

ou

```
lanceBB < nom_fichier
```

```
ou lanceBB << END
```

contenu du fichier de commandes

```
END
```

Certaines commandes ont des paramètres par défaut. Pour les commandes de gestion des projets,

l'interpréteur mémorise le dernier nom de projet tapé. De la même manière, l'interpréteur mémorise

le dernier nom de composant tapé. Pour utiliser ce paramètre par défaut, il faut juste taper `<return>`.

Exemple :

```
bbatch 3> typecheck MM_1
```

```
....
```

```
bbatch 4> pogenerate
```

```
pogenerate MM_1 1 ? (yes=return)
```

...

Utilisation de l'aide interactive

La commande `help a_che` la liste des commandes disponibles. La liste est a_chée de la façon suivante :

General commands :

(cd) `change_directory`

...

(v) `version_print`

Project level commands :

(add) `add_definitions_directory`

(apl) `add_project_lib`

...

(spl) `show_projects_list`

Machine level commands (available after `open_project`) :

(b2c) `ComenCtrans`

(af) `add_file`

...

(us) `unproved_status`

(vr) `verify_rule`

Les commandes sont présentées dans l'ordre suivant :

1. Commandes générales,
2. Commandes de gestion des projets,
3. Commandes applicables à des composants : il faut ouvrir un projet pour pouvoir utiliser ces commandes.

Les raccourcis des commandes sont indiqués entre parenthèses juste avant le nom de la commande.

Vous pouvez également obtenir de l'aide sur une commande particulière en tapant la commande :

`help nom_commande`

ou

`h nom_commande`

Par exemple :

`bbatch 9> help help`

`help [command] get help on commands`

5.2 LISTE DES COMMANDE INTERACTIVES DE PREUVE

Développé par la société ClearSy¹, l'Atelier B est l'outil industriel qui permet une utilisation

5.3 OBLIGATIONS DE PREUVE B

Développé par la société ClearSy¹, l'Atelier B est l'outil industriel qui permet une utilisation

5.4 OBLIGATIONS DE PREUVE EVENT-B

Développé par la société ClearSy¹, l'Atelier B est l'outil industriel qui permet une utilisation

5.5 GENERATEUR D'OBLIGATIONS DE PREUVE 4.2+

Les obligations de preuve théoriques sont des paramètres de l'outil. Elles sont disponibles dans le répertoire d'installation de l'Atelier B, dans le sous-répertoire press/include :

- paramGOPSoftware.xml : obligations de preuve pour le B logiciel
- paramGOPSystem.xml : obligations de preuve pour le B système
- wellDefinedness.xml : obligations de preuve pour la bonne définition

Ces obligations de preuve peuvent être étendues en modifiant ces fichiers.

```
514      <Proof_Obligation>
515          <Tag>WellDefinednessProperties</Tag>
516          <Definition name="B definitions"/>
517          <Definition name="ctx"/>
518          <Definition name="mhcst"/>
519          <Definition name="aprp"/>
520          <Hypothesis><xsl:copy-of select="Sets/*"/></Hypothesis>
521          <Goal>
522              <Well_Definedness>
523                  <xsl:copy-of select="Properties/*"/>
524              </Well_Definedness>
525          </Goal>
526      </Proof_Obligation>
```

obligation de preuve pour la bonne définition des propriétés des constantes d'une machine

Par ailleurs, dans ce même répertoire, on trouve la définition de la grammaire du bxml (bxml.xsd), nouveau format de représentation des modèles B permettant l'interopérabilité.

05.

6 REFERENCES

GLOSSAIRE, BIBLIOGRAPHIE, LIENS UTILES

6.1 GLOSSAIRE

- **Composant** : modèle textuel pouvant être une spécification B (machine abstraite), un raffinement ou une implémentation.
- **Obligation de preuve** (OP) :
- **Générateur d'obligation de preuve** (GOP) :
- **Prouveur** :
- **Base de données projet** (BDP) : répertoire contenant les informations internes à un projet.
- **Manifest** :

6.2 BIBLIOGRAPHIE

- **The B-Book : Assigning programs to meanings**, J.R. Abrial, Cambridge University Press (1996)
- **Modeling in Event-B : system and software engineering**, J.R. Abrial, Cambridge University Press (2010)
- **The B-Method : an introduction**, S. Schneider, Palgrave Macmillan (2001)
- **Program Development by Refinement : case-studies using the B-Method**, E. Sekerinski & K. Sere, Springer (1999)
- **Spécification formelle avec B**, H. Habrias, Lavoisier (2001)



Clearsy System Engineering

Parc de la Duranne
320 Av. Archimède
Les pléiades III Bat. A
13857 Aix-en-Provence Cedex 3

Phone /Online

Phone : +33 (0)4 42 37 12 70
Fax : +33 (0)4 42 37 12 71
contact@clearsy.fr
www.atelierb.eu