

Remerciements

Je tiens à remercier les personnes ayant permis à mon travail de fin d'études de voir le jour.

Je remercie **Monsieur Didier Solheid** (responsable de l'équipe MES-IT et maître de stage), pour son suivi durant toute la durée de mon stage, et pour m'avoir accueilli au sein de Technord.

Je remercie **Monsieur Kévin Putzeys** (ingénieur projet de l'équipe MES-IT), pour son aide et ses conseils durant toute la durée de mon stage.

Je remercie **Monsieur Christophe Brose** (maître assistant à la HEPL et professeur superviseur), pour son suivi durant toute la durée de mon stage, ainsi que pour la relecture de ce document et pour ses conseils lors de la rédaction de celui-ci.

Je remercie **l'ensemble du corps enseignant de la Haute École de la Province de Liège**, pour les savoirs qu'ils m'ont transmis tout au long de mes études.

Je remercie **l'ensemble du personnel de Technord Liège**, pour son accueil et pour l'excellente ambiance de travail.

Je remercie **ma sœur et ma mère**, pour la relecture attentive de ce document.

Je remercie **mes parents**, pour le soutien tant financier que moral tout au long de mes études.

Enfin, je remercie toutes les personnes ayant contribué de près ou de loin à la réalisation de mon stage et de mon travail de fin d'études.

Table des matières

1. Introduction.....	1
2. Présentation de l'entreprise.....	2
2.1 – La société Technord.....	2
2.2 – Historique	3
2.3 – Secteurs d'activités	3
2.4 – Technord dans le monde	4
2.5 – Implantations	5
2.6 – Le département MES-IT	6
3. Cahier des charges.....	7
3.1 – Description générale	7
3.2 – Objectifs principaux	8
3.3 – Documents et rapports générés	9
3.3.1 - Import et export de données	9
3.3.2 - Rapports de type « template list »	10
3.3.3 - Rapports de type « measurement list » et « alarm list ».....	11
3.4 – Objectifs secondaires	12
4. Analyse	13
4.1 – Cas d'utilisation.....	15
4.2 – Évaluation de l'existant	16
4.2.1 - aaExport – projet open source.....	16
4.2.2 - CreateGalaxy – ancien projet interne	17
4.3 – Exploration des technologies.....	18
4.3.1 - Wonderware ArchestrA.....	18
4.3.2 - GRAccess – API d'ArchestrA	25
4.3.3 - Tâches asynchrones en C# .NET	36
4.3.4 - EPPlus – librairie de création de documents Excel	39
4.3.5 - DocX – librairie de création de documents Word	41
4.3.6 - Subversion – utilitaire de contrôle de versions.....	44
4.4 – Organisation du projet	46
4.4.1 - Architecture.....	46
4.4.2 - Décisions architecturales	50
4.4.3 - Structure de la solution	53
4.4.4 - Diagramme des classes	56

5. Développement	58
5.1 – Outils utilisés.....	59
5.2 – Fonctionnalités du projet.....	61
5.2.1 - Connexion.....	61
5.2.2 - Arbres de données.....	62
5.2.3 - Liste de données avec filtres	66
5.2.4 - Sauvegarde et chargement de l'état des arbres.....	68
5.2.5 - Fenêtre de recherche	69
5.2.6 - Gestion des tâches et file d'attente.....	74
5.2.7 - Affichage et suppression d'attributs <i>UDAs</i>	77
5.2.8 - Import et export de données	78
5.2.9 - Export compatible avec ArchestrA	82
5.2.10 - Rapports de documentation Word.....	83
5.2.11 - Rapports de documentation Excel.....	86
5.2.12 - Fenêtre d'options et configuration	88
5.3 – Design et ergonomie.....	90
5.3.1 - Interface graphique.....	90
5.3.2 - Amélioration de l'expérience utilisateur.....	92
5.3.3 - Tests d'ergonomie.....	93
5.4 – Problèmes rencontrés.....	96
6. Procédure d'installation.....	98
6.1 – Prérequis.....	98
6.2 – Installation de l'application.....	99
6.3 – Lancement de l'application	99
7. Avenir du projet	100
8. Conclusion.....	101
Bibliographie.....	102
Annexes	104
Annexe 1 – Diagramme des classes	104

Chapitre 1

Introduction

Durant la dernière année du bachelier en *informatique et systèmes*, un stage de quatorze semaines doit être réalisé en entreprise. Cette immersion en milieu professionnel a pour but de découvrir les différents aspects du métier d'analyste programmeur, tout en approfondissant différentes connaissances acquises au cours des études.

Dans le but d'ouvrir un maximum de portes, mon choix s'est porté sur un stage mêlant à la fois l'informatique de gestion et l'informatique industrielle. J'ai opté pour la société d'automation Technord, située au Sart-Tilman. L'intégration de leur équipe MES-IT¹ s'est révélée très agréable. L'ambiance familiale et la flexibilité des horaires m'ont permis de réaliser le travail demandé dans des conditions optimales.

Le projet qui m'a été attribué consistait en la réalisation en langage C# d'une application de gestion de données pour le système de supervision Wonderware ArchestrA. L'application devait répondre aux besoins de l'équipe MES-IT de Technord : automatiser l'import et l'export de grandes quantités de données, permettre la modification de ces données (à l'aide de Microsoft Excel) et générer de la documentation au sujet d'un projet ArchestrA. Mon travail avait donc pour but d'offrir un gain de productivité considérable.

La mise en œuvre du projet a commencé par une phase d'analyse, comprenant notamment la découverte des logiciels et la réalisation d'une série de tests, ainsi que la gestion de l'architecture logicielle et la prise de décisions techniques. Après m'être familiarisé avec le système ArchestrA, une étude approfondie de son API² GРАccess s'est révélée nécessaire. Cette dernière s'étant montrée très limitée, j'ai également étudié en détail la base de données SQL d'ArchestrA, permettant une récupération plus complète et plus flexible des données, au prix d'une complexité accrue. Sitôt l'analyse terminée, le développement réel de l'application a pu commencer, suivi d'une étude d'amélioration de l'utilisabilité et l'ergonomie.

Le contenu de ce document est divisé en plusieurs chapitres, suivis d'une conclusion :

- **Présentation de l'entreprise** : courte présentation de Technord et du département MES-IT ;
- **Cahier des charges** : la description des différents objectifs demandés ;
- **Analyse** : l'explication détaillée des différentes étapes de l'analyse ;
- **Développement** : la présentation point par point de l'application réalisée ;
- **Procédure d'installation** : un guide concis destiné aux utilisateurs de l'application ;
- **Avenir du projet** : ce qu'il adviendra de mon travail au sein de Technord.

¹ MES-IT : Manufacturing Execution System, Information Technology

² API : interface de programmation (Application Programming Interface)

Chapitre 2

Présentation de l'entreprise

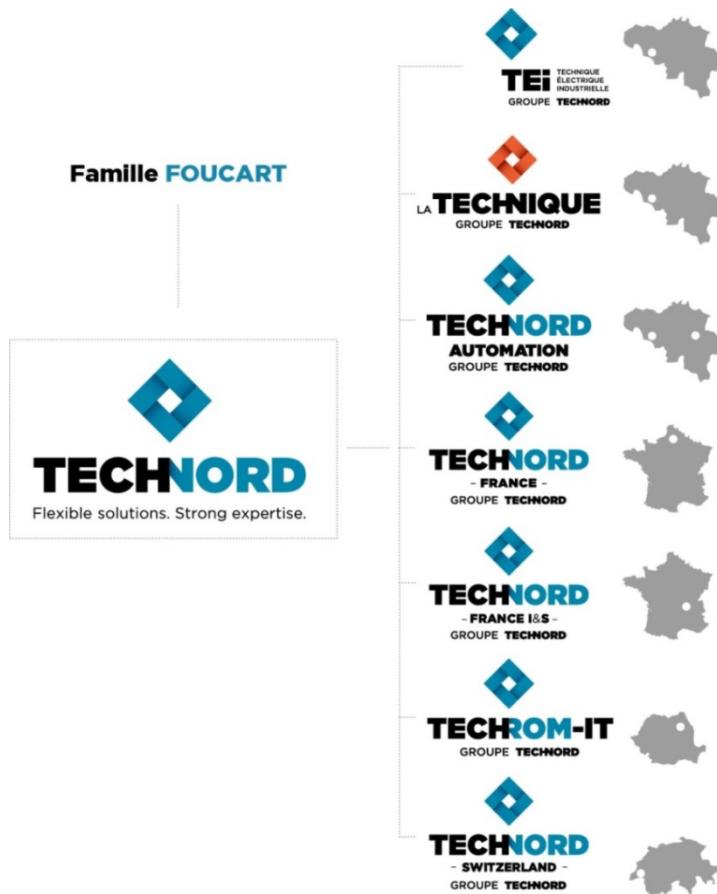
2.1 – La société Technord

« Première entreprise familiale de Wallonie picarde dans son secteur d'activités, Technord rassemble les compétences nécessaires à la conception, la fabrication, l'installation, la mise en service et la maintenance d'équipements industriels sur les cinq continents.¹ »

Créé en 1988 par Michel Foucart, le groupe Technord comprend 3 entreprises complémentaires :

- *Technique Électrique Industrielle (T.E.I.)* : entreprise générale d'électricité et d'électronique ;
- *Technord Automation* : entreprise spécialisée dans l'informatique industrielle, l'automation et les réseaux ;
- *La Technique* : entreprise de distribution de matériel électrique industriel spécialisé dans l'automatisme et les réseaux.

Le groupe comprend à ce jour six implantations situées dans plusieurs pays : la Belgique (Tournai, Liège), la France (Lille, Lyon), la Roumanie (Iasi) et la Suisse (Nyon). Technord Liège fait partie de *Technord Automation*.



Source : <http://www.technord.com/fr/corporate/organisation>

¹ Source : Technord groupe, <http://www.technord.com/fr/corporate/organisation/presentation-generale>, 7/05/2016

2.2 – Historique

1945	Création de la société <i>Technique Électrique Industrielle</i> : entreprise générale d'électricité et d'électromécanique.
1975	Michel Foucart devient Directeur salarié de la T.E.I.
1982	Création de la société <i>Technord Automation</i> : entreprise spécialisée dans l'informatique industrielle, l'automatisme et les réseaux.
1988	Création du groupe Technord. Michel Foucart en devient l'Administrateur Délégué. Crédit : Technord
1997	Création de la société <i>La Technique</i> : entreprise de distribution de matériel électrique industriel et d'éclairage.
2004	Ouverture de la filiale Technord France s.a.s (à proximité de Lille).
2010	Ouverture de la filiale Techrom-IT (Roumanie, Iasi). Ouverture d'une filiale à Lyon. Philippe Foucart devient l'Administrateur Délégué du groupe et Michel Foucart devient Président-Fondateur.
2013	Ouverture de la filiale Technord Switzerland (à Nyon, à proximité de Genève).

2.3 – Secteurs d'activités

Du génie électrique à l'informatique industrielle, Technord a évolué au rythme de la technologie. Ceci lui a permis de continuellement pouvoir répondre aux demandes des clients avec des solutions sur mesure.



Source : Présentation de Technord (PowerPoint)

Le cœur du métier chez Technord est la gestion de projets, sur le court et long terme, ce qui lui permet de bien connaître les spécificités de ses associés et de leur proposer des solutions appropriées et innovantes. L'esprit de l'entreprise est basé sur trois axes :

- la flexibilité : elle permet de répondre rapidement et au mieux à la demande des clients ;
- la rapidité de décision : la taille modeste de l'entreprise permet d'anticiper ses besoins ;
- le savoir-faire du personnel : il permet de faire face efficacement aux défis des clients.

Les principaux domaines visés par Technord sont le domaine pharmaceutique, l'agro-alimentaire, la chimie, la verrerie, l'infrastructure et le transport. Les activités sont diversifiées :

- le génie électrique en basse et moyenne tension, et la fabrication d'armoires électriques ;
- l'informatique industrielle, le process control et la supervision ;
- la gestion technique de bâtiments et d'équipements, et les réseaux industriels ;
- l'utilisation rationnelle de l'énergie ;
- la distribution de produits électriques et d'automatisme ;
- la maintenance et le dépannage.

Technord possède de nombreuses certifications technologiques, ainsi que des certifications de sécurité et enfin des certifications fournies par des clients, tels que *Nestlé* et *Proximus*. La majorité des certifications technologiques leur ont été attribuées par *Siemens* et *Wonderware*. D'autres concernent des produits *Rockwell Automation*, *Schneider Electric*, *Corning* ou encore *Panduit*.



2.4 – Technord dans le monde

Le groupe Technord rassemble de nombreuses compétences et réalise des projets dans le monde entier. La carte qui suit offre un aperçu des différents lieux concernés par ces projets.



Source : Présentation de Technord (PowerPoint)

2.5 – Implantations¹



Siège du groupe Technord – Tournai

Les trois parties du groupe y sont représentées :

- *Technique Électrique Industrielle* (électricité et électronique) ;
- *Technord Automation* (informatique industrielle, automatisme) ;
- *La Technique* (distribution de matériel).



Technord – Liège

Le site de Liège est une partie importante de *Technord Automation*.

Il se divise en deux départements :

- le département PLC-SCADA² (automatisation et supervision) ;
- le département MES-IT³ (informatique industrielle et gestion).



Technord France - Lille

L'implantation de Lille est une partie importante de l'entreprise *Technique Électrique Industrielle*.

Ses rôles principaux sont l'électricité générale et le génie électrique. Elle représente aussi *Technord Automation*.



Technord France - Lyon

L'implantation de Lyon est similaire à celle de Liège.

Elle comprend les deux mêmes départements :

- le département PLC-SCADA ;
- le département MES-IT.



Technord Switzerland – Nyon (Suisse)

Technord Switzerland se limite à la distribution de matériel électrique industriel et d'éclairage.



Techrom-IT – Iasi (Roumanie)

Le site roumain est essentiellement chargé de tâches de gestion, de planification et d'échanges de courrier.

¹ Source des images : Présentation de Technord (PowerPoint)

² PLC-SCADA : Programmable Logic Control, Supervisory Control and Data Acquisition

³ MES-IT : Manufacturing Execution System, Information Technology

2.6 – Le département MES-IT

Mon stage s'est déroulé dans le département MES-IT de l'implantation liégeoise de Technord. Ce département dépend de l'entreprise *Technord Automation*.



Le M.E.S. (Manufacturing Execution System) représente le lien entre les systèmes de gestion globale des entreprises (E.R.P.) et les systèmes de pilotage des processus industriels. Il contribue à l'amélioration de l'efficacité des lignes de production. L'I.T. (Information Technology) consiste en l'utilisation et/ou la mise au point de technologies informatiques pour récupérer, manipuler et transmettre des données.

Le département MES-IT regroupe ces deux aspects, en manipulant des logiciels comme *Wonderware ArchestrA* et en développant des applications à l'aide des technologies C# .NET et *WinDev*. Ci-dessous figurent quelques clients récurrents du département.



Source : <http://www.technord.com/fr/lignes-de-produits/automation-process-control-it-mes/mes>

Si la réalisation d'un travail représente une partie importante du stage, l'intégration d'une équipe de professionnels est un aspect tout aussi important. Du point de vue humain, l'équipe du département MES-IT est très sympathique et s'est révélée facile à intégrer. D'une manière générale, l'ambiance à Technord Liège était assez conviviale, les employés n'hésitant pas à se retrouver après leur journée de travail.

Au niveau des heures de travail, les horaires étaient relativement flexibles, permettant par exemple d'arriver et partir un peu plus tôt ou plus tard, pour autant que le nombre d'heures total soit respecté. Cette flexibilité permet de travailler dans des conditions optimales et de s'adapter aux imprévus de la vie quotidienne.

Chapitre 3

Cahier des charges

3.1 – Description générale

ArchestrA est une plate-forme complète d'intégration d'applications d'automation. Cet outil de Wonderware permet la gestion d'un système industriel, sous la forme de conteneurs de données, scripts, alarmes (déclencheurs), ...

En revanche, il ne permet ni la réalisation de tâches d'ajout/modification de données par lots, ni la génération de rapports. Par ailleurs, la gestion des exports de données y est calamiteuse et ne permet pas l'apport de modifications dans un fichier exporté.

Wonderware fournit une interface de programmation nommée GAccess. Elle permet aux développeurs de créer, en C#, en VB.NET ou en C++, des applications d'accès et de manipulation des données d'un projet ArchestrA.



L'application à réaliser, nommée Galaxy Toolkit¹, devait comprendre un ensemble d'outils permettant d'automatiser les tâches de création, modification et export de données d'une *galaxie*² ArchestrA, et de générer facilement des rapports Word et Excel sur base de données spécifiques. L'application devait être développée en utilisant le framework .NET et le langage C# (avec WinForms), ainsi que GAccess en version 2012 R2.

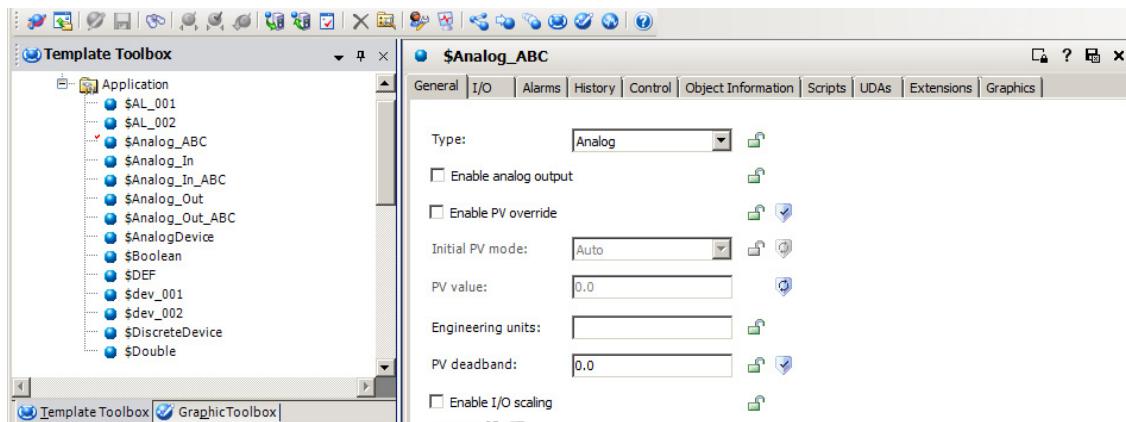
Le public cible de Galaxy Toolkit est constitué des employés de Technord chargés de la gestion et du développement d'applications ArchestrA. L'application sera donc utilisée exclusivement en interne, au sein de l'entreprise.

¹ Source de l'image : cahier des charges original du projet.

² Galaxie : projet de la plate-forme ArchestrA, regroupant toutes les données, paramètres et scripts du projet.

3.2 – Objectifs principaux

- Une fenêtre de login (similaire à celle d'ArchestrA IDE) permettant de se connecter à une *galaxie*¹ d'ArchestrA. Elle est accompagnée d'un message d'attente ou d'erreur.
- Une fenêtre principale partagée en deux zones verticales :
 - sur la gauche, un arbre peuplé automatiquement reprenant les différents *templates*² de base et leurs enfants ;
 - la partie droite, contenant une zone contextuelle (liste et autres outils).



- Une liste reprenant les *instances*³ existantes, pour un ensemble de *templates* sélectionnés dans l'arbre (multi-sélection).
- Un outil de recherche de *templates* et *instances*, autorisant l'encodage partiel des noms, ainsi qu'un « joker » (caractère %). Le nombre de résultats est indiqué.
- Possibilité de naviguer (via un double clic ou un bouton) depuis un ou plusieurs résultat(s) de recherche : développer l'arbre pour pointer vers le(s) résultat(s) sélectionné(s).
- Une fenêtre permettant l'affichage des attributs *UDAs*⁴ des *templates* ou *instances* sélectionné(e)s. La sélection et la suppression des *UDAs* propres à un objet (mais pas ceux hérités des objets parents) sont possibles.
- Permettre l'export d'*instances* et de leurs attributs *UDAs* sous forme de document Excel (voir point 3.3.1). Le document peut être édité hors de l'application, puis importé dans celle-ci, dans le but d'ajouter et modifier des données par lots.
- Permettre la génération de rapports Word au sujet des *templates* et de rapports Excel à propos des mesures réalisées dans un projet et des alarmes (déclencheurs) d'une *galaxie* ArchestrA (voir points 3.3.2/3.3.3).
- Générer un fichier d'export CSV compatible avec l'IDE ArchestrA. Le fichier pourra ensuite être importé directement dans l'IDE ArchestrA. Respecter la même structure que celle des fichiers exportés via l'IDE ArchestrA.

¹ Galaxie : projet de la plate-forme ArchestrA, regroupant toutes les données, paramètres et scripts du projet.

² Template : structure de définition d'éléments, similaire à une classe dans un langage orienté objet.

³ Instance : élément concret dérivé d'un template, similaire à un objet instancié dans un langage orienté objet.

⁴ UDA : attribut personnalisé (données additionnelles) d'un template ou d'une instance (User Defined Attribute).

3.3 – Documents et rapports générés

3.3.1 - Import et export de données

Les documents générés pour l'import/export d'*instances* sont des documents Excel (*.xlsx).

A	B	C	D
1			
2	Infos		
3	DATE CREATION	23/03/2016 15:03:02	
4	DATE MODIFICATION	23/03/2016 15:03:02	
5	TEMPLATES		277
6			
7			
8	[Import]	Templates	Instances
9	x	\$AADWGenBoutonsB	0
10	x	\$alINDEFF_Refreshe	1
11	x	\$alINDEFF_UserDefin	0
12	x	\$Alimentateur	3
13	x	\$Alveolaire	14

Un onglet *GÉNÉRAL* fournit des informations sur le document (date, nombre de *templates*, ainsi que la liste de tous les *templates* parents des *instances* exportées (voir ci-contre).

La liste générale comprend le nom de chacun des *templates*, suivi du nombre d'*instances* de ceux-ci. Devant chaque élément, une colonne permet d'indiquer (pour un import ultérieur) si le *template* doit être importé ou ignoré.

Un onglet est créé pour chaque *template*, listant les différentes *instances* descendant de celui-ci.

A	B	C	D	E	F	G	H	I	J
[Import]	Instances	Area	Container	Contained Name		Description	UDA	AL2O3	
x	FR1_CHAUX_10_50	FR1_Rapports				Matière 10 50	Locked	0	
x	FR2_CHAUX_10_50	FR2_Rapports				Matière 10 50	Security	1	
x	FR3_CHAUX_10_50	FR3_Rapports				Matière 10 50	DataType	Float	
							Is Array	false	
								0,0	
								0,0	
								0,0	

Une fois encore, chaque ligne est précédée d'une colonne pour l'import (indiquant si l'élément est importé ou ignoré). Chaque nom d'*instance* est suivi des données de cette dernière :

- zone concernée (*area*) ;
- conteneur (*container*) ;
- nom du contenu (*contained name*) ;
- description.

Si l'une de ces données était modifiée dans le document, elle serait mise à jour en cas d'import. Si de nouvelles *instances* étaient ajoutées au document, elles seraient insérées dans ArchestrA.

Les attributs *UDAs* variant d'un *template* à l'autre, ils sont listés au-dessus de la liste des *instances* d'un *template*, ainsi que leurs informations :

- nom d'attribut (*UDA*) ;
- verrou (*locked*) ;
- politique de sécurité (*security*) ;
- type de donnée (*data type*) ;
- dimension : tableau ou non (*is array*).

3.3.2 - Rapports de type « template list »

Ce type de rapport est généré au format Microsoft Word (*.docx). Des styles visuels similaires à ceux de l'exemple sont utilisés. Le document comprend une liste de *templates*, ainsi que leurs informations détaillées :

- une section pour chaque *template*, portant son nom ;
- description ;
- nom du *template* parent (*derived from*) ;
- tableau des attributs *UDAs* :
 - nom (*UDA name*),
 - type de données (*data type*),
 - les extensions activées (*input-output*, *input*, *output*, *alarm*, *history*) ;
- tableau des *field attributes*¹ :
 - nom (*field attribute name*),
 - type de données (*data type*),
 - les fonctionnalités activées (*IO scaling*, *limit alarms*, *rate of change alarms*, *target deviation alarms*, *bad value alarm*, *statistics*) ;
- liste des scripts :
 - nom du script,
 - tableau informatif (*name*, *description*, *trigger*),
 - déclarations du script,
 - code exécuté par le script.

 Solutions flexibles. Expertise solide.	Project: RX59	Site: Place:	Place: TECHNORD : 73145.MES-SDS
Document: Software Design Specification			Page 36 of 603
Customer Reference: fulldoc			

1.2.1.28 *Template Archestra SdwClapetMotoriseIS3P*

1.2.1.28.1 *Description*
Clapet motorisé

1.2.1.28.2 *Derived from*
\$dwActionneur

1.2.1.28.3 *UDA's*

UDA name	Data Type	IO	I	O	A	H
bAlarmeDefAU	boolean		X		X	
bAlarmeDefIS	boolean		X		X	
bAlarmeDefMTH	boolean		X		X	
bAlarmeDefFRKM	boolean		X		X	
bAlarmeDefTemp	boolean		X		X	
bAlarmeDisc2Fdc	boolean		X		X	
bAlarmePosition1	boolean		X		X	
bAlarmePosition2	boolean		X		X	
bAlarmePosition3	boolean		X		X	
bCommandeFermetureManu	boolean		X			
bCommandeInhibitionDefIS	boolean		X			
bCommandeInhibitionDefMTH	boolean		X			
bCommandeInhibitionDefFRKM	boolean		X			
bCommandeInhibitionDefTemp	boolean		X			
bCommandeInhibitionDisc2FDC	boolean		X			
bCommandeInhibitionDiscPosition1	boolean		X			
bCommandeInhibitionDiscPosition2	boolean		X			
bCommandeInhibitionDiscPosition3	boolean		X			
bCommandeOuvertureManu	boolean	X				

¹ Field attribute : attribut (d'un template ou d'une instance) correspondant à une grandeur concrète mesurable, analogique (décimale) ou discrète (booléenne).

3.3.3 - Rapports de type « measurement list » et « alarm list »

Ces deux types de rapports sont générés au format Excel (*.xlsx). Des styles visuels similaires à ceux des exemples sont utilisés.

- Le rapport de type « measurement list » comprend des informations sur les valeurs mesurées du procédé de fabrication. Pour chaque *instance*, les colonnes suivantes sont créées :

- nom de l'*instance* suivi du nom de mesure (*tag A2*) ;
- zone concernée (*area*) ;
- nom d'identification *Intouch*¹ ;
- description de la mesure ;
- valeur d'un attribut d'adressage du capteur de mesure (choisi lors de l'export) ;
- unités de la mesure (*eng units*) ;
- limites (*min EU, max EU, min raw, max raw, raw type, low limit, high limit*) ;
- valeurs diverses (*force storage period, value deadband*) ;
- des colonnes *package* et *topic name* vides (complétées manuellement).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
		 TECHNORD Solutions flexibles. Expertise solide	Concern: <i>List Measurement</i> <i>PDM</i> <i>Migration Supervision RX</i>	Document: <i>Automation-Functional-Design-000002079</i>	Project:	Site:	Place:									
			Customer Reference: <i>Automation-Functional-Design-0000020792</i>													
Name (A2)	Area A ²	Name (Intouch)	Description	ItemName	TopicName	EngUnits	MinEU	MaxEU	MinRaw	MaxRaw	RawType	Low Limit	High Limit	Force Storage Period	Value Deadband	Package
04.PV	RX59_F103	AE0004	F103: Oxygène	W8040_S	ACCESSPLC1P	%	0	100	0	10000	Linear	0,00	40,00	300000	0,1	FEPR
04.PV	RX59_F104	AE0104	F104: Oxygène	W8048_S	ACCESSPLC1P	%	0	100	0	10000	Linear	5,00	45,00	300000	0,1	FEPR
04.PV	RX59_F203	AE0204	F203: Oxygène	W8040_S	ACCESSPLC2P	%	0	100	0	10000	Linear	5,00	45,00	300000	0,1	FEPR
04.PV	RX59_F204	AE0304	F204: Oxygène	W8048_S	ACCESSPLC2P	%	0	100	0	10000	Linear	5,00	45,00	300000	0,1	FEPR
04.PV	RX59_F102	AE0404	F102: Oxygène	W8032_S	ACCESSPLC1P	%	0	100	0	10000	Linear	5,00	45,00	300000	0,1	FEPR
04.PV	RX59_F202	AE0504	F202: Oxygène	W8032_S	ACCESSPLC2P	%	0	100	0	10000	Linear	5,00	45,00	300000	0,1	FEPR

- Le rapport de type « alarm list » contient des informations sur les *alarmes*² présentes au sein d'une galaxie (ces informations peuvent figurer dans plusieurs types d'attributs et extensions). Pour chaque élément, les colonnes suivantes sont créées :

- nom de l'*instance* suivi du nom d'attribut ou extension (*tag A2*) ;
- zone concernée (*area A2*) ;
- valeur d'un attribut d'adressage du capteur surveillé (choisi lors de l'export) ;
- statut de l'alarme (*active state*) ;
- priorité (*priority*) ;
- description ;
- des colonnes *package*, *topic name* et *remarks* vides (complétées manuellement).

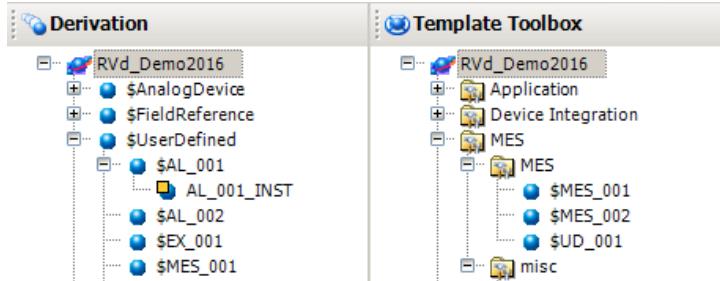
A	B	C	D	E	F	G	H
		 TECHNORD Solutions flexibles. Expertise solide	Concern: <i>List Alarm</i> <i>PDM</i> <i>Migration Supervision RX 1504</i>	Document: <i>Automation-Functional-Design-000002079</i>	Project:	Site:	Place:
			Customer Reference: <i>Automation-Functional-Design-0000020791</i>				
Tag A2	PLC Address	AccessName	Area A2	Active State	Priorit	Description	Package
T901_AL_01.Alarm_10	B1765	ACCESSPLC1P	RX59_BoucleEpiT901	On	500	BoucleEpiT901:EA72561-Coupe M.T Ppt BoucleEpiT901	
T901_AL_01.Alarm_11	B954	ACCESSPLC1P	Rx59_BoucleEpiT901	On	500	EPITocT901:EpiPal_AT_M3_Alm - Invalidit BoucleEpiT901	
T901_AL_01.Alarm_12	B973	ACCESSPLC1P	Rx59_BoucleEpiT901	On	500	EPIToc_AMS_Alerte :EpiPal_AT_M3_AM BoucleEpiT901	
T901_AL_01.Alarm_13	B974	ACCESSPLC1P	Rx59_BoucleEpiT901	On	500	EPIToc_AMS_Action :EpiPal_AT_M3_AM BoucleEpiT901	
T901_AL_01.Alarm_14	B955	ACCESSPLC1P	Rx59_BoucleEpiT901	On	500	EPITocT901:EpiPal_AT_M3_ASHAlm - Al BoucleEpiT901	

¹ Intouch : partie d'ArchestrA dédiée à la supervision et à la création de représentations graphiques du projet.

² Alarme : procédé paramétrable, déclenché lorsqu'une condition est remplie durant l'exécution d'un projet.

3.4 - Objectifs secondaires

- Choix de visualisation de l'arbre : un onglet avec un arbre suivant la hiérarchie et l'héritage (comme l'arbre « Dérivation » d'ArchestrA IDE) et un onglet avec un arbre de classement par dossiers (similaire à l'arbre « Toolbox » d'ArchestrA IDE).



- Les icônes des nœuds de chaque arbre indiquent le statut des éléments, tel que cela est présenté dans ArchestrA : check-in (prêt), check-out (occupé), erreurs, avertissements.
- Possibilité de sauvegarder le déploiement et la sélection courante des arbres dans un fichier texte éditable facilement (snapshot). Possibilité de restaurer l'état des arbres à partir d'un fichier snapshot (pour une même *galaxie*). Repérer les erreurs éventuelles.
- Plusieurs colonnes présentes dans la liste des *instances* existantes, reprenant des informations sur les éléments. Possibilité de trier/filtrer les résultats selon les colonnes.
- Une barre de statut reprenant des informations : nom du serveur et de la *galaxie*, nombre de *templates*, nombre d'*instances* pour la sélection, ...
- Une boîte d'options permettant une configuration persistante de l'application (fichier sauvegardé aux côtés de l'exécutable) : colonnes affichées dans les listes, utilisation automatique de snapshots lors de l'ouverture et la fermeture, choix des optimisations, ...
- Réaliser la génération des rapports de *templates* (Word) et de mesures et alarmes (Excel) à partir de modèles de documents liés (comprenant un en-tête et une image).
- S'assurer que l'application ne nécessite pas de programme d'installation (en dehors de celui de GRAccess). Faire en sorte que les fichiers du programme soient « standalone ».
- Un utilitaire de journalisation (log) fournissant des informations sur les erreurs au format CSV (tableau consultable dans Excel).

Chapitre 4

Analyse

La phase d'analyse représente une partie aussi importante que celle de développement. Trop souvent sous-estimée, elle permet d'assurer la cohérence, la facilité de maintenance et l'évolutivité du projet. Elle constitue également une étape essentielle pour assurer sa modularité et, par extension, la réutilisabilité de ses composants.

Dans un premier temps, il est important de comprendre et de décrire de manière précise les besoins du client. L'analyse des besoins doit non seulement cerner les fonctionnalités à mettre en œuvre, mais également la manière dont elles seront utilisées. Les besoins définis seront validés par le client. Cette étape a pour but d'assurer sa satisfaction et d'éviter les surcoûts engendrés par des objectifs mal définis.

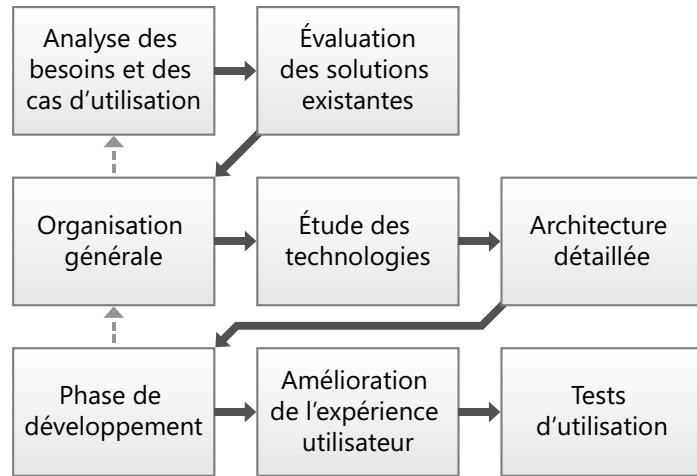
Pour réaliser la phase d'analyse, des conventions seront utilisées. UML (*Unified Modeling Language*) offre un standard de modélisation du système, des besoins et de l'architecture logicielle. Il fait partie des notations les plus utilisées de nos jours. Les diagrammes UML s'articulent autour d'une philosophie orientée objet. Ils conviennent donc parfaitement pour travailler avec un langage tel que le C#.

Il est rarement possible de définir chaque aspect d'un projet à l'avance. L'évolution des objectifs, les contraintes techniques rencontrées ou encore l'apport de nouvelles idées sont autant de raisons de recommencer plusieurs fois l'analyse, voire même de revenir à la phase d'analyse en plein développement. Pour les projets de grande envergure, il peut aussi être préférable de découper l'analyse en plusieurs parties. Il est donc souvent nécessaire d'adopter une démarche itérative.

Une démarche itérative consiste à effectuer des allers-retours entre le plan initial et l'évolution de celui-ci. La modélisation d'un logiciel est rarement satisfaisante dès la première esquisse. Le passage par plusieurs itérations permettra d'affiner l'analyse pour aboutir à une version définitive. Au cours du développement, si des difficultés inattendues sont rencontrées, il est également préférable de revenir en arrière et de réorganiser le travail, afin de s'assurer de sa qualité, de sa fiabilité et de sa facilité de maintenance.

Tout au long du cycle de développement, les besoins des utilisateurs doivent rester déterminants, que ce soit pour la clarification des objectifs, pour l'affinage de l'analyse ou encore pour la vérification de la cohérence durant la conception. Ils trouvent également leur utilité lors de la phase de test, afin de garantir la satisfaction des utilisateurs.

La mise en œuvre du projet réalisé au cours de mon stage a nécessité différentes étapes. Le fait que les objectifs m'aient été communiqués peu à peu a exigé plusieurs retours à la phase d'analyse. Une démarche itérative a donc été adoptée. Les différentes étapes sont illustrées ci-dessous.



L'analyse des besoins a déjà été partiellement décrite au cours du chapitre 3, où les différentes consignes reçues ont été regroupées sous forme d'objectifs principaux et secondaires. Elle est complétée par l'étude des cas d'utilisation, présente dans ce chapitre. La définition de l'organisation générale et de l'architecture logicielle détaillée, ainsi que l'étude des technologies, représentent la majeure partie de ce chapitre.

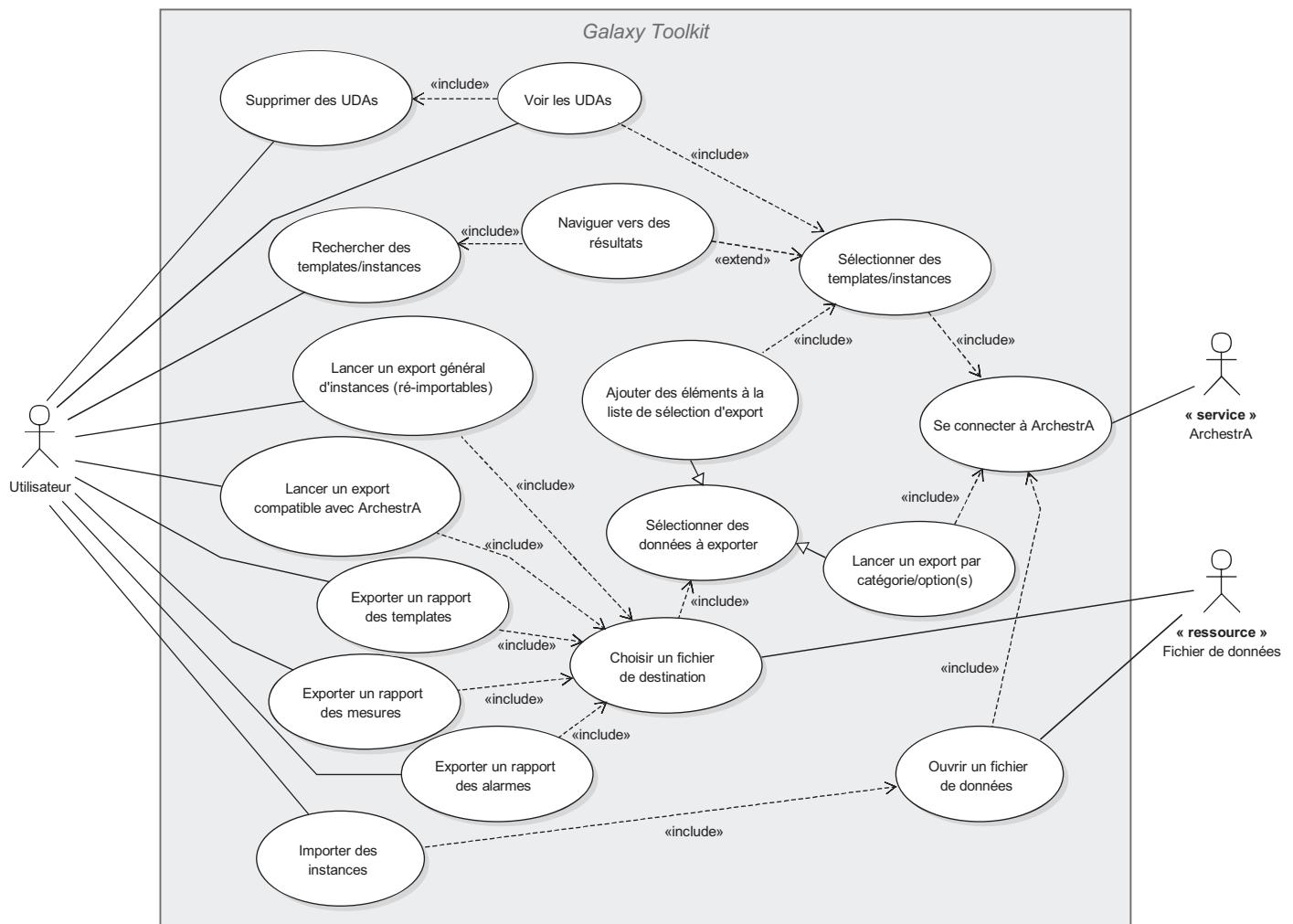
La description du développement, l'amélioration de l'expérience utilisateur (design et ergonomie) et les tests d'utilisation (ergonomie) sont abordés dans le chapitre 5.

Le contenu de ce chapitre est divisé en plusieurs parties :

- **Cas d'utilisation** : définition des principaux usages de l'application ;
 - **Évaluation de l'existant** : analyse des solutions existantes et de ce qu'elles pouvaient apporter ;
 - **Exploration des technologies** : étude détaillée de chaque nouvelle technologie utilisée ;
 - **Organisation du projet** : description de l'architecture logicielle du projet.

4.1 – Cas d'utilisation

La première étape de l'analyse consiste à définir les besoins du client. Après avoir ciblé les différents objectifs principaux et secondaires, il est intéressant de déterminer les actions qui en découlent lors d'une utilisation. Le diagramme UML des cas d'utilisation remplit ce rôle. Chaque cas d'utilisation décrit un scénario qui définit la façon dont le système devrait interagir avec les utilisateurs (appelés acteurs) pour atteindre un but spécifique.



4.2 – Évaluation de l’existant

L’évaluation des solutions existantes consiste à analyser les produits similaires. Elle permet dans un premier temps de déterminer la nécessité ou non du nouveau projet. Elle se révèle ensuite utile pour constater les avantages et les inconvénients des produits existants, pour s’en inspirer et pour définir ce qui peut être amélioré. Le but est de réaliser un meilleur produit que ceux déjà disponibles, ou un produit plus ciblé.

4.2.1 - aaExport – projet open source

Description

L’application aaExport a pour but de réaliser différents types d’exports de projets ArchestrA. Elle s’exécute en ligne de commande et génère des fichiers au format texte (*.txt). Elle utilise l’API GAccess pour communiquer avec ArchestrA.

Source du projet : <https://github.com/aaOpenSource/aaExport>

Observations

Cette application utilise la même API (GAccess) que celle de mon projet. Son code permet donc de fournir des exemples de la manière dont fonctionne GAccess.

Malheureusement, aaExport ne remplit pas les mêmes objectifs que mon projet. Les exports réalisés n’ont presque rien en commun et aucune fonctionnalité d’import n’a été prévue. De plus, l’application ne dispose d’aucune interface graphique et sa gestion des données est archaïque.

Conclusions

Le projet aaExport était utile pour se familiariser avec les concepts de base de l’API GAccess. Il m’a permis de gagner du temps lors de la phase d’étude de GAccess et de réaliser quelques tests. Cependant, aucune portion de code n’était réutilisable pour mon projet car les objectifs recherchés n’étaient pas les mêmes.

4.2.2 - CreateGalaxy – ancien projet interne

Description

CreateGalaxy est un ancien projet réalisé par la société Technord. Il offre comme seule fonctionnalité l'import et export d'*instances* d'ArchestrA. Il s'exécute en ligne de commande et génère des fichiers au format CSV (éditables dans Excel). Il utilise l'API GRAccess pour obtenir des données exportées depuis ArchestrA et pour y écrire des données importées.

Observations

Cette application utilise la même API (GRAccess) que celle de mon projet. Son code permet donc de fournir des exemples de la manière dont fonctionne GRAccess. Elle remplit l'un des objectifs de mon projet : un import et export purement fonctionnel de données d'ArchestrA.

Malheureusement, l'application n'est pas fiable. Aucune vérification des données importées n'est réalisée (aboutissant à un échec en cas d'erreur d'encodage) et aucune gestion des erreurs n'est mise en place (ce qui peut corrompre toute la base de données ArchestrA en cas d'erreur). Par ailleurs, aucun paramétrage de l'export et aucune interface graphique ne sont disponibles.

Au niveau architecture logicielle, aucune organisation du code n'a été réalisée : l'entièreté du code de l'application se situe dans la fonction principale (*main*). De plus, le code est mal structuré et ne dispose d'aucun commentaire.

Conclusions

Le projet CreateGalaxy ne remplissait un rôle similaire qu'à une seule des fonctionnalités qui m'ont été demandées, mais il m'a néanmoins permis de me familiariser avec GRAccess, en particulier pour comprendre la manière dont sont traités les tableaux de données. Cependant, la nature du code et son manque de rigueur et de fiabilité n'ont pas permis de le réutiliser.

4.3 – Exploration des technologies

4.3.1 - Wonderware ArchestrA¹

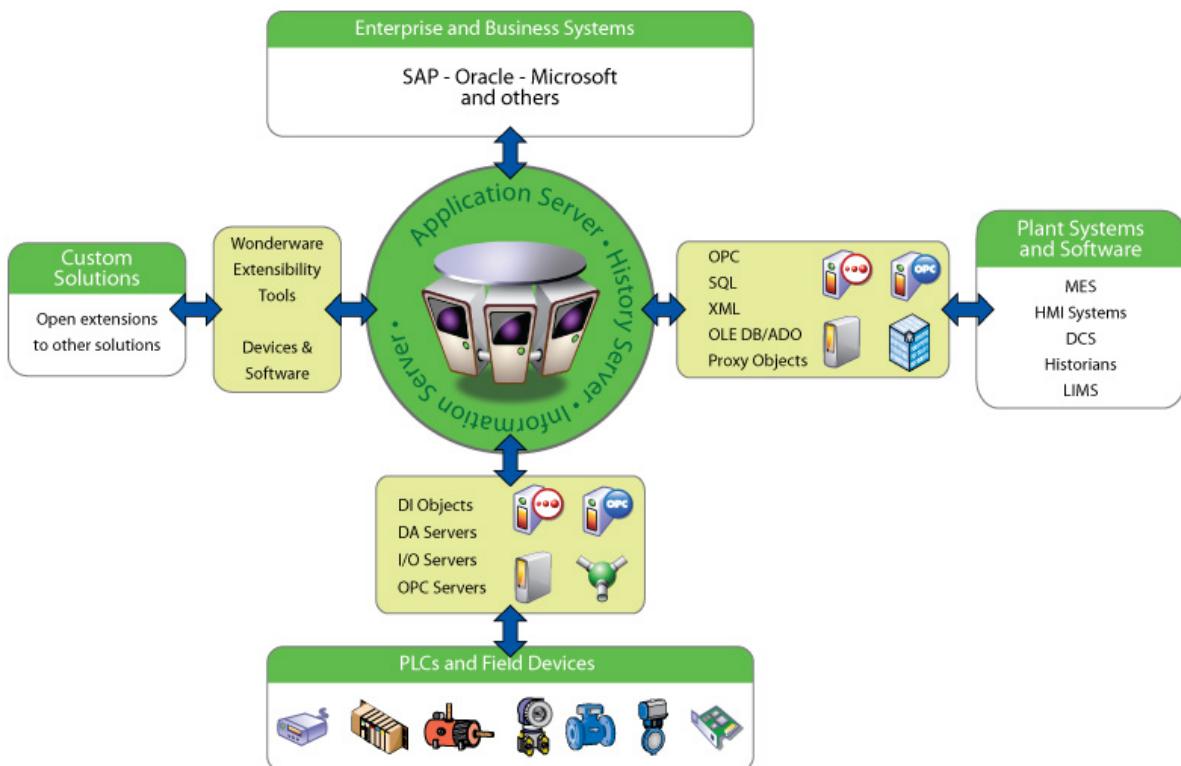
Introduction à ArchestrA



ArchestrA est une plate-forme complète, permettant l'intégration d'applications d'automation et de supervision de systèmes industriels, conçue pour gérer aussi bien les anciens systèmes que les dernières technologies.

L'approche modulaire de l'architecture ArchestrA rend les systèmes évolutifs, en permettant par exemple l'ajout d'extensions. La flexibilité offerte facilite l'intégration de systèmes, logiciels et matériels provenant de n'importe quel fournisseur. Un ensemble de services constituent l'infrastructure ArchestrA. Ceux-ci permettent aux concepteurs d'applications d'automation de faire abstraction de la complexité des technologies manipulées, en ne nécessitant que des compétences d'assemblage et non de programmation proprement dite.

Mise en évidence de l'approche modulaire d'ArchestrA



Source : <http://global.wonderware.com/IT/Pages/WonderwareSystemPlatform.aspx>

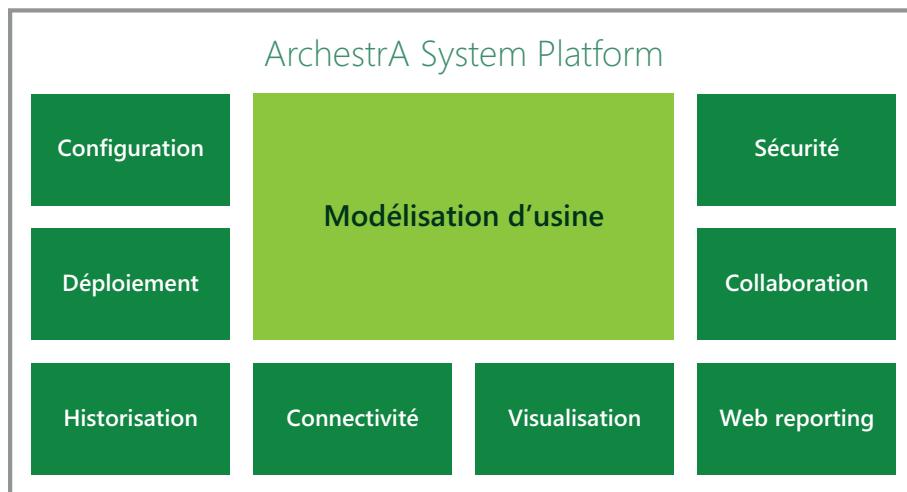
Cette approche permet de réaliser des applications rapidement et de simplifier leur réutilisabilité, grâce à la nature abstraite des données. La découpe des différents services facilite également le travail collaboratif, en leur permettant de travailler en parallèle sur un projet.

¹ Source du logotype : <https://en.wikipedia.org/wiki/Wonderware>, 19/05/2016

ArchestrA System Platform

ArchestrA System Platform est le noyau central d'ArchestrA. Cette plate-forme inclut l'ensemble des services élémentaires à un système d'information industriel : modélisation, exécution, visualisation, analyse, historisation et optimisation. Elle permet la gestion de tâches de supervision, acquisition de données, gestion de workflow¹ en temps réel, MES², ...

Le schéma qui suit offre un aperçu des services essentiels fournis par System Platform :



Source : <http://software.schneider-electric.com/pdf/brochure/wonderware-system-platform>

ArchestrA System Platform agit en tant que « système d'exploitation industriel » en fournissant ses différents services. Ceux-ci permettent de construire une « modélisation d'usine » unifiée, qui représente de manière logique les processus, équipements et systèmes industriels. La modélisation fournit également un contexte aux données, permettant leur diagnostic et leur correction, ainsi que l'enregistrement de mesures et la création de rapports d'historisation.

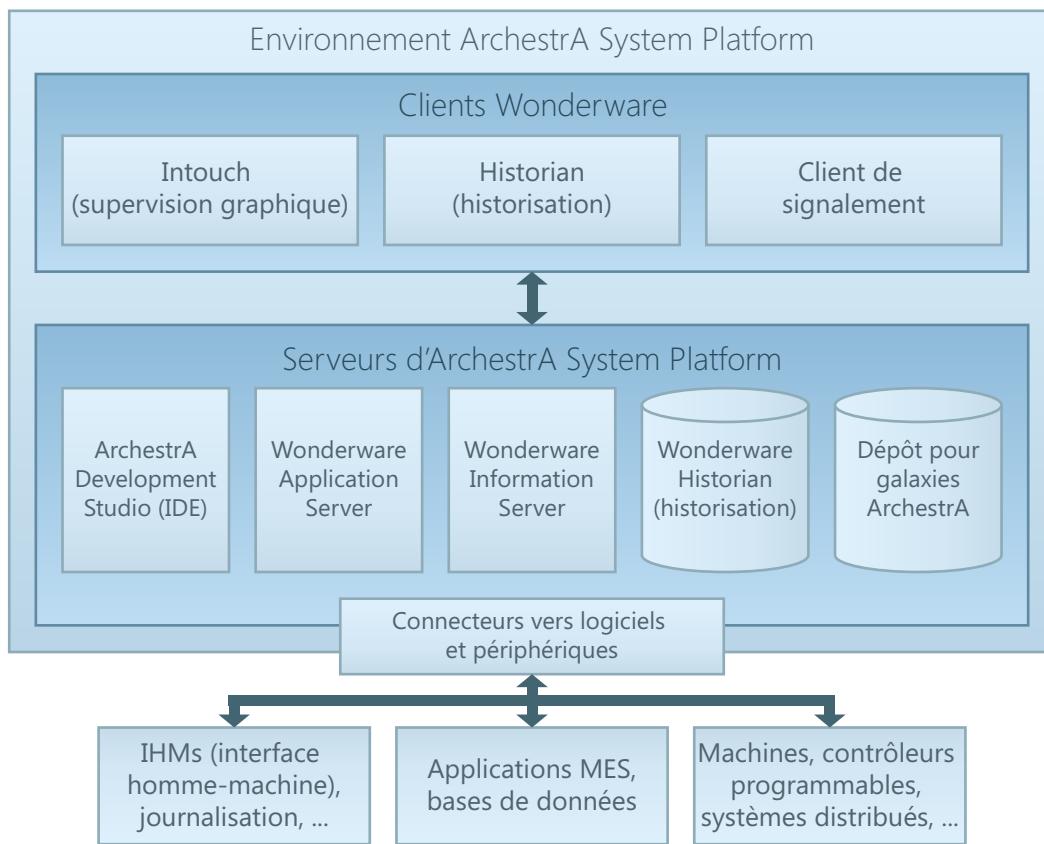
D'une manière générale, ArchestrA System Platform offre une standardisation des développements, ainsi qu'une réduction des coûts de maintenance et d'évolution. Il permet la collecte et l'archivage de données, quelle que soit leur source : automates, bases de données... Il assure enfin une organisation flexible, adaptée pour les projets de petite et de grande envergure.

¹ Workflow : représentation d'une suite de tâches ou opérations effectuées par une personne ou un système.

² MES (Manufacturing Execution System) : système informatique chargé de collecter en temps réel des données de production d'une usine (ou d'une partie de celle-ci).

Services ArchestrA utilisés pour mon projet

ArchestrA System Platform est une plate-forme constituée d'une multitude de services. Ceux-ci sont organisés sous forme de modules, listés sur le schéma qui suit :



Parmi les différents éléments de la plate-forme, seuls deux modules ont été utilisés pour le projet :

- **Galaxy Repository** (dépôt des projets ArchestrA) :

Il s'agit du moteur permettant l'accès aux données des différents projets ArchestrA. Il repose sur un ensemble de bases de données SQL Server. Chaque projet ArchestrA (appelé *galaxie*) possède sa propre base de données.

Les données comprises dans ce module sont celles utilisées par ArchestrA IDE, et par l'API GRAccess (présentée au point 4.3.2). Outre les accès indirects au moyen des deux outils précédents, un accès direct m'a également été utile pour réaliser des requêtes SQL.

- **ArchestrA IDE (Development Studio) :**

L'IDE d'ArchestrA permet la création et la configuration des données, attributs, scripts et alarmes au sein d'une « galaxie » (projet ArchestrA). Cet outil m'a permis de me familiariser avec le fonctionnement d'ArchestrA et de vérifier la cohérence des données par rapport à celles obtenues à l'aide de l'API GRAccess.

ArchestrA IDE m'a également permis de réaliser une série de tests, en y ajoutant ou modifiant préalablement des données, qui étaient ensuite récupérées via GRAccess.

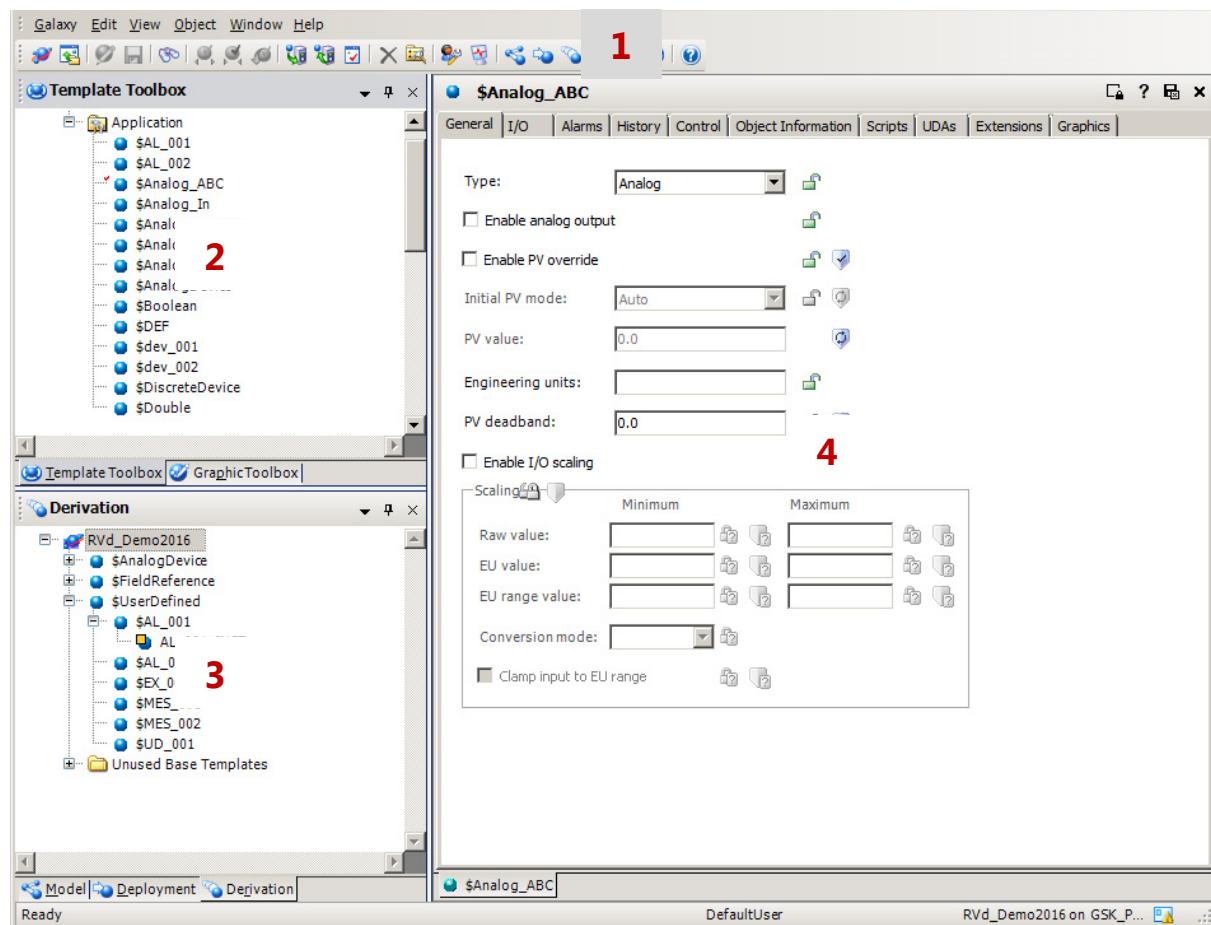
Présentation des données d'ArchestrA

L'organisation des données d'un projet ArchestrA peut sembler compliquée au premier abord. Chaque type d'élément y est en effet désigné par un nom bien particulier. Cette page comprend donc une explication des différents termes, afin de faciliter leur compréhension.

Galaxie	Une <i>galaxie</i> désigne un projet de la plate-forme ArchestrA. Elle regroupe un ensemble de <i>templates</i> et leurs <i>instances</i> , ainsi que d'autres types d'éléments : serveur(s) d'application, vues (représentations graphiques du système modélisé) et autres objets de configuration et de supervision. Chaque <i>galaxie</i> dispose de sa propre base de données SQL Server.
Template	<p>Un <i>template</i> désigne un canevas de données, comparable à une classe dans un langage orienté objet. Il définit une structure, avec un ensemble d'attributs de base (description, type, ...), de <i>scripts</i>, d'<i>alarmes</i>, d'attributs additionnels (<i>field attributes</i>, <i>UDAs</i>) et d'<i>extensions</i> d'attributs.</p> <p>En dehors des <i>templates</i> préexistants (<i>\$AnalogDevice</i>, <i>\$UserDefined</i>, ...), tout <i>template</i> créé doit dériver d'un autre <i>template</i>. Il hérite ainsi de ses attributs et valeurs, tout en ayant l'opportunité d'en ajouter d'autres. Les <i>templates</i> dérivent donc tous, directement ou indirectement, de l'un de ceux de départ. Leurs attributs de base dépendent de celui-ci.</p>
Instance	Une <i>instance</i> représente un ensemble de données fonctionnelles, basées sur la structure d'un <i>template</i> . Elle est comparable à un objet instancié, dans un langage orienté objet. L' <i>instance</i> hérite des attributs et valeurs du <i>template</i> dont elle dérive, auxquels elle peut ajouter ses propres données.
Field attribute	Un <i>field attribute</i> est un attribut additionnel d'un <i>template</i> ou d'une <i>instance</i> . Il est possible d'en ajouter autant que nécessaire. Chaque <i>field attribute</i> représente une grandeur concrète mesurable. Il peut s'agir d'une grandeur analogique (valeur entière ou décimale) ou discrète (valeur booléenne). Un <i>field attribute</i> comprend différents paramètres : mode d'accès (lecture et/ou écriture), catégorie, description. Il peut également activer une série de fonctionnalités : <i>alarmes</i> (dépassement de limite, vitesse de variation, déviation, valeur rejetée), historisation et statistiques.
UDA	Un <i>UDA</i> (User Defined Attribute) est un attribut additionnel personnalisé d'un <i>template</i> ou d'une <i>instance</i> . Il est possible d'en ajouter autant que nécessaire. Sa simplicité apparente (choix d'un type de donnée, d'une dimension et d'une valeur) lui permet de représenter aussi bien une grandeur concrète, qu'une grandeur abstraite ou descriptive.
Script	Un <i>script</i> représente une unité de code exécutable d'un <i>template</i> ou d'une <i>instance</i> . Il est possible d'en ajouter autant que nécessaire. Chaque <i>script</i> comporte une expression de condition, un type de déclencheur pour cette condition (vraie, fausse, tant qu'elle reste vraie, ...) et une série de paramètres de temps et de priorité. Il contient également un ensemble de déclarations (variables) et une portion de code à exécuter.

Extension	<p>Pour chaque attribut (attribut de base, <i>field attribute</i> ou <i>UDA</i>), il est possible d'ajouter une série d'<i>extensions</i>: lecture d'une source et/ou écriture d'une destination, <i>alarme</i> (déclencheur personnalisé) et historisation (largement paramétrable).</p> <p>Grâce à ces extensions, il devient possible de compléter les fonctionnalités des différents attributs de manière plus ou moins importante et plus ou moins flexible. Par exemple, il est possible de fournir à un <i>UDA</i> les mêmes fonctionnalités que celles d'un <i>field attribute</i>. Pour cette raison, les <i>field attributes</i> tendent à disparaître des dernières versions d'ArchestrA.</p>
Alarme	<p>Une <i>alarme</i> représente un simple déclencheur. Elle s'active lorsqu'une condition particulière est remplie (dépassement de valeur, variation trop rapide, ...). Elle permet la mise en place d'un message ou d'un signal dès que sa condition est vérifiée.</p>

Organisation d'ArchestrA IDE



L'IDE d'ArchestrA se divise en quatre zones. Chacune de ces zones remplit un rôle précis :

- **1 – La barre de menu :**

La barre de menu comprend une série d'outils remplissant divers rôles : gestion de données, contrôle de validité, sauvegarde, recherche, ...

- **2 – Un arbre de classement de *templates* :**

Les arbres situés dans cette zone ne contiennent que des *templates*. Ils permettent d'établir un classement personnalisé des *templates* dans des dossiers. Cela permet d'établir un classement fonctionnel, qui peut trouver son utilité, lorsque des changements sont apportés.

- **3 – Un arbre de données :**

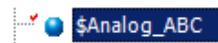
Les arbres situés dans cette zone contiennent toutes sortes d'éléments : *templates*, *instances*, serveurs applicatifs, vues, ... Tout y est mélangé. L'organisation des données de chaque arbre respecte une logique précise : hiérarchie de dérivation, état de déploiement, ...

- **4 – La zone contextuelle :**

Cette zone apparaît lorsqu'un élément de l'un des arbres est ouvert. Elle contient une série d'onglets qui varient suivant la catégorie de l'élément. Chaque page regroupe un ensemble de champs, de paramètres et de listes (attributs, extensions, ...). C'est ici que la majorité des valeurs sont créées ou éditées. La zone contextuelle fournit notamment des outils pour gérer les *field attributes*, les *UDAs*, les *scripts* et les *alarms*.

Gestion des accès aux données

Chaque fois qu'un élément (*template*, *instance*, ...) est ouvert au sein d'ArchestrA, un indicateur rouge apparaît à côté de son nom. Il signifie que l'élément est en cours de modification. Ce système est comparable aux exclusions mutuelles (*mutex*) : l'élément est verrouillé, jusqu'à ce que la modification soit terminée ou annulée.

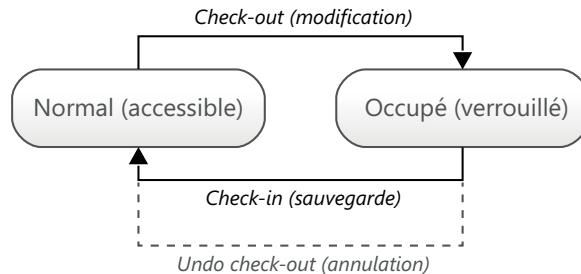


Un objet ArchestrA peut posséder deux états : normal ou verrouillé. Lorsque quelqu'un ouvre un élément pour le modifier, il réalise un *check-out*. Cela signifie qu'il verrouille l'élément pour y apporter des modifications. Personne d'autre ne pourra modifier cet élément, tant qu'il n'aura pas été déverrouillé. Si l'IDE ArchestrA est fermé sans que l'élément ait été libéré, il conservera son état, jusqu'à ce que la personne l'ayant verrouillé termine sa transaction.

Pour déverrouiller un élément, il existe deux possibilités :

- réaliser un *check-in*, c'est-à-dire sauvegarder ses modifications ;
- annuler le *check-out*, ce qui revient à annuler les changements apportés.

Diagramme d'états-transitions des objets ArchestrA



Conclusions

ArchestrA est une plate-forme d'intégration d'applications d'automation et supervision complète et assez complexe. Le logiciel comprend une multitude de services lui assurant une grande flexibilité et une compatibilité avec la majorité des dispositifs industriels, qu'ils soient récents ou anciens. L'un des buts d'ArchestrA est de permettre la modélisation complète d'un système industriel, afin d'assurer son exécution, son analyse et sa supervision, tant sur le plan visuel qu'au niveau de l'historisation.

Cependant, l'IDE d'ArchestrA présente des limites. Son utilisation ne permet pas de réaliser des modifications par lots, nécessitant donc d'appliquer chaque changement élément par élément. La gestion de plusieurs éléments à la fois y est archaïque. Les *instances* sont visibles uniquement dans des arbres où elles sont mélangées à toutes sortes d'autres objets, ce qui rend leur recherche et leur sélection fastidieuses. En termes d'ergonomie et d'utilisabilité, bon nombre d'aspects de l'IDE pourraient être améliorés.

4.3.2 - GRAccess – API d'ArchestrA

Introduction à GRAccess

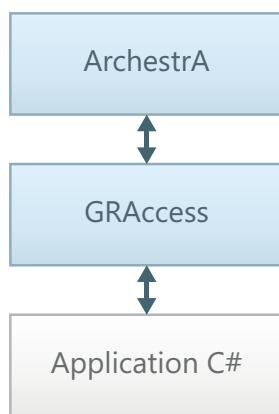


GRAccess Toolkit fournit un accès aux *galaxies* du serveur ArchestrA depuis l'extérieur. Il permet d'écrire ses propres programmes pour automatiser leur lecture, leur configuration et leur modification. Il fonctionne avec les langages de programmation C#, C++ et VB.NET.

Source de l'image :

http://www.pantek.cz/pdf/produkty/ias/archestra/archestra_faq.pdf

GRAccess assure l'interfaçage entre ArchestrA et les applications qui l'utilisent. Plusieurs méthodes permettent l'obtention, l'ajout et le retrait de données. Le contexte des données récupérées permet aussi leur modification de manière directe. Ce dernier aspect est détaillé dans la section suivante.

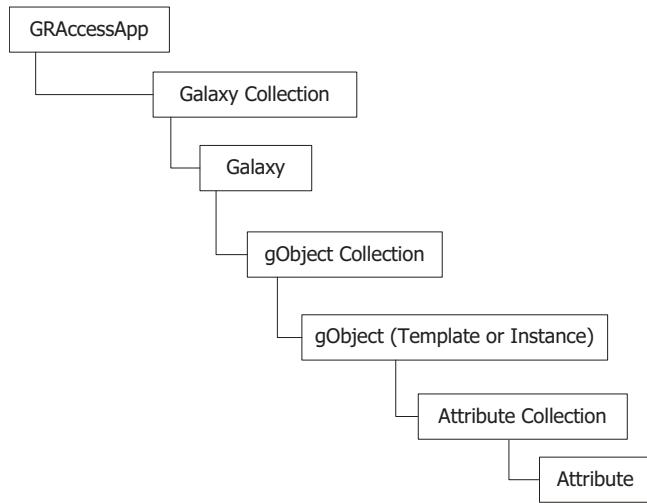


L'utilisation de *GRAccess Toolkit* nécessite une licence de développeur ArchestrA. L'application qui s'en sert doit être exécutée avec des droits d'administrateur, faute de quoi le chargement de *GRAccess* échoue.

Modèle de données hiérarchique

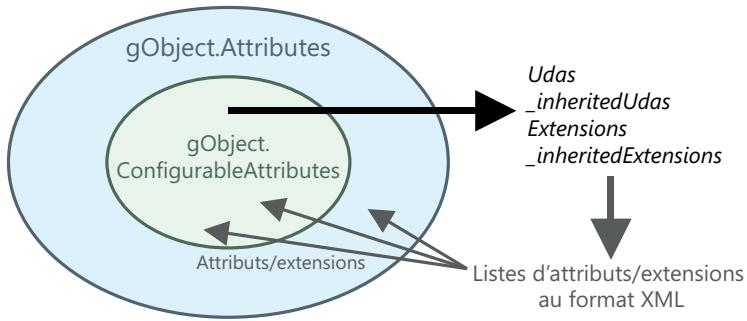
GRAccess fournit un modèle de données hiérarchique. À la racine de ce modèle se trouve l'objet *GRAccessApp*. Celui-ci procure un accès à ArchestrA et à l'ensemble de ses *galaxies*. Cet objet constitue la base de l'utilisation de GRAccess : tous les autres éléments manipulés découlent de lui. Ses enfants directs représentent les *galaxies* d'ArchestrA. L'accès à l'une d'entre elles peut être établi, à condition de fournir un nom d'utilisateur et un mot de passe valide.

Une fois la *galaxie* ouverte, l'ensemble de ses données sont accessibles sous forme d'objets *gObject*. Le seul moyen d'accéder à ces éléments est d'appeler des fonctions de la *galaxie*. Ces fonctions permettent d'exécuter des requêtes selon une condition ou par nom d'élément. Un ensemble de résultats *gObject* est ensuite accessible.



Source : *GRAccess Toolkit API User's Guide*

Chaque objet représente un *template* ou une *instance*. Il renferme une série de variables membres : nom d'identification, nom hiérarchique, nom de l'objet parent, zone/area, conteneur, catégorie, état actuel, erreurs et avertissements, ... En dehors de ces valeurs élémentaires, l'objet contient aussi deux listes d'attributs : *Attributes* et *ConfigurableAttributes*. Quelques tests m'ont permis de réaliser que la seconde liste constitue un sous-ensemble de la première : tous les éléments présents dans *ConfigurableAttributes* appartiennent aussi à *Attributes*. L'inverse n'est pas toujours vrai.



La liste des attributs est en vérité un dictionnaire¹ de données : chaque attribut est identifié par une clé unique, qui correspond à son nom. Ce dictionnaire d'attributs mélange sans aucune distinction des attributs généraux, des *field attributes*, des *UDAs*, des *extensions*, des *scripts* et des *alarmes*. Toutefois, il contient aussi de faux « attributs », qui permettent d'obtenir des renseignements au sujet des « vrais » attributs. Par exemple, les « faux » attributs *Udas*, *_inheritedUdas*, *Extensions* et *_inheritedExtensions* fournissent respectivement des renseignements à propos des *UDAs* propres à l'objet, des *UDAs* hérités, des *extensions*, *scripts* et *alarmes* propres à l'objet, et enfin des *extensions*, *scripts* et *alarmes* hérité(e)s.

Ces « faux » attributs ont pour seules valeurs des listes au format XML. Celles-ci fournissent les noms et les caractéristiques de « vrais » attributs, mais pas leur valeur. La valeur peut être obtenue en recherchant le nom de ces éléments dans le dictionnaire des attributs.

Voici un exemple de valeur pour le « faux » attribut *Udas* :

```

<UDAInfo>
  <Attribute Name="Object.Script.AffectInputSource"
    DataType="MxBoolean" Category="MxCategoriesWriteable_USC_Lockable"
    Security="MxSecurityOperate" IsArray="false" HasBuffer="false"
    ArrayElementCount="0" InheritedFromTagName=" "/>
  <Attribute Name="PV_FB"
    DataType="MxFLOAT" Category="MxCategoriesWriteable_USC_Lockable"
    Security="MxSecurityOperate" IsArray="false" HasBuffer="false"
    ArrayElementCount="0" InheritedFromTagName=" "/>
</UDAInfo>
  
```

Ces listes XML fournissent de précieuses informations au sujet des attributs concernés. Elles permettent de connaître leur nom, qui est nécessaire pour les retrouver dans le dictionnaire d'attributs. Mais en plus, elles indiquent le type de valeur et sa dimension (pour les tableaux). Ces informations sont nécessaires pour pouvoir interpréter la valeur des attributs. En effet, ces valeurs sont implémentées sous forme d'objets génériques, sans indication de type.

¹ Dictionnaire de données : structure de données contenant une liste d'objets, identifiés par une clé unique.

L'utilisation de la clé (souvent une chaîne de caractères, mais pas obligatoirement) permet d'accéder directement à l'élément ciblé, sans devoir parcourir la liste.

Le principe est similaire à celui d'une table de hachage. Cette dernière ne permet toutefois que l'utilisation de chaînes de caractères en tant que clés.

Le tableau qui suit reprend la liste des types de données fréquents.

MxBoolean	Il s'agit d'une valeur booléenne, valant <i>true</i> ou <i>false</i> . Dans le cas des <i>field attributes</i> , ce type permet de savoir qu'il s'agit d'une valeur discrète (par opposition aux valeurs analogiques, de type <i>MxInteger</i> , <i>MxFloat</i> ou <i>MxDouble</i>).
MxInteger	Il s'agit d'un nombre entier.
MxFloat	Il s'agit d'un nombre décimal avec une précision limitée.
MxDouble	Il s'agit d'un nombre décimal avec une grande précision.
MxString	Ce type désigne une chaîne de caractères basique.
MxBigString	Ce type est optimisé pour la manipulation de chaînes de caractères de très longues tailles.
MxInternationalizedString	Ce type permet d'associer une chaîne de caractères à un code régional. En pratique, c'est le code <i>1033</i> (USA) qui est utilisé par ArchestrA. Une option de mon application permet de choisir le code à utiliser, au cas où il viendrait à changer.
MxElapsedTime	Ce type désigne une durée. Il s'écrit sous la forme <i>hh:mm:ss.aaaaaaaa</i> (où h = heures, m = minutes et s = secondes). Exemple : <i>02:31:22.1020000</i>
MxTime	Ce type désigne une date et une heure, avec une précision au dixième de microseconde.

Il existe d'autres types, tels que des structures de données personnalisées. Ces types de données sont rarissimes. Bien que je n'en aie rencontré aucun durant mon stage, j'ai tout de même fait en sorte que mon application puisse prendre en charge la majorité d'entre eux.

GRAccess agit en tant que contexte de données (*datacontext*). Cela signifie que les objets qu'il fournit restent liés à la base de données dont ils sont issus. La moindre modification apportée à un objet récupéré sera donc répercutée dans ArchestrA. Grâce à ce principe, la modification des éléments existants est grandement simplifiée (d'autres méthodes sont disponibles pour l'ajout et le retrait d'éléments). Néanmoins, ce système de *datacontext* peut aussi se révéler dangereux. Il convient donc de manier les objets de *GRAccess* avec prudence.

Établissement d'une connexion

Pour ouvrir un accès GРАccess, il faut créer un objet *GRAccessApp*. Celui-ci permet ensuite de récupérer la liste des *galaxies* et de se connecter à l'une d'entre elles.

```
// ouvrir GРАccess
try
{
    GРАccessApp accessCtx = new GРАccessAppClass();
}
catch (Exception dcExc)
{
    // erreur
}

// lister noms des galaxies
IGalaxies galaxies = accessCtx.QueryGalaxies("127.0.0.1");
if ((Object)Galaxies == null || GetContext.CommandResult.Successful == false)
{
    // erreur
}
HashSet<string> galaxiesNames = new HashSet<string>();
foreach (IGalaxy gal in galaxies)
    galaxiesNames.Add(gal.Name);

// se connecter à la galaxie TestGalaxy (si elle existe)
IGalaxy openGalaxy;
if (galaxies.Contains("TestGalaxy"))
{
    openGalaxy = galaxies["TestGalaxy"];
    openGalaxy.Login("nom_utilisateur", "mot_de_passe");
    if (openGalaxy.CommandResult.Successful == false)
    {
        // erreur
    }
}
```

Récupération de données

Il existe deux méthodes pour obtenir les objets ArchestrA dans GРАccess : selon une condition (dérivation, catégorie, zone/area, ...) ou selon une liste de noms.

```
// récupérer des éléments selon une condition
IgObjects instances =
openGalaxy.QueryObjects(EgObjectIsTemplateOrInstance.gObjectIsInstance,
                        EConditionType.derivedOrInstantiatedFrom "$AnalogDevice",
                        EMatch.MatchCondition);
if (openGalaxy.CommandResult.Successful == false) { /* erreur */ }

// récupérer des éléments par nom
string[] templatesNamesArray = new string[]{ "$AnalogDevice", "$UserDefined" };
IgObjects templates =
openGalaxy.QueryObjectsByName(EgObjectIsTemplateOrInstance.gObjectIsTemplate,
                             ref templatesNamesArray);
if (openGalaxy.CommandResult.Successful == false) { /* erreur */ }
```

L'absence de méthode pour l'obtention de tous les objets peut sembler handicapante. Il suffit toutefois d'utiliser une condition qui est toujours vraie pour obtenir tous les éléments.

```
// récupérer tous les templates de base (avec condition toujours vraie)
IgObjects templates =
openGalaxy.QueryObjects(EgObjectIsTemplateOrInstance.gObjectIsTemplate,
                        EConditionType.basedOn, "-N/A-",
                        EMatch.NotMatchCondition);
if (openGalaxy.CommandResult.Successful == false) { /* erreur */ }
```

Modification de données

Le système de *datacontext* fourni par GRAccess a pour conséquence que la moindre modification des données récupérées est prise en compte, ce qui met à jour la galaxie ArchestrA. Réaliser cette action de manière incontrôlée peut cependant s'avérer dangereux. Il est préférable d'avertir ArchestrA des modifications, en utilisant le système d'états et de check-out/check-in.

```
// récupérer l'instance existante
string[] instanceName = new string[] { "mon_instance" };
IgObjects existingInstance =
openGalaxy.QueryObjectsByName(EgObjectIsTemplateOrInstance.gObjectIsInstance,
                             ref instanceName);
if (openGalaxy.CommandResult.Successful)
{
    // mise à jour de l'instance
    foreach (IgObject inst in existingInstances)
    {
        // CHECK-OUT
        inst.CheckOut();
        try
        {
            // modifier les valeurs de l'instance ici
            // (renvoyer une exception si une erreur est repérée)
            // [...]

            // CHECK-IN (sauvegarde)
            inst.Save();
            inst.CheckIn();
        }
        catch (Exception exc) // erreur
        {
            // UNDO-CHECK-OUT (annuler)
            inst.UndoCheckOut();
        }
    }
}
```

Insertion de données

Lorsque de nouveaux éléments doivent être insérés, il faut récupérer le *template* parent dont le nouvel objet doit dériver. Il suffit ensuite d'utiliser une méthode de création d'objet, puis d'utiliser la même technique que pour la modification de données.

```
// récupérer template parent
string[] templateName = new string[] { "$AnalogDevice" };
IgObjects queryParent =
openGalaxy.QueryObjectsByName(EgObjectIsTemplateOrInstance.gObjectIsTemplate,
                               ref templateName);
if (openGalaxy.CommandResult.Successful == false) { /* erreur */ }

// extraire template parent du résultat
ITemplate templateParent = null;
if ((Object)queryParent != null)
    foreach (IgObject item in queryParent)
        templateParent = (ITemplate)item;
if ((Object)templateParent == null) { /* erreur */ }

// ajouter une nouvelle instance
IInstance newInstance = templateParent.CreateInstance("mon_instance", true);
if ((Object)newInstance == null) { /* erreur */ }

// CHECKOUT
newInstance.CheckOut();
try
{
    // éditer valeurs de l'instance
    // (renvoyer exception si erreur repérée)
    // [...]

    // CHECKIN (sauvegarde)
    newInstance.Save();
    newInstance.CheckIn();
}
catch (Exception exc) // erreur
{
    // UNDO-CHECKOUT (annuler)
    newInstance.UndoCheckOut();
    newInstance.DeleteInstance();
}
```

Test de performances

Durant ma découverte de *GRAccess*, j'ai pu constater qu'il était relativement lent. J'ai donc mis au point une série d'optimisations pour compenser cela : utilisation de SQL, mémorisation en mémoire centrale des données déjà récupérées, ...

Pour mettre en évidence la différence de performance apportée par mes optimisations, j'ai réalisé une série de tests. Ils comparent la durée nécessaire pour obtenir un même résultat en se servant de *GRAccess*, de requêtes SQL et d'une lecture du « cache » mémoire. La tâche accomplie consiste à récupérer toutes les informations de toutes les *instances* dérivées d'un *template*.

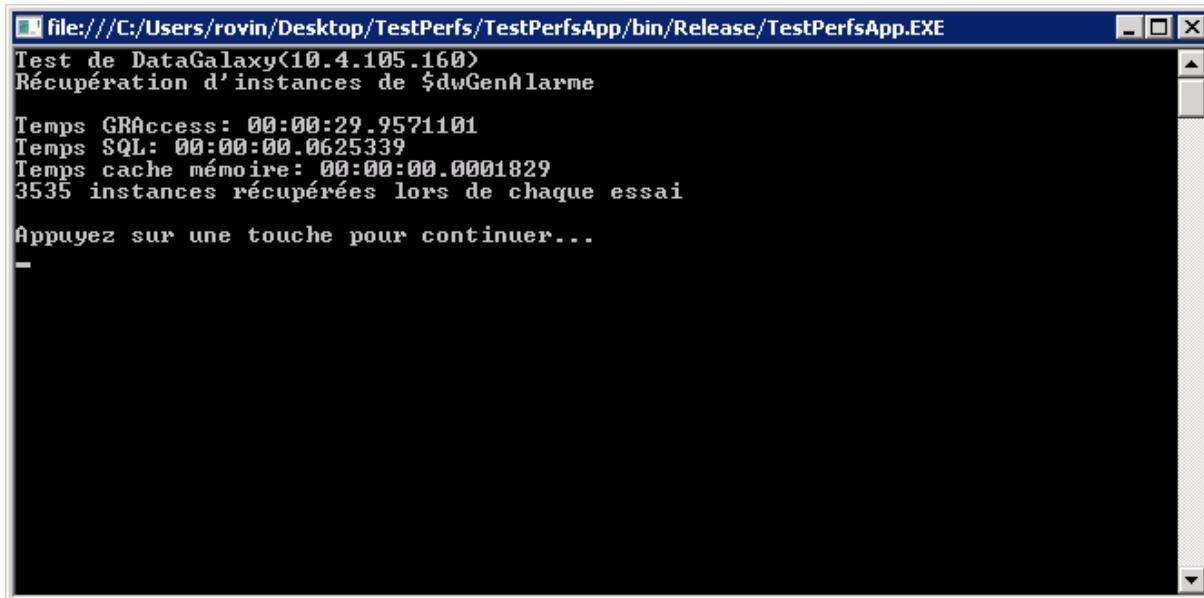
La méthode utilisée pour les mesures est l'utilisation de la classe *Stopwatch*. Celle-ci se sert de compteurs de temps à haute résolution (similaires à *PerformanceCounter* en C++).

```
Stopwatch stopwatch = new Stopwatch();
stopwatch.Reset();
stopwatch.Start();

// [...] // exécution de requête (dont la durée est mesurée)

stopwatch.Stop();
Console.WriteLine("Temps écoulé: {0}", stopwatch.Elapsed);
```

Les résultats obtenus sont affichés dans une fenêtre en mode console :



The screenshot shows a terminal window titled "file:///C:/Users/rovin/Desktop/TestPerfs/TestPerfsApp/bin/Release/TestPerfsApp.EXE". The window displays the following text:

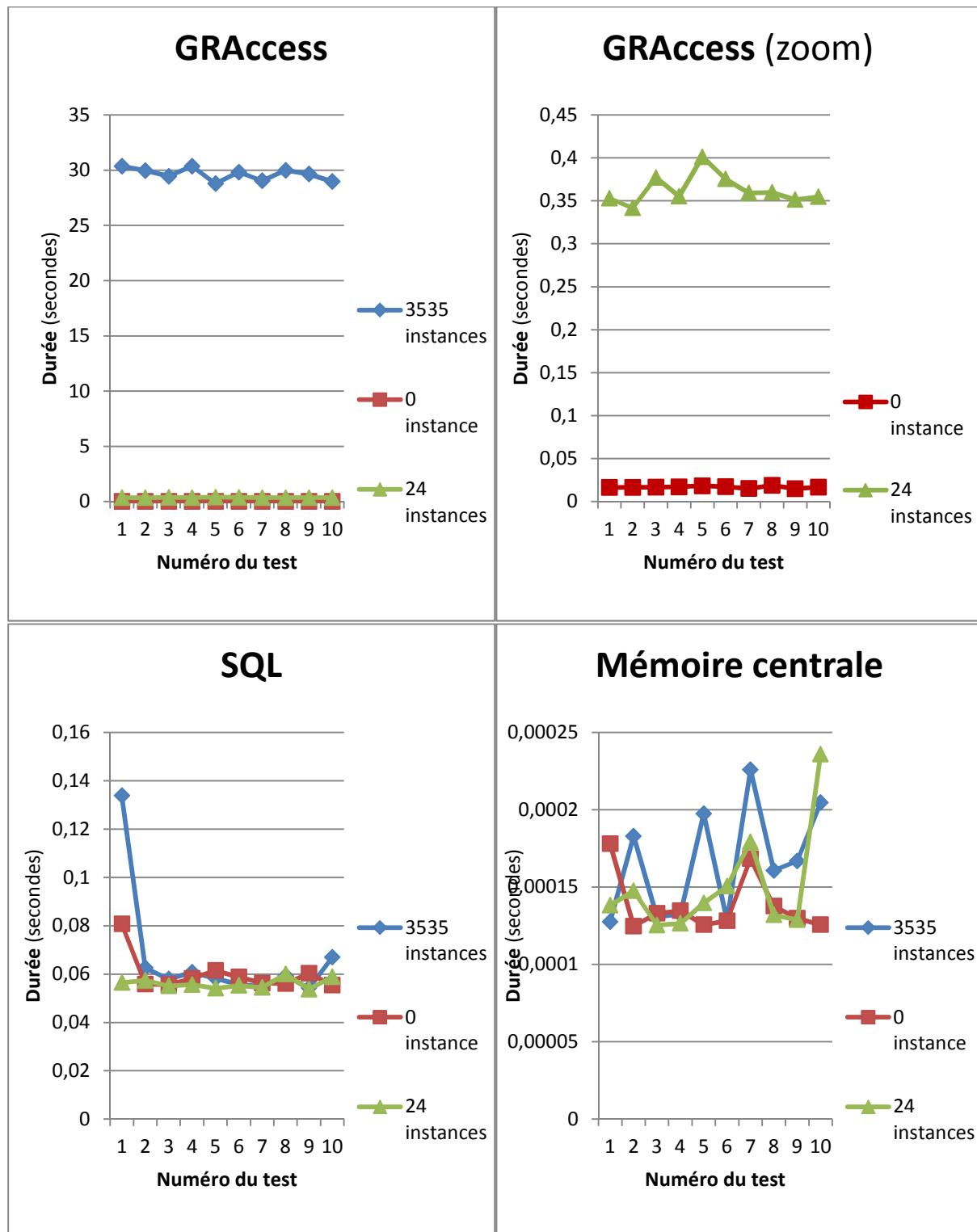
```
Test de DataGalaxy<10.4.105.160>
Récupération d'instances de $dwGenAlarme
Temps GRAccess: 00:00:29.9571101
Temps SQL: 00:00:00.0625339
Temps cache mémoire: 00:00:00.0001829
3535 instances récupérées lors de chaque essai
Appuyez sur une touche pour continuer...
```

Les résultats sont précis au dixième de microseconde près. Le nombre d'*instances* traitées est également indiqué.

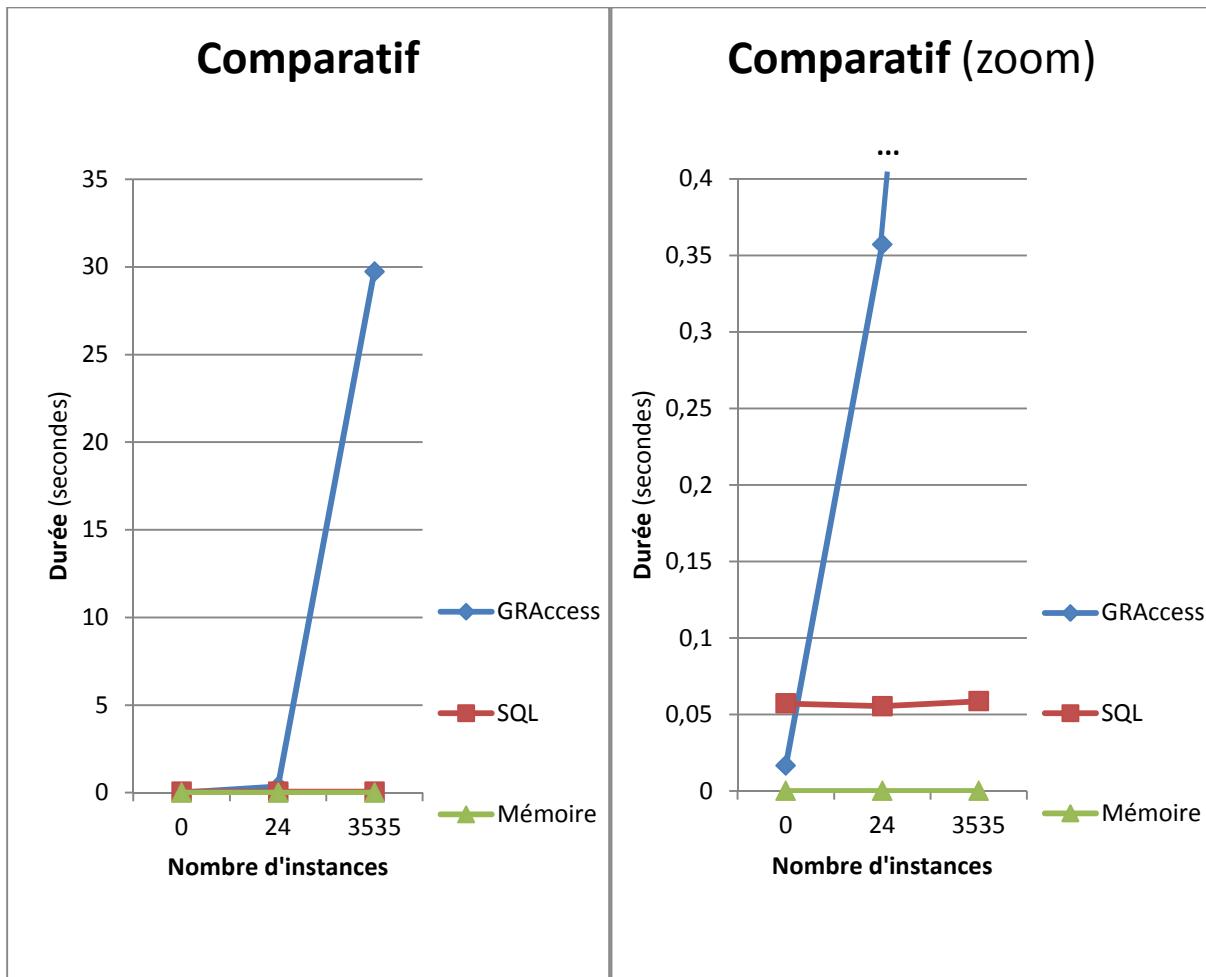
Trois séries de tests ont été réalisées :

- un *template* avec un très grand nombre d'*instances* (3535) ;
- un *template* ne comprenant aucune *instance* ;
- un *template* avec un petit nombre d'*instances* (24).

Pour minimiser les influences extérieures et les valeurs extrêmes, chaque test a été réalisé dix fois. Les graphiques ci-dessous illustrent les résultats obtenus.



En conservant les valeurs médianes pour chaque série, voici les résultats obtenus :



Il apparaît clairement que seul *GAccess* est influencé par le nombre d'*instances* récupérées. Il est également nettement plus lent pour les traiter que le *SQL* et la mémoire. La requête *SQL* semble toujours avoir à peu près la même durée d'exécution, qu'il y ait des *instances* à récupérer ou non. Il en va de même pour l'utilisation de la mémoire centrale.

Lorsqu'aucune *instance* n'est disponible, *GAccess* devient plus rapide que le *SQL*, dont la vitesse ne varie pas ou peu. Cependant, dès l'instant où des instances sont récupérées, le *SQL* est nettement plus rapide. Quel que soit le nombre d'*instances*, la mémoire centrale reste toujours plus performante que les autres méthodes.

Conclusions

GRAccess Toolkit est un outil fourni par Wonderware pour gérer des *galaxies* ArchestrA depuis des applications extérieures. Il assure la récupération de la plupart des éléments d'ArchestrA, ainsi que leur édition. Il est néanmoins incomplet : il ne dispose d'aucun moyen pour récupérer la liste des dossiers de classement de *templates* (utilisés au sein de l'arbre « Templates Toolbox » d'ArchestrA). Il manque aussi de flexibilité, n'offrant que deux méthodes d'obtention d'objets. Aucune méthode n'assure la récupération de tous les éléments sans condition (il est dès lors nécessaire de fournir une condition sans effet pour obtenir tous les objets). Aucune méthode ne permet la recherche d'éléments à partir de noms partiels ou de manière insensible à la casse. Il est également impossible de connaître le nombre d'éléments répondant à un critère sans les récupérer. Pour réaliser une application évoluée, *GRAccess* ne peut donc s'auto-suffire. L'utilisation de requêtes SQL indépendamment de *GRAccess* permet de combler ses lacunes (dossiers de classement, recherche, statistiques, ...).

Une installation de *GRAccess Toolkit* ne prend en charge qu'une seule version d'ArchestrA. Pour chaque version, une librairie différente est disponible. Je me suis donc assuré que le moins de code possible dépende de la librairie, afin de rendre le code de mon application en grande partie réutilisable pour la gestion d'autres versions.

Une autre caractéristique de *GRAccess* est son absence totale de gestion des erreurs. Aucune valeur de retour n'est émise et aucune exception n'est renvoyée en cas d'erreur. Il est donc très important de procéder, après chaque opération, à un contrôle systématique de sa réussite.

L'obtention des données via *GRAccess* est très lente. La lecture de certains attributs dans un objet récupéré provoque une nouvelle requête implicite, ce qui ralentit fortement le procédé. Par exemple, la lecture des erreurs et avertissements d'un *template* prend jusqu'à huit fois plus longtemps que celui nécessaire pour lire toutes ses autres valeurs. Il convient donc d'être attentif à éviter certains attributs, lorsque c'est possible.

L'ajout d'une série d'optimisations a engendré un gain de temps considérable dans mon application. L'utilisation du SQL s'est révélée beaucoup plus rapide que celle de *GRAccess*. Elle a donc permis d'accélérer certaines requêtes. Néanmoins, la complexité accrue de la base de données d'ArchestrA n'a pas permis de se passer de *GRAccess* pour tout. En dépit de sa lenteur, ce dernier reste le moyen le plus fiable de lire et surtout d'ajouter ou modifier des données. L'utilisation d'un « cache » de données s'est révélée être une optimisation très efficace : les objets d'ArchestrA ne sont lus que la première fois, puis sont conservés dans la mémoire centrale. Les accès ultérieurs sont ainsi presque instantanés.

4.3.3 - Tâches asynchrones en C# .NET

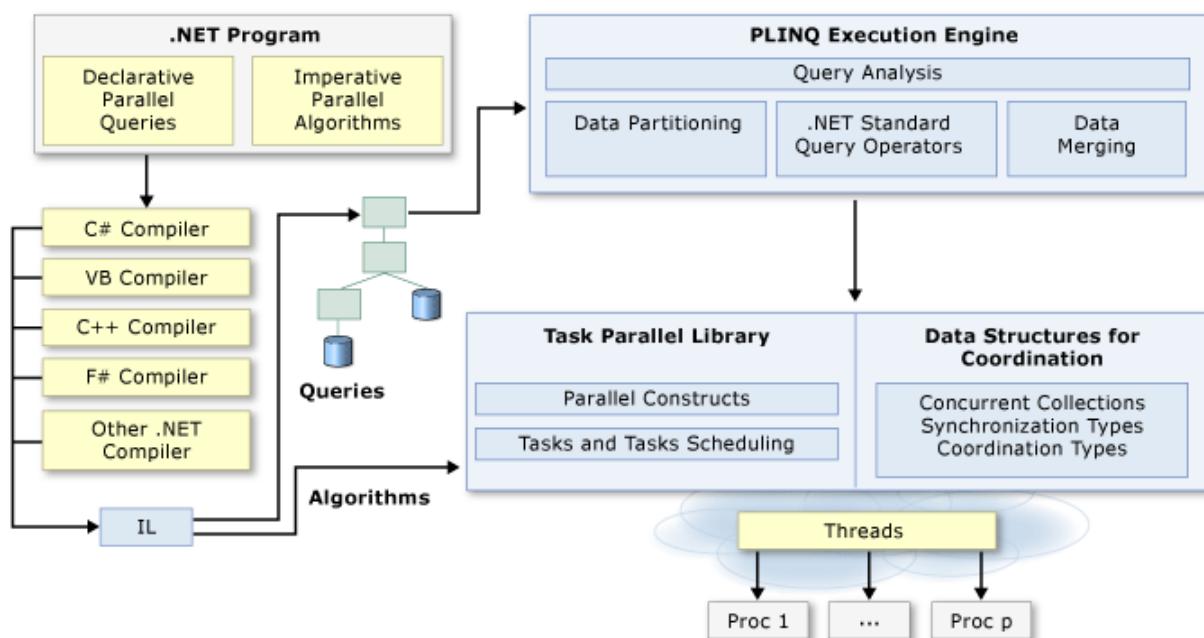
Introduction à la Task Parallel Library

De nos jours, la plupart des applications utilisent des threads. Ils permettent d'exécuter plusieurs actions en parallèle. Si aucun thread n'était utilisé, l'accomplissement de chaque tâche aurait l'exclusivité de l'utilisation du processeur. Par exemple, l'interface graphique de l'application resterait complètement figée, jusqu'à ce que l'action exécutée soit terminée.

Le langage C# propose plusieurs méthodes pour créer et gérer des threads. Chaque instance de la classe *Thread* représente un thread spécifique du système d'exploitation. La manipulation des instances influence directement l'exécution, sans aucune forme d'abstraction. Cette méthode, similaire à celle du C++, n'est avantageuse que pour certains domaines particuliers. Il est généralement préférable de l'éviter.

Une seconde approche consiste à utiliser la classe *ThreadPool*. Cette dernière prend en charge l'instanciation et l'exécution des différents threads de manière automatique. Elle n'offre guère de contrôle sur la manière dont les opérations se déroulent. Cette méthode, existant également en Java, est avantageuse au niveau abstraction. Il n'est plus nécessaire de gérer des mécanismes complexes. Elle présente toutefois une faiblesse importante : il est impossible de vérifier si un thread a terminé le travail qui lui a été confié. Il convient dès lors de prendre en charge cet aspect soi-même, en créant les éléments nécessaires pour indiquer qu'une tâche est terminée.

Enfin, le framework .NET propose une troisième approche, qui lui est propre : l'utilisation de la classe *Task*, issue de la *Task Parallel Library*. Elle offre le meilleur des deux mondes : une gestion automatisée des threads et la possibilité de savoir qu'une action est terminée. Comme la classe *ThreadPool*, elle s'occupe de la création et de l'exécution des threads. Cependant, elle permet d'être informé qu'une tâche est terminée et peut même renvoyer un résultat. Il est aussi possible d'y joindre une méthode dite de *callback*, appelée automatiquement après la fin de la tâche.



Source : <https://msdn.microsoft.com/en-us/library/dd460693%28v=vs.110%29.aspx>

Principes d'utilisation des tâches

Pour démarrer une tâche, il suffit d'appeler *Task.Run* et de lui fournir des instructions à exécuter. Ces instructions peuvent aussi être inscrites dans un objet de type *Action*, qu'il convient alors de passer en paramètre à la tâche. Pour réaliser une attente synchrone de la fin de la tâche, la méthode *Wait* peut être utilisée.

```
Task tache = Task.Run(() => // lancement d'une tâche asynchrone
{
    Console.WriteLine("tâche async");
});
tache.Wait(); // attente synchrone de la fin de la tâche
```

Il est assez rare d'utiliser l'instruction *Wait* à d'autres moments que pour la fin de l'application. Les tâches asynchrones sont en effet créées pour être non bloquantes. De plus, lorsqu'une attente est nécessaire pour réaliser une instruction, elle est généralement effectuée implicitement, rendant la méthode *Wait* inutile.

Pour définir une valeur de retour, il faut utiliser *Task<T>*, où *T* représente le type de la donnée renvoyée. Si une partie du code tente de lire la valeur de retour, elle réalisera un *Wait* implicite et attendra la fin de la tâche pour lire la valeur.

```
Task<int> tache = Task.Run(() =>
{
    return 123;
});
Console.WriteLine(tache.Result); // attend implicitement et affiche la valeur 123
```

Il est souvent utile qu'une tâche soit non bloquante durant son exécution, tout en ayant l'opportunité d'être informé lorsqu'elle se termine. La méthode *ContinueWith* sert à définir une méthode de *callback*, appelée à la fin de l'exécution de la tâche.

```
Task<int> tache = Task.Run(() =>
{
    return 123;
}).ContinueWith((t) =>
{
    Console.WriteLine(t.Result); // affichera la valeur 123
});
```

Le code du *callback* peut bien entendu contenir un appel vers une fonction prévue à cet effet. Il est également possible de créer une tâche sans la lancer directement. Elle peut alors être utilisée comme paramètre d'une fonction, par exemple. La classe *Task* offre une grande flexibilité.

```
Task tache = new Task(() =>
{
    Console.WriteLine("tâche async");
}).ContinueWith((t) =>
{
    OnTaskComplete(t); // appel d'une fonction en callback
});
// [...]
tache.Start(); // lancement
```

Conclusions

Parmi les trois méthodes de gestion des threads proposées par le langage C#, la classe *Task* est la plus avantageuse. Elle offre une abstraction complète des threads, tout en permettant d'être informé dès qu'une tâche est terminée. Il est possible d'y joindre la référence d'une méthode, qui sera appelée automatiquement à la fin de l'action demandée.

L'utilisation de ce système permet avant tout de réaliser des actions de longue durée, sans paralyser l'interface graphique de l'application. La combinaison des tâches asynchrones avec l'affichage d'une boîte de progression a fourni la possibilité que l'utilisateur soit informé de l'état d'avancement, tant en gardant le contrôle d'une interface graphique non figée.

La classe *Task* m'a aussi permis de mettre en place un système de file d'attente. Ainsi, lorsque plusieurs actions (ne pouvant pas être exécutées au même moment) sont demandées successivement, elles sont mises en attente l'une à la suite de l'autre. La boîte de progression informe l'utilisateur du nombre de tâches en attente. Elle lui fournit également la possibilité d'interrompre la tâche en cours. Lorsqu'une demande d'annulation est faite, la tâche en est informée, lui permettant de choisir de s'interrompre dès que possible.

4.3.4 - EPPlus – librairie de création de documents Excel

Introduction à EPPlus



EPPlus est une librairie en C# .NET permettant de lire et écrire des fichiers au format Microsoft Excel 2007 et les versions ultérieures (*xlsx*). Elle est capable de manipuler facilement et efficacement leurs données et leurs styles. Cette librairie est très aboutie, tout en restant très simple à utiliser.

Source de l'image : <https://twitter.com/msexcel>

EPPlus supporte principalement les fonctionnalités suivantes :

- accès aux cellules individuelles et aux groupes de cellules, et édition de leur contenu ;
- mise en forme des cellules : couleur et épaisseur des contours, couleur de fond, polices de caractères et taille, alignements ;
- création et édition de graphiques et figures ;
- ajout d'images ;
- dessin de formes géométriques et modification de leur aspect ;
- ajout de commentaires ;
- protection de contenu et chiffrement ;
- tables de pivot ;
- utilisation de formules mathématiques ;
- vérification et validation des données ;
- scripts VBA (Visual Basic).

Fonctionnement de la librairie

L'utilisation de la librairie est relativement simple. Il n'est pas nécessaire de gérer les différentes imbrications de la structure interne du document, contrairement à la librairie *Open XML SDK*.

La création d'un document commence par l'instanciation d'un package, destiné à recevoir les différentes feuilles Excel. Il est ensuite possible d'y ajouter des feuilles Excel (*worksheets*).

```
FileInfo newFile = new FileInfo(outputDir.FullName + @"\fichier.xlsx");
ExcelPackage package = new ExcelPackage(newFile);
ExcelWorksheet worksheet = package.Workbook.Worksheets.Add("Ma feuille");
```

L'édition du contenu d'une cellule peut être effectuée de deux manières :

```
worksheet.Cells[1, 1].Value = "Texte";
worksheet.Cells["A1"].Value = "Texte";
```

Il est également possible d'altérer une rangée de cellules. Cette pratique se révèle particulièrement utile pour l'édition du style des cellules.

```
worksheet.Cells[3, 2, 3, 5].Style.Numberformat.Format = "#,##0";
worksheet.Cells["C2:C5"].Style.Numberformat.Format = "#,##0";
```

Pour optimiser la modification d'un ensemble de propriétés de mise en forme, il est possible de récupérer la référence d'un groupe de cellules dans une variable. Cette dernière peut ensuite subir toutes les modifications nécessaires.

```
using (var range = worksheet.Cells[1, 1, 1, 5])
{
    range.Style.Font.Bold = true;
    range.Style.Fill.PatternType = ExcelFillStyle.Solid;
    range.Style.Fill.BackgroundColor.SetColor(Color.DarkBlue);
    range.Style.Font.Color.SetColor(Color.White);
}
```

Sitôt les modifications terminées, le document doit être sauvegardé :

```
package.Save();
```

Conclusions

La librairie la plus fréquemment utilisée pour manipuler des documents Excel est celle de Microsoft Office : *Office.Interop*. Malheureusement, son utilisation nécessite l'installation de la suite Microsoft Office. Elle n'était donc pas exploitable pour un projet devant s'exécuter sur un serveur (où Office n'est pas présent).

La librairie *EPPlus* répondait parfaitement aux besoins de mon projet. Elle est plus évoluée que la plupart des autres librairies remplissant le même rôle. Elle offre une grande flexibilité, tout en conservant une certaine simplicité. Elle s'est aussi montrée très performante.

4.3.5 - DocX – librairie de création de documents Word

Introduction à DocX



DocX est une librairie en C# .NET permettant de lire et écrire des fichiers au format Microsoft Word 2007 et les versions ultérieures (*docx*). Elle est capable de gérer les données et les styles de manière intuitive. La librairie est assez légère et un peu limitée.

Source de l'image : <https://docx.codeplex.com>

DocX supporte principalement les fonctionnalités suivantes :

- gestion des sections et paragraphes d'un document ;
- insertion, retrait et remplacement de texte dans un paragraphe ;
- mise en forme du texte : police de caractères, taille, couleur, gras, soulignement, ... ;
- propriétés des paragraphes : indentation, bordure, alignement, espacement, ... ;
- direction du texte : de gauche à droite ou de droite à gauche ;
- ajout d'images ;
- création et édition de tableaux ;
- modification d'en-tête et de pied de page ;
- ajout d'hyperliens.

La librairie permet également d'utiliser le système de styles prédéfinis d'un document existant. Elle ne permet toutefois pas d'en ajouter d'autres.

Fonctionnement de la librairie

L'utilisation de la librairie est très simple et intuitive. La gestion de l'agencement des données se limite à sa plus simple expression, procurant un gain de temps non négligeable.

La création d'un document ne demande qu'une seule ligne de code :

```
DocX document = DocX.Create("Fichier.docx");
```

L'ouverture d'un document existant est très similaire :

```
using (DocX document = DocX.Load("Fichier.docx"))
```

L'ajout d'un paragraphe de texte nécessite d'abord la création du paragraphe dans le document. L'élément créé est retourné par la fonction et peut être utilisé pour manipuler son contenu.

```
Paragraph parag = document.InsertParagraph();
parag.Append("Mon texte").Font(new FontFamily("Helvetica"));
```

Il est également possible d'accéder à un élément existant :

```
Paragraph parag = document.Paragraphs.ElementAt(2);
```

Lorsque les modifications sont terminées, le document doit être sauvegardé :

```
document.Save();
```

Comparaison avec la librairie Open XML SDK

La librairie standard utilisée par Microsoft pour créer des documents Excel est *Open XML SDK*. Cette librairie, si elle ne connaît aucune limite en termes de fonctionnalités, se révèle toutefois très compliquée et très hasardeuse à utiliser. En effet, si les balises générées ne respectent pas une imbrication parfaitement compatible avec celle utilisée par Microsoft, le fichier sera illisible. De plus, le moindre ajout de données demande une grande quantité d'opérations.

Les exemples qui suivent se chargent de créer un document, d'y ajouter un paragraphe « Hello World » et de sauvegarder le document. Le premier exemple utilise *Open XML SDK* et se révèle assez compliqué. Le second exemple, beaucoup plus simple, utilise la librairie *DocX*. Le fichier généré est identique dans les deux cas.

Création du fichier avec Open XML SDK :

```
static void Main(string[] args)
{
    // création du package
    WordprocessingDocument doc =
    WordprocessingDocument.Create
    (
        @"Test.docx",
        WordprocessingDocumentType.Document
    );

    // ajout d'une section principale au package
    MainDocumentPart mainPart = doc.AddMainDocumentPart();
    mainPart.Document = new Document();

    // creation du contenu
    Text textFirstLine = new Text("Hello World");
    Run run = new Run(textFirstLine);
    Paragraph para = new Paragraph(run);
    RunProperties runProp = new RunProperties();
    RunFonts runFont = new RunFonts();
    runFont.Ascii = "Arial Black";
    runProp.Append(runFont);
    run.PrependChild<RunProperties>(runProp);

    // ajout du contenu au document
    Body body = new Body(para);
    mainPart.Document.Append(body);

    mainPart.Document.Save();
    doc.Close();
}
```

Source : <http://cathalscorner.blogspot.be/2010/06/cathal-why-did-you-create-docx.html>, 22/05/2016

Création du même fichier avec DocX :

```
static void Main(string[] args)
{
    // creation du document
    using (DocX document = DocX.Create("Test.docx"))
    {
        // creation du contenu
        Paragraph p = document.InsertParagraph();
        p.Append("Hello World").Font(new FontFamily("Arial Black"));

        document.Save();
    }
}
```

Source : <http://cathalscorner.blogspot.be/2010/06/cathal-why-did-you-create-docx.html>, 22/05/2016

Conclusions

La librairie la plus fréquemment utilisée pour manipuler des documents Word est celle de Microsoft Office : *Office.Interop*. Malheureusement, son utilisation nécessite l'installation de la suite Microsoft Office. Elle n'était donc pas exploitable pour un projet devant s'exécuter sur un serveur (où Office n'est pas présent).

Parmi les autres possibilités, la librairie *DocX* répondait aux besoins du projet, même si elle trouve ses limites au niveau de la gestion des styles automatiques. Son utilisation est beaucoup plus simple et beaucoup plus fiable que celle de la librairie *Open XML SDK*. Elle offre donc un précieux gain de temps et d'efficacité. Je n'ai trouvé aucune alternative permettant de combler les lacunes de *DocX*, tout en gardant sa simplicité.

4.3.6 - Subversion – utilitaire de contrôle de versions

Introduction à Subversion (SVN)



Subversion, communément abrégé *SVN*, est un logiciel de gestion de versions. Il permet de stocker un ensemble de fichiers, tout en conservant la chronologie de toutes les modifications effectuées. Il est fréquemment utilisé pour gérer des projets dans les entreprises de développement informatique.

Source de l'image : https://fr.wikipedia.org/wiki/Apache_Subversion

Subversion repose sur une architecture client-serveur. Un serveur informatique centralisé héberge la référence des fichiers gérés, appelée « dépôt » (*repository*). Un ou plusieurs poste(s) client(s) manipulent une copie des fichiers de référence. Lorsqu'ils effectuent une synchronisation, les changements opérés sont détectés et « publiés » dans les fichiers de référence.

Les *commits* constituent une méthode de publication d'une série de modifications, réalisée de manière atomique : l'ensemble des changements sont publiés au même instant et de manière indissociable. Lors de la publication d'un *commit*, l'utilisateur est invité à fournir une dénomination résumant les changements opérés, ainsi qu'une description optionnelle. Ces informations peuvent ensuite être consultées dans un historique de modifications du projet.

Subversion est un logiciel évolué. Il est capable de réaliser toutes sortes de commandes. Par exemple, il permet de renommer ou déplacer un fichier, sans perdre la trace de son historique à l'emplacement précédent. Il dispose également de procédures de gestion des conflits, en cas de modifications contradictoires apportées par plusieurs clients.

Si l'utilisation de *Subversion* côté client peut être réalisée en ligne de commande, les utilisateurs préfèrent généralement utiliser une application conviviale pour s'en occuper. L'application utilisée chez Technord se nomme *TortoiseSVN*.

L'utilitaire TortoiseSVN



TortoiseSVN est un utilitaire simplifiant la prise en charge de la partie client de *Subversion*. Son utilisation est simple : il est implémenté en tant qu'extension de l'explorateur Windows. Il ajoute de nouveaux menus contextuels, permettant une gestion intuitive des fichiers.

Les dossiers gérés par *TortoiseSVN* arborent une icône particulière, qui permet de les distinguer. Cette icône indique l'état des dossiers et de leur contenu. L'utilisateur peut savoir du premier coup d'œil si un dossier contient des fichiers qui n'ont pas été publiés via un *commit*.

Source de l'image :

http://www.phusewiki.org/wiki/index.php?title=User_Guide_for_Standard_Script_Repository_GC

Conclusions

Subversion est un logiciel de gestion de versions très utilisé en entreprise, comme c'est le cas chez Technord. Son utilisation, côté client, est simple. Elle se limite à publier des *commits* lorsqu'un ensemble de fichiers a subi une modification.

La gestion de versions représente une partie importante de la gestion d'un projet. Il est important de publier les changements liés au même moment, faute de quoi un autre utilisateur pourrait récupérer des éléments incohérents.

L'utilisation de *Subversion* ne m'a posé aucun problème particulier. J'étais déjà habitué à un autre logiciel similaire, nommé *Git*. Les principes des deux outils sont fortement similaires.

4.4 - Organisation du projet

4.4.1 - Architecture

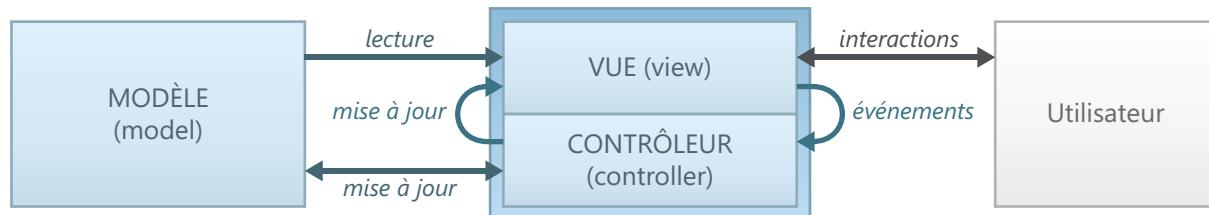
Architecture M-VC

Le patron architectural utilisé au sein de mon projet est l'architecture *modèle-vue-contrôleur*. C'est une architecture conçue pour les besoins des applications interactives et des applications web. Le paradigme consiste à regrouper les fonctions dans trois groupes distincts :

- Le **modèle de données** prend en charge les problématiques de gestion de données : récupération des données depuis une source externe, interactions avec une base de données, structure et traitement des données. Le modèle comprend également des méthodes de mise à jour des données. Il ne contient aucun lien direct vers le contrôleur ou la vue.
- La **vue** correspond à la partie visuelle de l'application. Elle prend en charge les interactions avec l'utilisateur : affichage des données du modèle, mise en forme, récupération des actions de l'utilisateur. Elle ne s'occupe pas du traitement des interactions mais délègue ce rôle au contrôleur, en lui transmettant des événements.
- Le **contrôleur** reçoit les événements de la vue et s'occupe de réaliser les actions nécessaires. Il appelle les méthodes du modèle pour réaliser des mises à jour ou pour obtenir des données. Il prend également en charge la gestion des changements réalisés par la vue.

L'architecture *modèle-vue-contrôleur* traditionnelle sépare complètement chacune des trois parties. Ce type de séparation est particulièrement adapté aux applications web et aux systèmes distribués. Il existe toutefois une variante, notée M-VC, qui consiste à réunir le contrôleur et la vue dans des fichiers liés. Cette technique est très utilisée pour les applications interactives en Java et C# car elle simplifie grandement les échanges entre la vue et le contrôleur. De plus, la structure du code généré par les utilitaires d'interfaces graphiques Swing (en Java) et WinForms (en C#) tend à se baser sur le même principe : la création de chaque fenêtre (ou élément graphique) génère deux fichiers liés, le premier gérant l'affichage et les événements, pendant que l'autre s'occupe du contrôle des changements et du traitement des interactions.

Patron architectural M-VC, regroupant la vue et le contrôleur

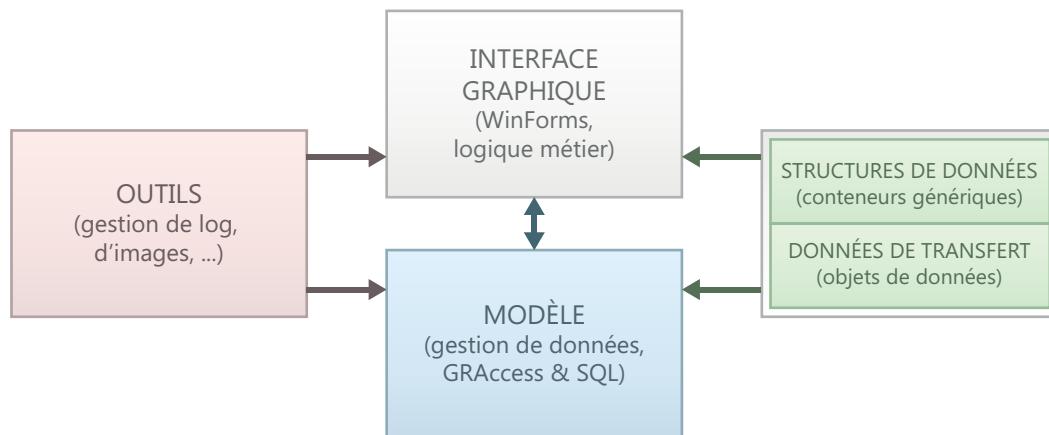


S'il m'avait été demandé d'utiliser un utilitaire d'interface graphique plus moderne, tel que WPF¹, il aurait été intéressant de se baser sur l'architecture MVVM (*modèle, vue, vue-modèle*), plus moderne également. Cependant, le cahier des charges imposait l'utilisation de WinForms.

¹ WPF (*Windows Presentation Foundation*) : utilitaire moderne de création d'interfaces graphiques, développé par Microsoft. Il permet une forte personnalisation de l'affichage et une meilleure séparation du code.

Découpe du projet

La découpe du projet découle directement des choix architecturaux. Le code de l'application a été regroupé en quatre parties distinctes, comme le montre le schéma qui suit.



Dans la mesure du possible, la logique métier (*business logic*) a été rassemblée au sein des fichiers du contrôleur. Celle-ci prend en charge la gestion des actions, sans se préoccuper du traitement des données ou des algorithmes complexes. Les fichiers du contrôleur sont liés aux fichiers de vue, qui gèrent l'affichage de l'interface graphique et la réception des événements d'interaction. Le contrôleur et la vue peuvent être considérés comme une seule et même partie, compte tenu de leur liaison étroite, décrite à la page précédente.

La partie modèle s'occupe de la récupération et de la gestion des données. Elle interagit avec les données d'Archestra via l'API GRAccess et via SQL Server. Elle réalise les traitements complexes, tels que la création des arbres de données (au niveau structurel), la déserialisation des listes d'attributs ou encore la recherche paramétrable d'éléments.

Les procédures utilisant GRAccess et celles se servant du SQL sont séparées dans deux sous-parties distinctes. Chacune de ces sections ne contient que le strict minimum, se limitant aux opérations dépendant de la technologie concernée. Les autres traitements sont mis en commun, en dehors de ces sous-parties. Si l'une des technologies utilisées devait être changée, par exemple pour gérer une autre version d'Archestra, la quantité de code réécrite serait minime.

Les méthodes d'interfaçage du modèle, destinées à une utilisation par le contrôleur, sont simplifiées autant que possible. Ainsi, ce dernier peut se concentrer sur la logique métier. Le contrôleur n'a, par exemple, aucun moyen de savoir si les opérations qu'il demande font appel à GRAccess ou à des requêtes SQL. Seul le modèle doit se préoccuper de cette distinction.

Outre la gestion des données et de l'affichage, d'autres parties composent l'application :

- La partie **outils** renferme un ensemble d'utilitaires généraux : journalisation (*log*), gestion de configuration, gestion d'images et de *sprites*¹, gestion de clés du registre. La plupart de ces éléments auraient pu avoir leur place au sein du modèle, puisqu'ils gèrent le traitement de données et d'entrées/sorties. Cependant, il m'a semblé judicieux de les séparer du modèle pour deux raisons.

Premièrement, ils gèrent des données complètement indépendantes, qui ne sont ni destinées à être affichées, ni destinées à être manipulées par le contrôleur et la vue. Par souci de clarté, il était donc préférable de ne pas les mélanger avec le modèle.

Deuxièmement, ces composants ne sont pas liés au projet. Cela signifie qu'ils peuvent être réutilisés pour d'autres projets. Nombreuses sont les applications qui manipulent le registre, les images et les fichiers de journalisation. Le fait que ces composants bénéficient de leurs propres fichiers bien séparés est donc un avantage pour leur réutilisabilité.

- La partie dédiée au **partage de données**, elle-même divisée en deux sous-parties, gérant respectivement les types de données (objets) et les conteneurs de données (structures d'objets). Ces définitions de données seront utilisées aussi bien par le modèle, que par le contrôleur et la vue.

Les classes désignant des **types de données** représentent des composants d'ArchestrA : *templates*, *instances*, *attributs*, ... Si ces éléments peuvent paraître faire double emploi, par rapport aux objets de GRAccess, ils sont néanmoins très utiles pour trois raisons :

- Comme mentionné lors de l'étude de GRAccess, les données récupérées par ce dernier reposent sur un principe de *datacontext* : chaque objet manipulé est directement lié à la base de données. En d'autres termes, la moindre modification apportée sur un élément récupéré serait répercutée dans ArchestrA. Ce comportement n'est pas souhaitable pour des objets qui seront manipulés, copiés et adaptés tout au long de l'exécution.
- Les objets obtenus via GGRAccess possèdent des types qui lui sont propres et qui dépendent de sa version. Si ces objets étaient utilisés partout, ils rendraient l'intégralité du code dépendante d'une seule version de GGRAccess, ne permettant ni le changement d'API, ni le changement de version. Par ailleurs, certaines actions requièrent l'utilisation de requêtes SQL, qui ne retournent pas ce type d'objets. Il serait donc impossible de mélanger des éléments récupérés via GGRAccess et via SQL. Le fait de copier les données dans des objets indépendants de GGRAccess présente donc un gros avantage de flexibilité et de généricité. Cela permet également de limiter les parties de code dépendant de l'API, comme mentionné à la page précédente.
- La nature des objets de GGRAccess est très complexe et n'est pas du tout adaptée à un affichage. Par exemple, les attributs, scripts et extensions sont tous mélangés et les listes permettant de les distinguer sont sérialisées en XML. L'utilisation d'objets indépendants de GGRAccess permet donc de simplifier leur utilisation, d'améliorer leur organisation et de faciliter leur affichage.

¹ Sprite : image bitmap regroupant plusieurs sous-images côte à côté, destinées à être séparées lors de leur utilisation.

Les classes de **conteneurs de données** permettent une organisation spécifique des données. Par exemple, le langage C# ne contient aucune classe permettant la manipulation d'arbres de données (au sens structurel du terme, pas au niveau de l'interface graphique). Créer une classe d'arbre était donc une nécessité. Cela m'a aussi permis d'apporter une série d'améliorations personnelles. À titre d'exemple, l'ajout d'un index contenant des références vers l'ensemble des noeuds de l'arbre a permis d'en décupler les performances. Implémenté sous la forme d'un *dictionnaire*¹ de données, l'index permet en effet l'accès direct à n'importe quel noeud, en fonction de son nom. Sans index, il était nécessaire de parcourir la hiérarchie complète de l'arbre pour trouver un élément.

Les conteneurs de données créés présentent l'avantage d'être totalement indépendants du reste de l'application. Cette caractéristique me permettra de les réutiliser dans d'autres projets, sans nécessiter la moindre adaptation du code. Ceci constitue une raison supplémentaire de les séparer du reste du code.

¹ Dictionnaire de données : structure de données contenant une liste d'objets, identifiés par une clé unique.

L'utilisation de la clé (souvent une chaîne de caractères, mais pas obligatoirement) permet d'accéder directement à l'élément ciblé, sans devoir parcourir la liste.

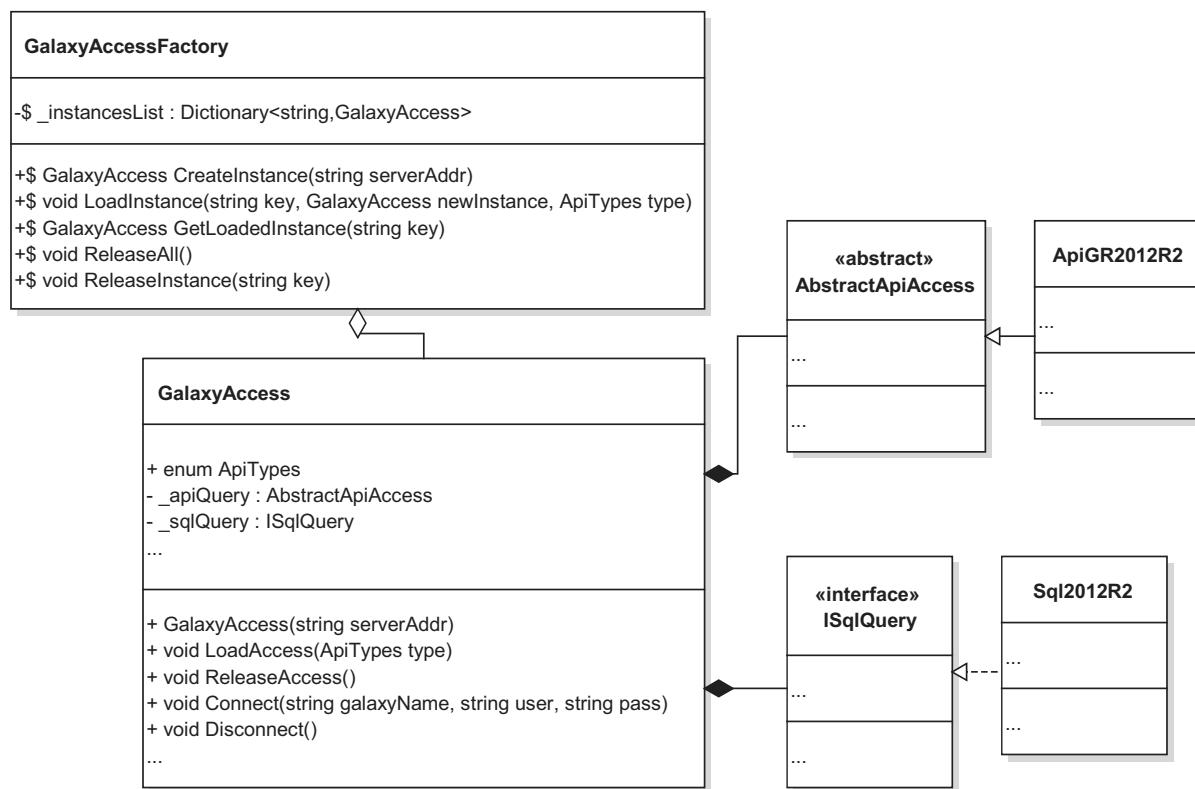
Le principe est similaire à celui d'une table de hachage. Cette dernière ne permet toutefois que l'utilisation de chaînes de caractères en tant que clés.

4.4.2 - Décisions architecturales

Patron de conception *factory* - gestion des versions

Le patron de conception créationnel *factory* (fabrique) permet l'instanciation d'objets dérivés d'un type abstrait. L'appelant se sert des méthodes de la classe abstraite pour utiliser l'objet et n'a pas connaissance de sa classe exacte. Le code qui utilise l'objet créé dépend donc exclusivement de la classe abstraite. Cette pratique permet de créer plusieurs variantes de classes dérivées sans nécessiter de changements au niveau du code qui les utilise.

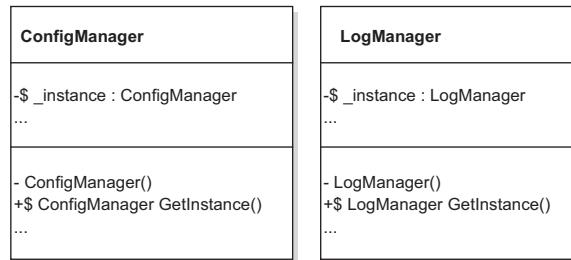
L'utilisation de la *factory* pour mon projet ne se contente pas de la création d'un seul objet : elle en crée deux. Le premier permet de réaliser une abstraction de la version de GRAccess utilisée. Le second réalise la même abstraction pour la partie dédiée au SQL. Le choix de version est effectué une seule fois au début, puis il n'est plus nécessaire de s'en soucier.



Le patron de conception *factory* permet, dans ce cas-ci, de gérer plusieurs versions différentes d'ArchestrA. Pour ajouter le support d'une nouvelle version, il suffit d'ajouter de nouvelles classes dérivées des classes abstraites et interfaces concernées. Une énumération est également prévue, pour permettre l'ajout d'une option dans la liste de sélection de version de l'interface graphique. Cette organisation assure ainsi l'évolutivité de l'application.

Patron de conception *singleton* - configuration et journalisation

Le patron de conception *singleton* a pour but d'assurer l'instanciation unique d'une classe. Il est utilisé lorsqu'un objet doit être global et unique dans une application. Par ailleurs, la nature du *singleton* le rend accessible partout.



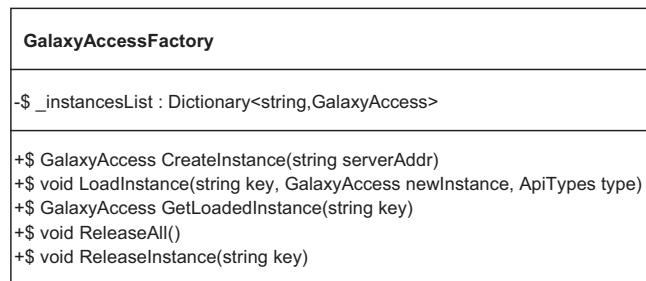
Le *singleton* est utilisé pour le conteneur de configuration et pour l'utilitaire de journalisation (*log*). Ces deux classes doivent pouvoir être appelées à n'importe quel endroit du code. Il est également préférable que ces éléments n'existent qu'en un seul exemplaire.

La configuration doit être la même partout, et la moindre modification doit se répercuter dans l'ensemble du programme. L'unicité permet ainsi le stockage de données globales. Elle assure également que le fichier de configuration ne soit lu qu'une seule fois (au démarrage) et que les données écrites (en cas de modification des paramètres) soient cohérentes.

Dans le cas de la journalisation, un accès à un fichier est établi. Une gestion de la concurrence est nécessaire. L'unicité de l'instance permet d'utiliser une variable d'exclusion mutuelle au sein de celle-ci. Toute tentative d'accès au fichier est ainsi contrôlée par cette variable.

Patron de conception *singleton* multiple - modèle de données

Si le patron de conception *singleton* s'applique généralement à une seule instance, son utilisation peut être étendue à un tableau d'instances, identifiées par une clé unique. Chaque accès avec la même clé récupérera la même instance. Ce système permet d'autoriser plusieurs instanciations, tout en gardant un contrôle minutieux sur leur nombre.



La raison pour laquelle j'ai mis en place un *singleton* multiple pour gérer l'instance d'accès à ArchestrA est de permettre la gestion de plusieurs galaxies en même temps. Avant la réception du cahier des charges, une idée avait été émise au sujet de comparaisons entre galaxies. L'idée n'a pas été conservée. Il m'a toutefois semblé préférable de conserver le mécanisme que j'avais mis en place car il offrait une opportunité d'évolutivité supplémentaire à moyen terme.

Création de contexte à l'extérieur de l'interface graphique

Le contexte d'une application prend en charge la gestion des événements de cette application, les interactions avec l'utilisateur (clavier, souris, ...), ainsi que la communication du système d'exploitation avec l'application. Il est indispensable de créer un contexte pour toute application fenêtrée, et ce, quel que soit le langage. Heureusement, cette mise en place se fait généralement très simplement.

Par défaut, la création d'un projet C# avec WinForms dans Visual Studio associe la création du contexte de l'application à la création de la première fenêtre. Si cette méthode est la plus simple à mettre en place, elle présente cependant certaines limites en termes de flexibilité, car la première fenêtre créée ne pourra pas être détruite avant la fermeture de l'application.

Souhaitant pouvoir détruire complètement la fenêtre d'identification sitôt la connexion établie, pour économiser l'empreinte mémoire de mon application, j'ai été contraint de modifier la gestion du contexte. Plutôt que de laisser la fenêtre de connexion le créer, j'ai réalisé sa mise en place manuellement, en dehors de toute fenêtre. Cette méthode présente certains avantages :

- L'évolution de l'application est facilitée. Il est possible de manipuler les formulaires souhaités dans l'ordre souhaité. Il n'y a pas une « fenêtre de départ » imposée. Le lancement des fenêtres est réalisé dans un fichier dédié à cette tâche, permettant de facilement changer leur ordre ou en ajouter d'autres.
- Le fait que l'ouverture de la première fenêtre ne soit pas la première étape accomplie au démarrage permet de réaliser d'autres tâches avant de l'ouvrir. Par exemple, il peut être utile de charger la configuration avant l'ouverture de la fenêtre, afin de pouvoir instancier celle-ci avec les paramètres adaptés.
- Le contexte existe encore après la fermeture de la fenêtre, jusqu'à la fin de l'application. Ainsi, même en cas de crash de l'interface graphique, il reste possible de fermer proprement les différentes ressources. De plus, il n'est plus nécessaire d'inclure la procédure de fermeture dans le code d'une fenêtre. Il est donc possible de changer l'ordre de fermeture des fenêtres sans devoir déplacer et modifier le code de fermeture.

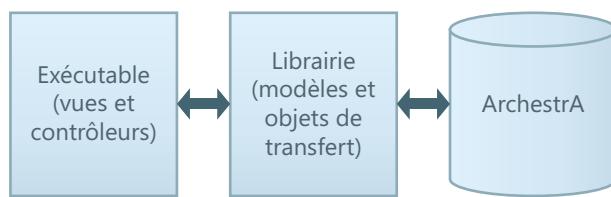
4.4.3 - Structure de la solution

Séparation des projets

Comme mentionné au point 4.4.1, mon projet repose sur une architecture M-VC. Par conséquent, les fichiers du modèle sont séparés de ceux du contrôleur et de la vue. D'autres parties contiennent un ensemble d'outils et des objets de transfert (types) et de structure de données.

Seule la partie dédiée à l'interface graphique et à la logique métier décide du comportement de l'application. Toutes les autres sections peuvent être considérées comme des outils, destinés à fournir au contrôleur les services dont il a besoin. Pour cette raison, j'ai décidé de séparer ma solution Visual Studio en deux projets distincts :

- une librairie de classes, comprenant le modèle, les outils et les objets de transfert ;
- une application exécutable, comprenant le contrôleur et la vue, et se servant de la librairie.



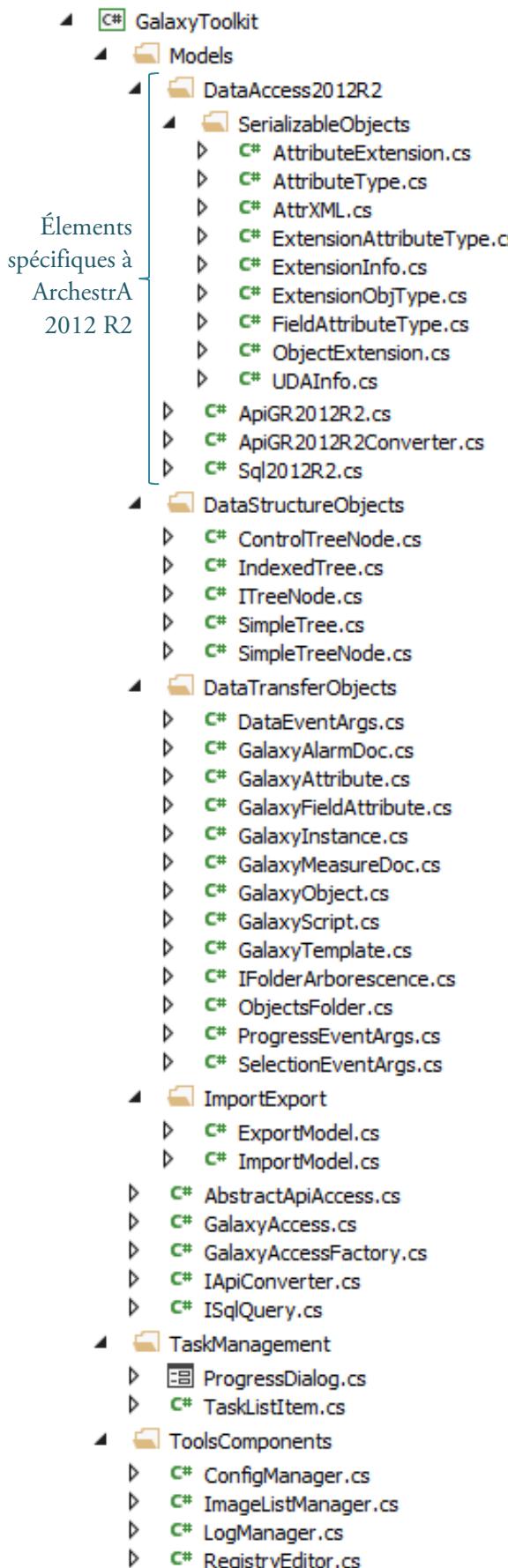
La séparation des projets présente un certain nombre d'avantages :

- Chacun des projets réalise une partie de la compilation. Il devient ainsi possible de n'effectuer qu'une compilation partielle de la solution. Si une modification doit être apportée à l'interface graphique, il n'est pas nécessaire de recompiler le modèle et les outils. L'inverse est vrai également. Cet état de fait procure un gain de temps considérable en phase de développement, où les recompilations nécessaires sont souvent très nombreuses.
- La séparation des projets permettrait une meilleure répartition du travail, si une équipe devait modifier le projet ultérieurement. Les employés chargés du *front-end*¹ travailleraient sur le premier projet, pendant que ceux chargés du *back-end*² s'occuperaient du second.
- La librairie de classes sera compilée sous la forme d'une librairie DLL, qui sera utilisée par le fichier exécutable de l'application. Cela signifie que la librairie pourrait être partagée par plusieurs applications. Ainsi, si Technord décidait de créer des applications supplémentaires (par exemple, pour l'édition ou pour la conversion de données), elles pourraient utiliser la même librairie. Les modifications apportées à celle-ci seraient mineures, représentant ainsi un gain de temps considérable.

¹ Front-end : développement des aspects graphiques et de la logique métier d'une application.

² Back-end : développement des traitements, algorithmes et échanges de données d'une application.

Librairie de classes - modèle de données et objets de transfert



Classes d'objets (dé)sérialisables :

Ces éléments sont utilisés pour réaliser la désérialisation de listes au format XML fournies par GРАccess (listes d'*UDAs* et d'extensions).

Classes d'obtention et mise à jour de données, spécifiques à une version d'une technologie :

Elles contiennent des méthodes de lecture et écriture de données utilisant GРАccess ou des requêtes SQL.

Conteneurs génériques.

Objets de transfert de données (types de données) :

Ces objets représentent des types de données d'ArchestrA : *templates, instances, attributs (UDAs et field attributes), scripts, alarms, ...*

Une classe permet aussi de décrire les dossiers de classement de l'un des arbres de *templates*. Enfin, plusieurs classes sont dédiées aux événements personnalisés.

Classes d'import et d'export :

Les méthodes de manipulation de fichiers de données (Word, Excel, ...) sont centralisées dans ces classes, ainsi que les traitements des données nécessaires.

Classes de traitement commun du modèle de données et interfaces d'abstraction des classes spécifiques à une version ou technologie.

Boîte de progression de la réalisation des tâches. Système de file d'attente et de gestion des tâches.

Gestion de configuration (données et fichier).
Gestion d'icônes de listes et de découpe de sprites.
Utilitaire de journalisation (log).
Utilitaire de lecture/écriture de clés du registre.

Application exécutable – logique métier et interfaces graphiques WinForms

C# GalaxyApp	
Views	
ConnectForm.cs	Fenêtre de départ – connexion à GRAccess et à une <i>galaxie</i> .
TreeControl.cs	Formulaire d'affichage des arbres de <i>templates</i> dans 2 onglets.
TreeViewMultiselect.cs	Formulaire d'affichage de liste multi-colonnes triable/filtrable.
ListColumnSorter.cs	Formulaire d'affichage d'attributs <i>UDAs</i> (onglet contextuel).
ListObjectsControl.cs	Formulaire d'export par catégorie/option (onglet contextuel).
UdaControl.cs	Formulaire d'export par liste de sélection (onglet contextuel).
ExportCategoryControl.cs	Lancement d'export commun aux deux formulaires précédents.
ExportSelectionControl.cs	Formulaire d'import de données (onglet contextuel).
ExportCommon.cs	
ImportControl.cs	
ExportClassifyForm.cs	Fenêtre d'options d'export de rapports de <i>templates</i> (classement).
ClassifyCategoriesControl.cs	Fenêtre d'options d'export de rapports de <i>mesures/alarmes</i> .
ClassifyTemplatesControl.cs	Fenêtre de recherche de <i>templates</i> et <i>instances</i> .
ExportTableAttrForm.cs	Choix de type de sélection des résultats de recherche dans l'arbre.
SearchForm.cs	Formulaire générique de choix (boutons radio générés).
TreeSelectionForm.cs	Fenêtre d'options de configuration de l'application.
ChoiceForm.cs	
OptionsForm.cs	Boîte de dialogue de présentation de l'application.
AboutBox.cs	
App.config	
app.manifest	
AppEntryPoint.cs	Point d'entrée (création de contexte) de l'application.
archestra.ico	
MainView.cs	Fenêtre principale de l'application.

4.4.4 - Diagramme des classes

L'approche orientée objet d'une analyse avec UML est centrée sur le diagramme des classes. Ce dernier décrit aussi bien un ensemble d'actions que des informations qui y sont liées dans une même entité. Chaque entité, nommée classe, contient donc un nom, des données (propriétés) et des actions (méthodes). Un parallèle peut directement être établi entre le diagramme des classes et la structure du code d'un projet en langage orienté objet.

Les diagrammes qui suivent n'illustrent que la hiérarchie des classes. Les diagrammes complets des classes principales sont détaillés en *Annexe 1*.

Diagramme des relations entre les classes – modèle et outils

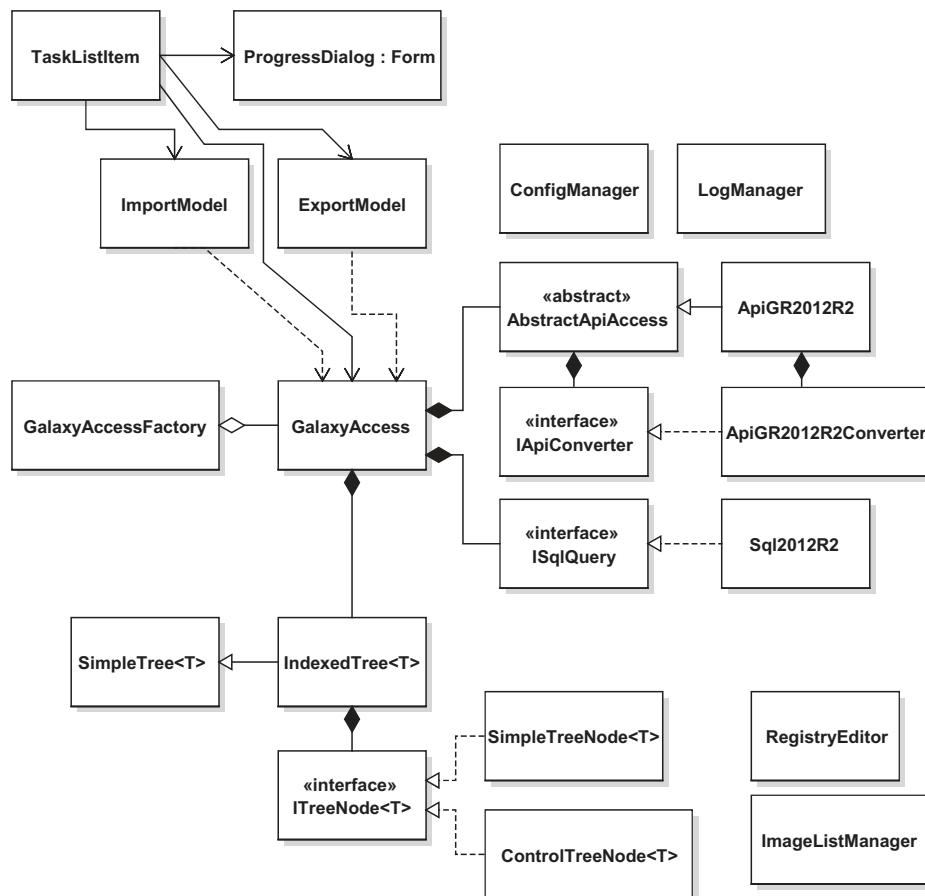


Diagramme des relations entre les classes – objets de transfert et structure

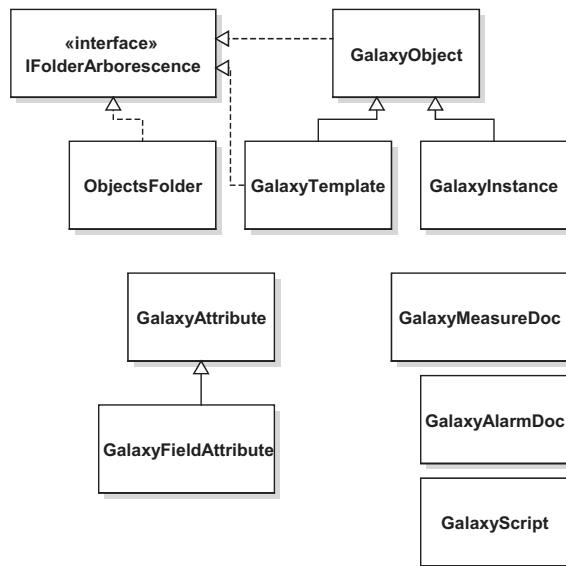
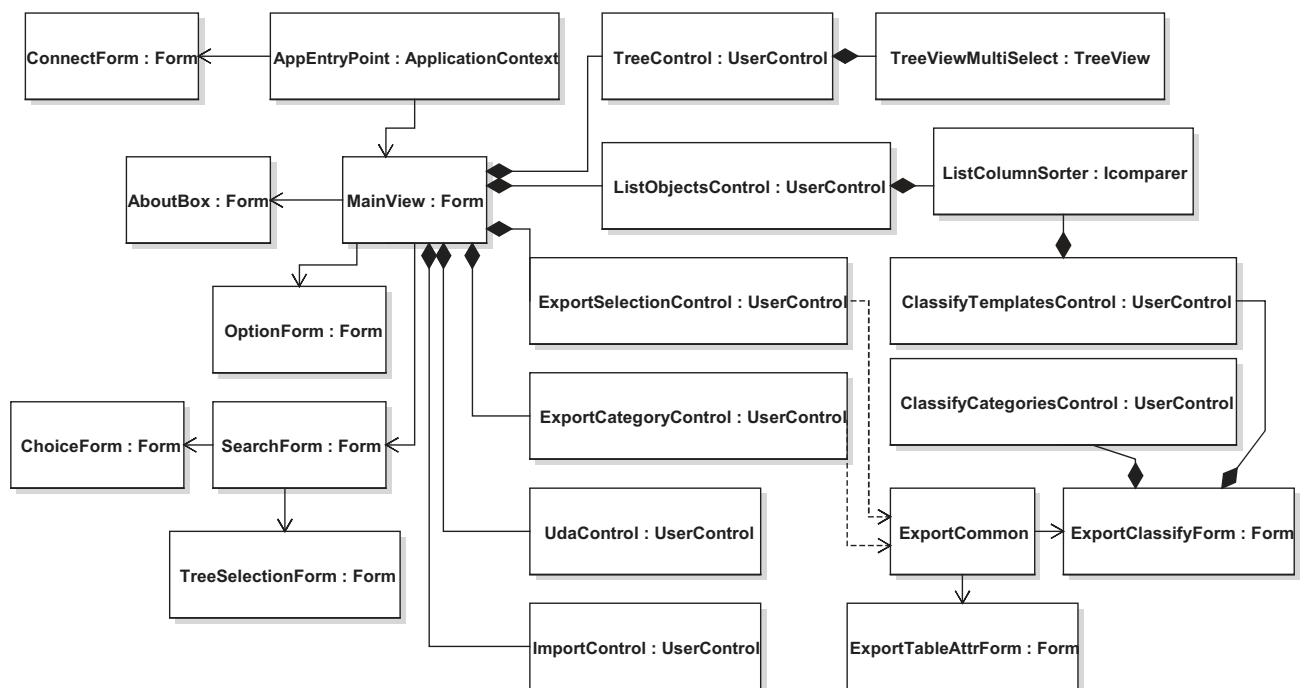


Diagramme des relations entre les classes – interface graphique



Chapitre 5

Développement

Ce chapitre détaille les fonctionnalités de mon projet et leur implémentation. Il se concentre essentiellement sur l'aspect fonctionnel de l'application : ce qu'elle permet de faire et la manière dont elle le fait. Les spécificités des technologies utilisées ont, quant à elles, déjà été abordées au chapitre précédent. Une description des efforts réalisés au niveau du design et de l'ergonomie est fournie après l'explication des fonctionnalités. Enfin, la fin du chapitre présente les problèmes rencontrés lors du développement.

Le contenu de ce chapitre est divisé en plusieurs parties :

- **Outils utilisés** : une liste brève de tous les outils utilisés pour réaliser mon projet ;
- **Fonctionnalités du projet** : la description de chaque fonctionnalité de l'application ;
- **Design et ergonomie** : les améliorations de l'expérience utilisateur et des tests d'ergonomie ;
- **Problèmes rencontrés** : la liste des problèmes rencontrés au cours du développement.

5.1 – Outils utilisés



Source de l'image : <http://www.microsoft-desktop.com/2010/01/windows-server-2008-r2-quelles-nouveautes>

Microsoft Windows Server 2008 R2

Système d'exploitation, utilisé pour essayer Archestra IDE et réaliser des tests de mon application.



Source de l'image : <http://vpourchet.com/vmware>

VMware Workstation 8

Gestionnaire de machines virtuelles, utilisé pour virtualiser un système Windows Server avec ArchestrA.



Source de l'image : <https://en.wikipedia.org/wiki/Wonderware>

Wonderware ArchestrA IDE 2012 R2

Outil d'édition de projets ArchestrA, utilisé pour découvrir les principes d'ArchestrA et pour ajouter des données de test (récupérées avec GRAccess).



Source de l'image : <http://jtower.com/blog/undocking-parts-of-team-explorer-in-visual-studio>

Microsoft Visual Studio 2013

Environnement de développement, utilisé pour écrire le code de mon application, la compiler et la déboguer.



Source de l'image : http://logos.wikia.com/wiki/Microsoft_SQL_Server

Microsoft SQL Server Management Studio 2008

Outil de gestion de bases de données, utilisé pour explorer la base de données d'ArchestrA et pour tester des requêtes.



Source de l'image : <https://www.techcress.com/how-to-encrypt-text-files-using-notepad>

Notepad++

Éditeur de texte évolué, utilisé pour ouvrir des fichiers de trace de débogage, des fichiers de données (*ini, json, xml*) et des fichiers de résultats (*csv*).



Microsoft Word 2013

Logiciel de traitement de texte, utilisé pour créer un canevas de document et pour vérifier le contenu des fichiers exportés par mon application (*docx*).

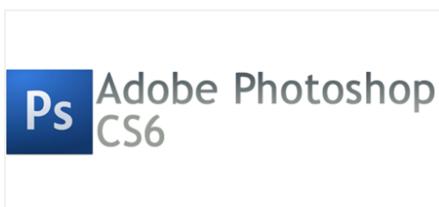
Source de l'image : <http://www.technewsable.com/office-2013-keyboard-shortcuts-for-microsoft-word>



Microsoft Excel 2013

Logiciel tableur, utilisé pour créer un canevas de document et pour vérifier le contenu des fichiers exportés par mon application (*xlsx, csv*).

Source de l'image : <http://www.ampercent.com/open-excel-sheet-startup-excel-20132016/18291>



Adobe Photoshop CS6

Outil de création d'images, utilisé pour créer les icônes et les images nécessaires pour mon application.

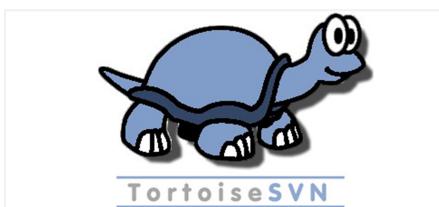
Source de l'image : <http://newallphoto.blogspot.be/2011/11/photoshop-logo.html>



Wunderlist

Outil de planification, utilisé pour organiser mon travail, fixer des échéances et garder une trace de toutes les tâches accomplies durant mon stage.

Source de l'image : <http://www.sparkvirtualassistance.com/what-is-wunderlist>



TortoiseSVN

Outil de gestion de versions, utilisé pour réaliser des sauvegardes de mon travail et contrôler ses changements.

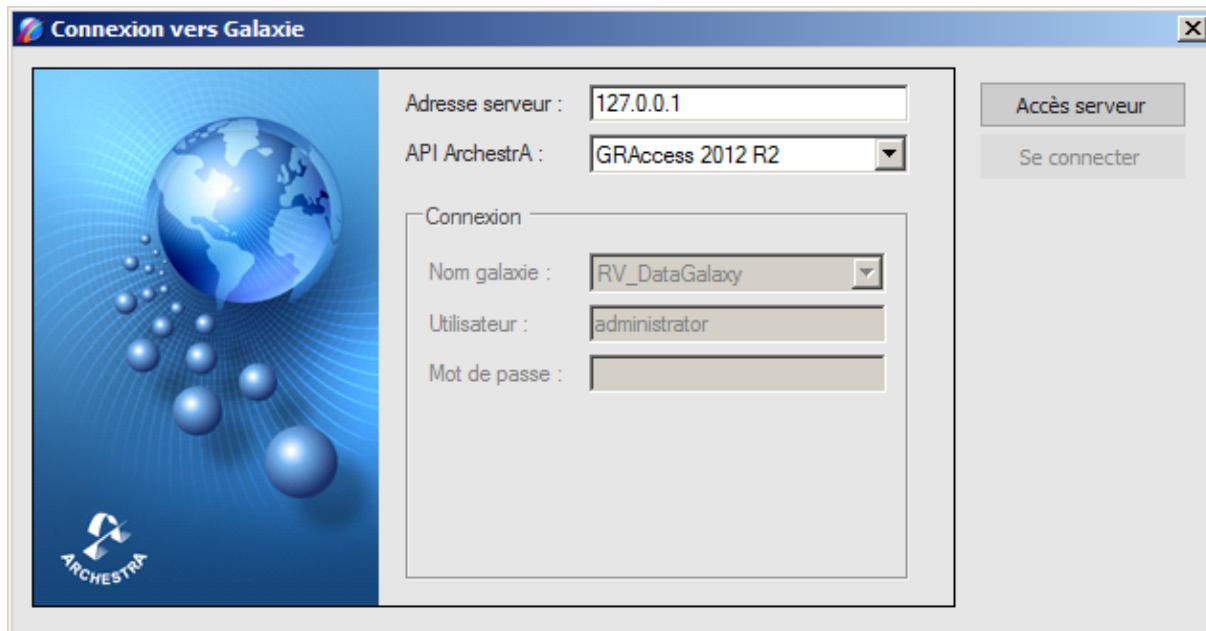
Source de l'image : <http://techtools24.blogspot.be/2014/08/tortoisessvn-188-revision-control.html>

5.2 – Fonctionnalités du projet

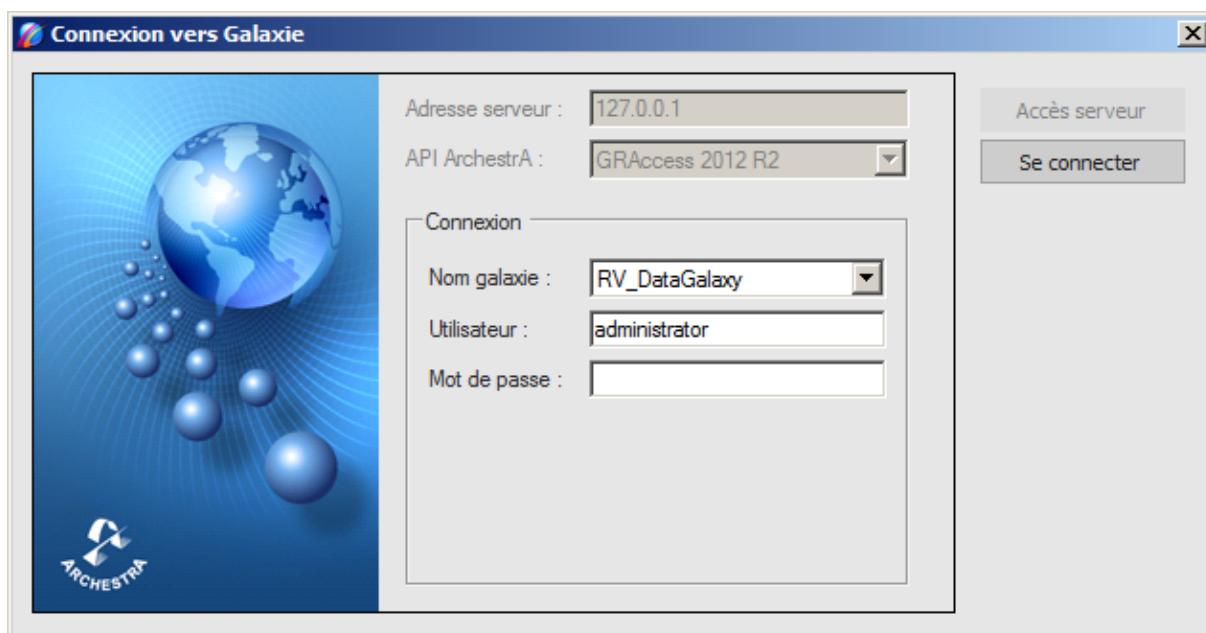
5.2.1 - Connexion

Le rôle de la fenêtre de connexion est d'obtenir un accès à ArchestrA via GRAccess, puis d'établir une connexion avec l'une des galaxies d'ArchestrA. L'apparence de la fenêtre est volontairement similaire à l'écran de connexion d'ArchestrA IDE.

Pour ce faire, l'utilisateur commence par saisir l'adresse IP du serveur ArchestrA auquel il souhaite accéder. Il précise aussi la version d'ArchestrA qui correspond à ce serveur (le projet ne supporte actuellement que la version 2012 R2).

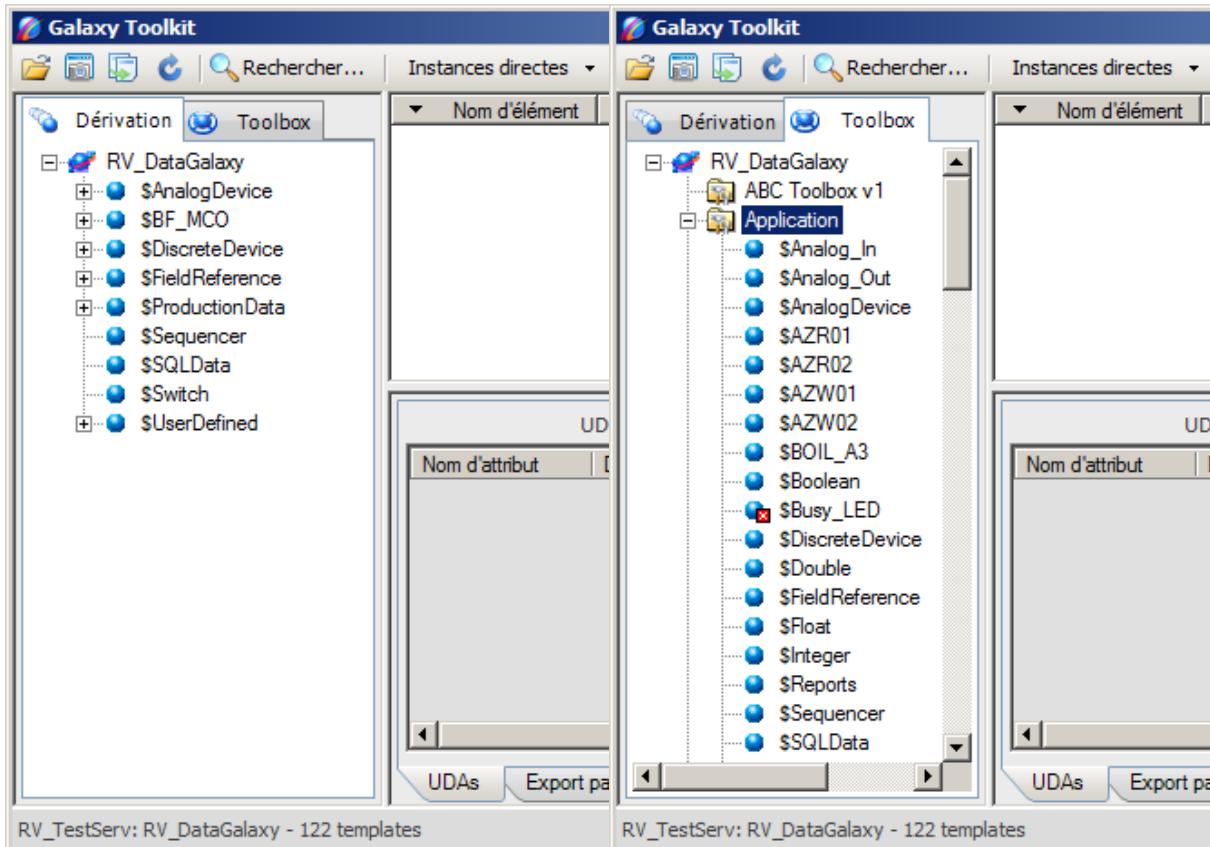


La deuxième étape consiste à choisir une galaxie et à fournir un nom d'utilisateur et un mot de passe pour s'y connecter.

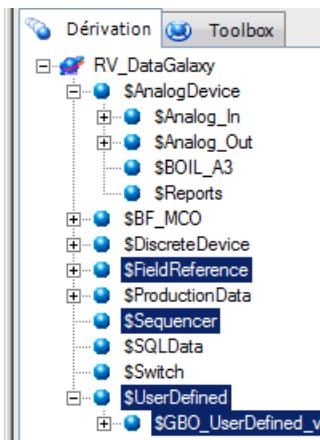


5.2.2 - Arbres de données

Après l'ouverture de la fenêtre principale, la première tâche effectuée est la construction des deux arbres de templates. Cette étape implique d'abord la récupération de l'ensemble des templates via l'API GRAccess, puis l'ensemble des dossiers de classement en SQL. Les arbres de données peuvent ensuite être construits, puis affichés. GRAccess ne permet pas d'obtenir des dossiers de classement, d'où l'utilisation du SQL pour les récupérer.



Sitôt les arbres affichés, il est possible de passer de l'un à l'autre via deux onglets. L'état des arbres (déploiement des nœuds et sélection des éléments) peut être sauvegardé ou chargé sous forme de fichiers *snapshots*, mais cette fonctionnalité sera expliquée au point 5.2.4.



Chaque arbre de templates permet la sélection d'un ou plusieurs éléments. Le support de la sélection multiple n'est pas présent dans les classes fournies par le langage C#. Pour cette raison, une librairie¹ a été utilisée, afin d'y parvenir.

Malheureusement, cette librairie ne permettait pas la sélection par programmation. J'ai donc été obligé de la modifier pour développer les fonctionnalités de chargement de *snapshots* et de synchronisation, décrites au point 5.2.4. Elle comportait également un bug : les désélections au sein d'une sélection multiple étaient ignorées. Un événement supplémentaire a été ajouté pour corriger cela.

¹ Source : SourceForge, .NET Multi-Select TreeView, <https://sourceforge.net/projects/mstreeview>, 16/02/2016.

Obtention des données

La première étape pour obtenir les données des arbres consiste à récupérer l'ensemble des templates via GRAccess. Malheureusement, aucune méthode ne permet d'obtenir des données sans condition. Une condition toujours vraie a donc été utilisée.

```
// récupérer tous les templates de base
IgObjects templates =
    OpenGalaxy.QueryObjects(EgObjectIsTemplateOrInstance.gObjectIsTemplate,
                           EConditionType.basedOn, "-N/A-",
                           EMatch.NotMatchCondition);
if (OpenGalaxy.CommandResult.Successful == false || (Object)templates == null)
    throw new Exception(OpenGalaxy.CommandResult.Text);
```

Parmi les résultats se trouvent, outre les templates, une série d'objets systèmes, de serveurs et de vues. Ces éléments doivent être ignorés. Les autres objets sont ajoutés à un dictionnaire de données, qui servira de base à la création des arbres.

```
// création index = dictionnaire de noeuds templates
var templatesIndex = new Dictionary<string, ITreeNode<GalaxyObject>>();
var ignoredBaseElements = new HashSet<string> { "$AppEngine", "$Area",
                                                "$DDESuiteLinkClient", "$InTouchViewApp",
                                                "$InTouchProxy", "$OPCClient", "$RedundantDIOObject",
                                                "$ViewEngine", "$WinPlatform" };

// remplir index
foreach (IgObject tmpl in templates)
{
    if (!ignoredBaseElements.Contains(tmpl.basedOn)) // ne pas ajouter si ignoré
    {
        GalaxyObject template =
            Converter.ApiToGalaxyObject(true, tmpl, null, true, false, false);
        var newItem = new ControlTreeNode<GalaxyObject>(template,
                                                       template.GetIcon,
                                                       template.Parent);
        templatesIndex.Add(newItem.Content.ToString(), newItem); // ajout noeud
    }
}
// ordonner index
templatesIndex = templatesIndex.OrderBy(key => key.Key)
                                .ToDictionary((keyItem) => keyItem.Key,
                                              (valueItem) => valueItem.Value);
```

Les données du dictionnaire sont suffisantes pour créer l'arbre de dérivation (hiérarchie). En revanche, aucune information n'est fournie au sujet des dossiers de classement, nécessaires pour l'arbre « toolbox ». GRAccess ne permet pas l'obtention de ces informations, c'est pourquoi elles sont récupérées via des requêtes SQL.

L'arbre de dérivation (tout comme l'arbre « toolbox ») possède un index. En dehors de sa hiérarchie naturelle de noeuds, il possède un dictionnaire de données reprenant la référence de chaque noeud. Le nom du template d'un noeud sert de clé dans le dictionnaire d'indexation.

Deux requêtes SQL sont nécessaires pour être apte à construire l'arbre « toolbox ». La première a simplement pour but de fournir la liste complète des dossiers de classement. Cependant, certains dossiers systèmes doivent être exclus des résultats, ainsi que leurs descendants.

```
-- requête dossiers
SELECT Res.folder_id AS folder_id, Res.folder_name AS folder_name,
       Res.parent_folder_id AS parent_folder
FROM dbo.folder AS Res
WHERE Res.folder_name IS NOT NULL
      AND (Res.parent_folder_id != 0
            OR Res.folder_name NOT IN ('System', 'Device Integration',
                                         'Archestra Symbol Library', 'DWH Library'))
      AND Res.parent_folder_id NOT IN
          (SELECT Exclude.folder_id FROM dbo.folder AS Exclude
           WHERE Exclude.folder_name IS NOT NULL
                 AND Exclude.parent_folder_id = 0
                 AND Exclude.folder_name IN ('System', 'Device Integration',
                                              'Archestra Symbol Library', 'DWH Library'))
ORDER BY Res.folder_name;
```

La requête ci-dessus élimine les dossiers indésirables et leurs descendants directs. Elle pourrait toutefois retourner des descendants indirects. Par mesure de sécurité, un algorithme vérifie récursivement si les dossiers qui ne sont pas à la racine possèdent un parent présent dans les résultats. Les éléments ne répondant pas à ce critère sont éliminés.

La deuxième requête SQL nécessaire a pour objectif de déterminer les noms des templates rangés dans chaque dossier. Pour ce faire, elle récupère l'ensemble des associations template/dossier.

```
-- requête associations
SELECT COALESCE(dbo.folder.folder_id, -1) AS folder_id, dbo.gobject.tag_name
FROM dbo.folder_gobject_link
INNER JOIN dbo.gobject ON dbo.folder_gobject_link.gobject_id =
    dbo.gobject.gobject_id
LEFT JOIN dbo.folder ON folder_gobject_link.folder_id = dbo.folder.folder_id
WHERE dbo.gobject.tag_name IS NOT NULL
      AND dbo.gobject.is_template = 1
      AND folder_name IS NULL
      OR (parent_folder_id != 0
           OR folder_name NOT IN ('System', 'Device Integration',
                                   'Archestra Symbol Library', 'DWH Library'))
      AND parent_folder_id NOT IN
          (SELECT Exclude.folder_id FROM dbo.folder AS Exclude
           WHERE Exclude.folder_name IS NOT NULL
                 AND Exclude.parent_folder_id = 0
                 AND Exclude.folder_name IN ('System', 'Device Integration',
                                              'Archestra Symbol Library', 'DWH Library')) )
ORDER BY dbo.gobject.tag_name;
```

Après l'exécution de ces deux requêtes, l'arbre « toolbox » peut être créé. Il se sert de l'index de l'arbre de dérivation pour se procurer la référence de chaque template à partir de son nom. Les objets templates sont donc communs aux deux arbres, puisqu'ils y sont désignés par référence. Grâce à ce système, la modification d'une valeur dans un arbre se répercute également sur l'autre.

Arbres de données avec indexation

La construction des arbres de données se base sur un dictionnaire comprenant l'ensemble des nœuds. Chaque entrée du dictionnaire est identifiée par le nom du template contenu dans son nœud. La mise en place d'une hiérarchie devient alors aisée. En effet, chaque template contient le nom de son parent parmi ses données. Ce nom peut être utilisé pour récupérer le nœud parent dans le dictionnaire et y ajouter le nœud courant en tant qu'enfant.

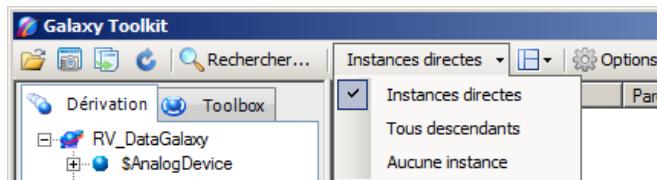
L'algorithme est détaillé ci-dessous. Suite à la création de l'arbre, le dictionnaire initial peut être utilisé en tant qu'index, puisqu'il comprend les références de l'ensemble des nœuds.

```
public void BuildFromIndex(Dictionary<string, ITreeNode<T>> lookup, bool isUpdate)
{
    // création arbre via dictionnaire
    foreach (ITreeNode<T> item in lookup.Values)
    {
        // a un parent nul ou cyclique -> enfant de racine
        if (item.ParentName == null || item.ParentName.Equals(item.ToString()))
            item.Parent = this.RootNode;
        else // a un parent normal
        {
            ITreeNode<T> parentNode;
            if (lookup.TryGetValue(item.ParentName, out parentNode))
                item.Parent = parentNode;
            else // parent inexistant -> racine
                item.Parent = this.RootNode;
        }
        // insertion comme enfant du parent
        item.Parent.AddChild(item);
    }

    // copier dictionnaire en tant qu'index
    if (isUpdate)
        this.NodesIndex = lookup;
    else
        this.NodesIndex = new Dictionary<string, ITreeNode<T>>(lookup);
    NodesIndex.Add(this.RootContent.ToString(), this.RootNode); // racine
}
```

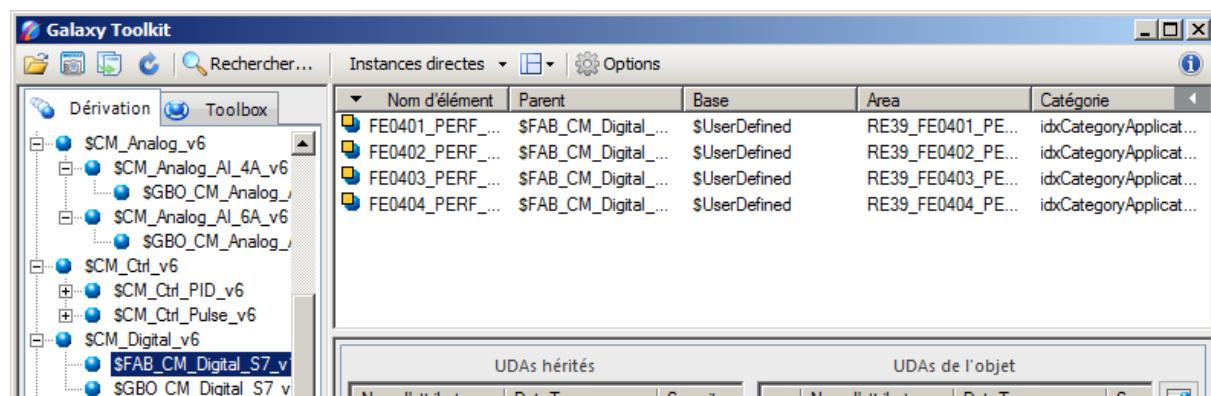
5.2.3 - Liste de données avec filtres

Il m'a semblé judicieux de prévoir plusieurs comportements lorsqu'un ou plusieurs templates sont sélectionnés. Le plus logique est bien entendu de récupérer les instances de ce template et de les afficher. Cependant, il peut parfois se révéler utile d'afficher l'ensemble des instances de tous les descendants du template. Pour cette raison, j'ai ajouté une option à mon application. Elle permet de choisir quel comportement adopter.



Un troisième cas a aussi été envisagé : ne rien récupérer du tout. En effet, l'obtention des instances prend un certain temps. Si l'utilisateur souhaite réaliser une sélection complexe de templates en vue d'un export, il peut s'avérer préférable de ne pas charger des objets chaque fois qu'il modifie sa sélection.

En supposant que l'utilisateur ait choisi de récupérer des instances, le mécanisme s'enclenchera chaque fois qu'il modifie sa sélection dans l'arbre courant. S'il change d'onglet, pour afficher l'autre arbre, les instances s'adapteront alors immédiatement à la sélection du nouvel arbre visible.



Lorsque l'événement de sélection ou de désélection d'un arbre est enclenché, la liste des noms de tous les templates sélectionnés dans l'arbre courant est récupérée. Le modèle de données vérifie, pour chaque template cité, si ses instances ont déjà été chargées auparavant. En effet, après leur récupération, les instances sont conservées en mémoire, dans une liste située à l'intérieur de l'objet template dont elles dérivent.

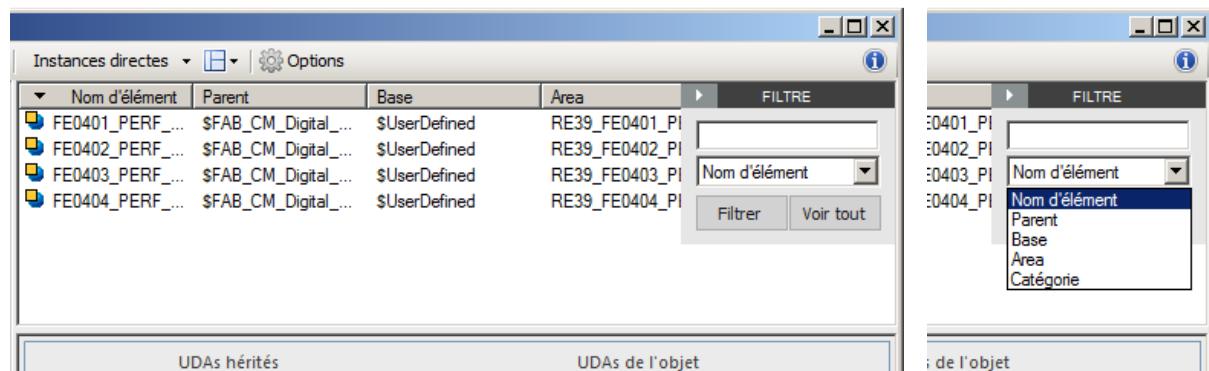
Les instances non présentes en mémoire sont récupérées via une requête de GRAccess. Leurs valeurs sont ensuite recopiées dans des objets conteneurs indépendants de l'API.

```
// récupérer des instances dérivées de templates spécifiés
IgObjects instances =
openGalaxy.QueryObjects(EgObjectIsTemplateOrInstance.gObjectIsInstance,
    EConditionType.derivedOrInstantiatedFrom, templateNames,
    EMatch.MatchCondition);
if (openGalaxy.CommandResult.Successful == false) { /* erreur */ }
```

Ce système de mémorisation des instances en mémoire fait partie des optimisations que j'ai réalisées. Il fournit un gain de temps considérable, lorsque des données sont affichées plus d'une fois. L'utilisateur a toutefois la possibilité de désactiver le cache mémoire (dans les options de l'application). Il est également possible de vider le cache à un instant donné, via le bouton de rafraîchissement présent dans la barre de menu.

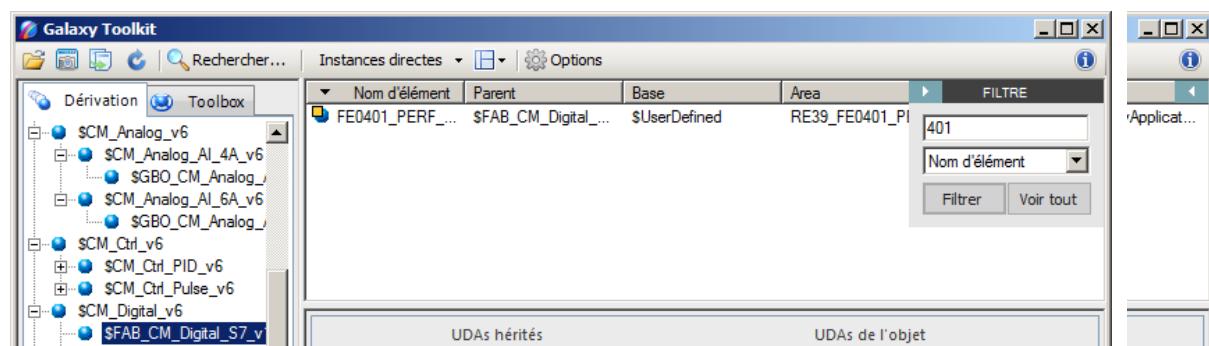
Filtrage des instances

Lorsque la liste d'instances reçoit des données, suite à une sélection dans un arbre, elle conserve une copie temporaire de celles-ci. Ceci permet l'utilisation rapide de filtres. L'utilisation d'un filtre permet d'afficher seulement les instances répondant à un certain critère. C'est une sorte de recherche rapide dans les éléments de la liste.



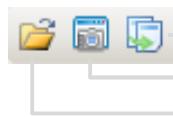
Deux listes d'instances sont donc présentes au sein du contrôleur : la liste initiale, comportant toutes les données, et la liste affichée, répondant au critère éventuel de filtrage. Ainsi, si le filtre est modifié, activé ou désactivé, la liste initiale peut être réutilisée, afin de générer une nouvelle liste d'affichage des éléments répondant au nouveau critère.

Lorsqu'un filtre est choisi, un indicateur coloré devient vert. De cette façon, l'utilisateur peut savoir, au premier coup d'œil, si un filtre est actif ou non. Même lorsque le panneau de filtrage est rétréci (comme sur l'image de droite), l'indicateur reste visible. Quand aucun filtre n'est utilisé, il est de couleur grise.



5.2.4 - Sauvegarde et chargement de l'état des arbres

Mon application permet, à un instant donné, de mémoriser l'état exact des arbres de données : le déploiement de chaque nœud (ouvert/fermé) et sa sélection ou non. Le résultat est enregistré dans un fichier *snapshot*. Il peut ensuite être chargé, afin de restaurer l'état des arbres. Les deux premières icônes de la barre de menu sont dédiées à cette fonctionnalité. La troisième icône effectue une synchronisation des arbres, en sélectionnant dans l'arbre masqué les mêmes nœuds que ceux sélectionnés dans l'arbre visible.



- Synchronisation de la sélection de l'arbre masqué (à partir de celle de l'arbre actif).
- Sauvegarde de l'état courant des arbres dans un *snapshot*.
- Chargement d'un *snapshot* pour restaurer un état antérieur.

La sauvegarde d'un *snapshot* nécessite le parcours de la hiérarchie des arbres, en analysant exclusivement la descendance des nœuds ouverts.

Les fichiers générés sont similaires au texte ci-dessous. Chaque nœud ouvert et/ou sélectionné est indiqué sous l'identifiant de l'arbre correspondant. Une valeur numérique permet d'indiquer son état : ouvert (1), sélectionné (2) ou les deux à la fois (3).

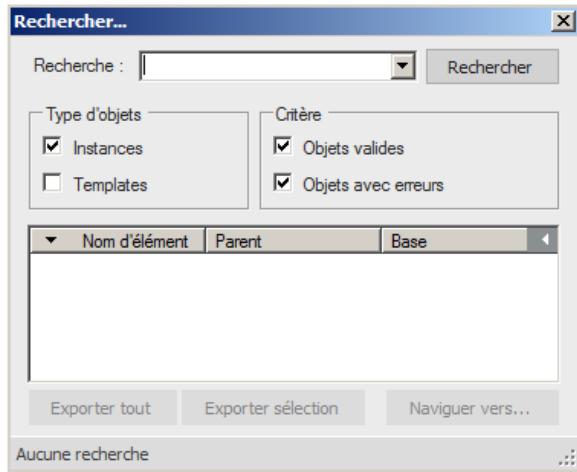
```
---DERIVATION---
RV_DataGalaxy=1
$UserDefined=1
$GBO_UserDefined_v6=1
$AutomationDevice_v6=1
$ControlModule_v6=1
$TEC_CM_Analog_AI_2A_v1=1
$TEC_CM_Analog_AI_v1=1
$TEC_CM_Analog_AI_2A_Sx_v1=2
$TEC_CM_Analog_AI_Sx_v1=2
---TOOLBOX---
RV_DataGalaxy=1
76=1
$TEC_CM_Analog_AI_2A_Sx_v1=2
$TEC_CM_Analog_AI_Sx_v1=2
```

Grâce au système d'index des arbres, il n'est pas nécessaire de parcourir toute leur hiérarchie pour restaurer un *snapshot*. Il suffit en effet de cibler directement dans l'index les éléments déployés ou sélectionnés, et de demander leur ouverture et/ou leur sélection. Il est par contre nécessaire de suivre leur chaînage de parenté, afin d'ouvrir tous les éléments dont ils dérivent.

La synchronisation des arbres, quant à elle, se contente de récupérer la sélection de l'arbre courant et d'activer les nœuds correspondants dans l'autre arbre. L'ajout d'une méthode à la librairie de gestion des arbres a été nécessaire, afin de permettre la sélection d'un ou plusieurs nœud(s) par programmation.

5.2.5 - Fenêtre de recherche

Mon application dispose d'une fenêtre de recherche. Celle-ci permet de trouver des objets selon plusieurs critères : leur type (template et/ou instance) et leur statut (valide et/ou erroné).



La liste de résultats est similaire à la liste d'instances présentée au point 5.2.3. Elle permet de trier les données selon l'une des colonnes affichées. Il est aussi possible de filtrer les résultats. Trois actions peuvent être réalisées après une recherche : placer tous les résultats dans la liste d'export (décrise au point 5.2.8), placer uniquement les éléments sélectionnés dans cette liste d'export, ou naviguer vers la sélection dans l'arbre actif et dans la liste d'instances.

Plusieurs critères permettent d'affiner la recherche.

The image displays two instances of the 'Rechercher...' dialog box side-by-side, illustrating how different filter combinations affect the search results.

Left Dialog (Search: 'boil'):

- Type d'objets:** Instances (checked), Templates (unchecked)
- Critère:** Objets valides (checked), Objets avec erreurs (checked)
- Results Table:**

Nom d'élément	Parent	Base
Analog_BoilerIO_...	\$BOIL_A3	\$AnalogDevice
BOIL_A1	\$BOIL_A3	\$AnalogDevice
BOIL_A2	\$BOIL_A3	\$AnalogDevice
- Status:** 3 résultats trouvés (3 results found)

Right Dialog (Search: 'boil'):

- Type d'objets:** Instances (checked), Templates (checked)
- Critère:** Objets valides (checked), Objets avec erreurs (checked)
- Results Table:**

Nom d'élément	Parent	Base
\$BOIL_A3	\$AnalogDevice	\$AnalogDevice
Analog_BoilerIO_...	\$BOIL_A3	\$AnalogDevice
BOIL_A1	\$BOIL_A3	\$AnalogDevice
BOIL_A2	\$BOIL_A3	\$AnalogDevice
- Status:** 4 résultats trouvés (4 results found)

Recherche de données en SQL

La première étape de la recherche consiste à extraire les paramètres et la valeur saisie. Ces données sont ensuite transmises à une procédure de recherche en SQL. Celle-ci commence par l'analyse des paramètres. Elle génère une condition SQL dans une chaîne de caractères, destinée à être concaténée à la véritable requête. Un système de *binding* est utilisé pour la valeur recherchée : elle n'est pas écrite directement dans le texte de la requête. À la place, elle y est associée de manière indirecte. Cette protection permet de prévenir les injections SQL.

```
// conditions de requête SQL
string queryCond = "";
if (type == GalaxyObject.ObjectType.Instances
    || type == GalaxyObject.ObjectType.Templates)
    queryCond += " AND Res.is_template = " + (int)type;
if (status == GalaxyObject.ObjectStatus.Normal
    || status == GalaxyObject.ObjectStatus.Error)
    queryCond += " AND Res.status = " + (int)status;

// nom recherché
SqlParameter[] param = null;
if (search.Length)
{
    queryCond += " AND LOWER(Res.tag_name) LIKE @Search";
    param = new SqlParameter[] { new SqlParameter("@Search", SqlDbType.NVarChar) };
    param[0].Value = "%" + search.ToLower() + "%";
}
```

Une fois la condition spécifiée et le *binding* réalisé, la véritable requête SQL est exécutée.

```
// commande requête
SqlCommand cmd = new SqlCommand();
cmd.Connection = connection;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT Res.is_template, Res.status, Res.is_checkout, " +
    "Res.tag_name, Res.hierarchical_name, " +
    "Parent.tag_name, BaseType.tag_name " +
    "FROM dbo.internal_list_objects_view as Res " +
    "LEFT JOIN dbo.internal_list_objects_view as Parent " +
    "ON Res.derived_from_id = Parent.gobject_id " +
    "LEFT JOIN dbo.internal_list_objects_view as BaseType " +
    "ON Res.base_type = BaseType.gobject_id " +
    "WHERE Res.tag_name IS NOT NULL " +
    "AND Res.base_type NOT IN " +
    "(SELECT Exclude.gobject_id " +
    "FROM dbo.internal_list_objects_view as Exclude " +
    "WHERE Exclude.tag_name IS NOT NULL " +
    "AND Exclude.tag_name IN " +
    "('AppEngine', 'Area', 'DDESuiteLinkClient', " +
    "'InTouchViewApp', 'InTouchProxy', 'OPCClient', " +
    "'RedundantDIOObject', 'ViewEngine', 'Symbol') ) " +
queryCond +
    " ORDER BY Res.tag_name";
```

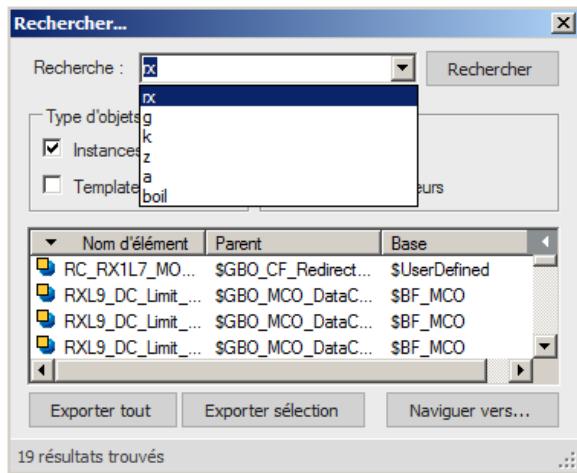
Pour rendre le *binding* effectif, il doit être associé à la commande SQL.

```
// binding paramètres
if (param != null && param.Length > 0)
    foreach (var p in param)
        cmd.Parameters.Add(p);
```

La suite des opérations consiste simplement à démarrer la commande et récupérer les données. Les objets sont ensuite affichés dans la liste de résultats de la même façon que dans la liste contenant les instances.

Historique de recherche

Les valeurs saisies dans le champ de recherche sont mémorisées dans un historique. Sa longueur est fixée par l'utilisateur dans les options de l'application. L'historique permet de consulter les dernières recherches effectuées. Il respecte l'ordre de saisie en plaçant les dernières valeurs en tête. Si une ancienne valeur est réutilisée, elle est simplement déplacée en haut de la liste.

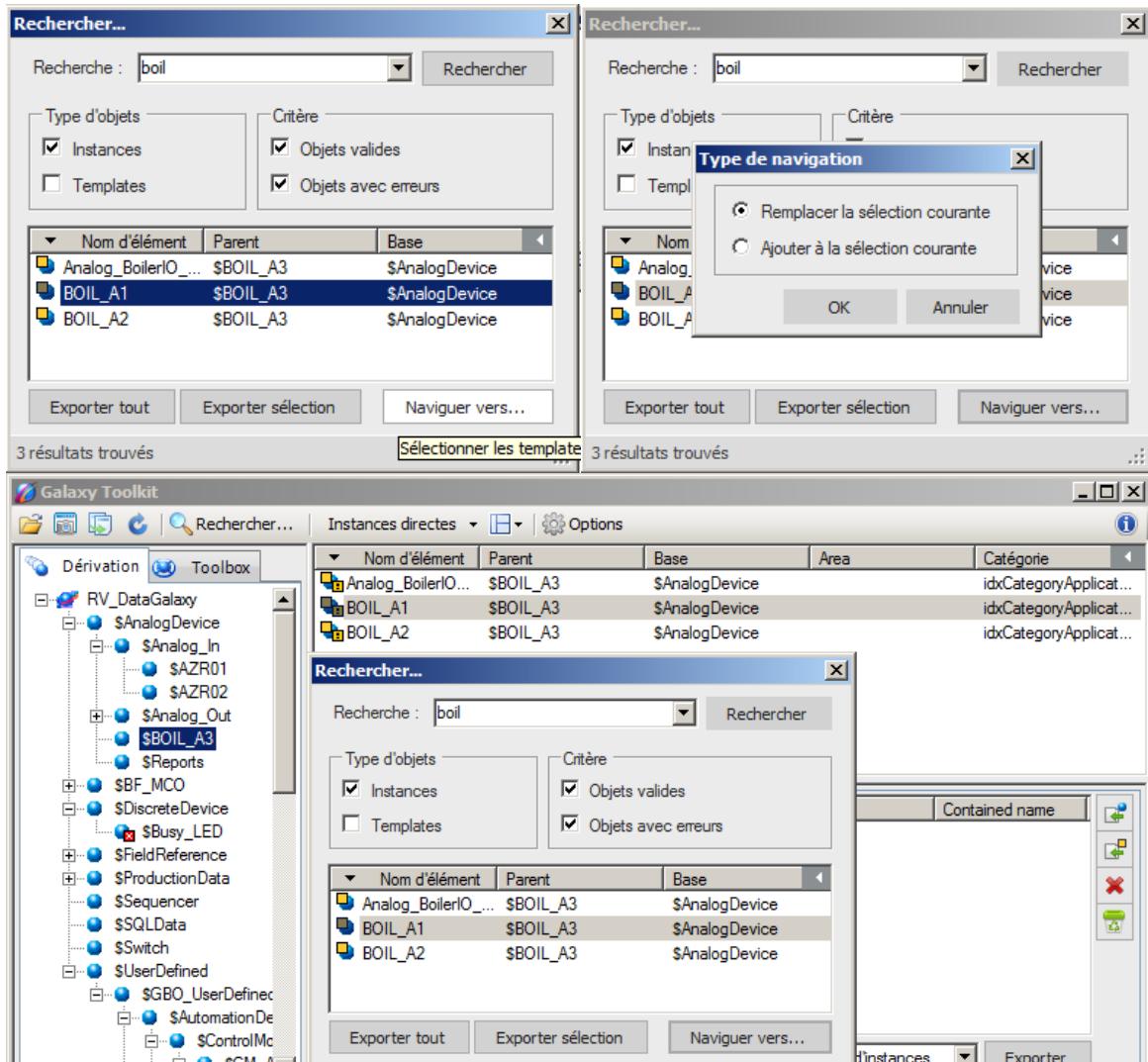


L'historique de recherche repose sur une liste de type *HashSet*. Elle comporte un ensemble de clés, sous forme de chaînes de caractères. Contrairement aux dictionnaires, les *HashSets* n'associent aucune valeur aux clés. Ils servent uniquement à stocker des chaînes de caractères, tout en permettant la vérification directe de l'existence d'une chaîne spécifique parmi les données.

Lorsqu'une valeur est saisie, le *HashSet* permet de savoir si elle figure déjà dans l'historique. Au niveau de l'affichage, une *ComboBox* éditable est utilisée en guise de champ de recherche. Il est donc possible d'y insérer les valeurs saisies, afin de pouvoir les afficher sous forme de liste.

Navigation vers la sélection

Lorsqu'une recherche est effectuée, il est possible de sélectionner un ou plusieurs résultats dans la liste obtenue. La navigation a pour but d'aller chercher ces éléments dans la fenêtre principale (dans l'arbre de templates actif et dans la liste d'instances). Il est parfois difficile de repérer un élément dans la hiérarchie des templates. En utilisant la navigation, cet élément est immédiatement sélectionné et affiché.

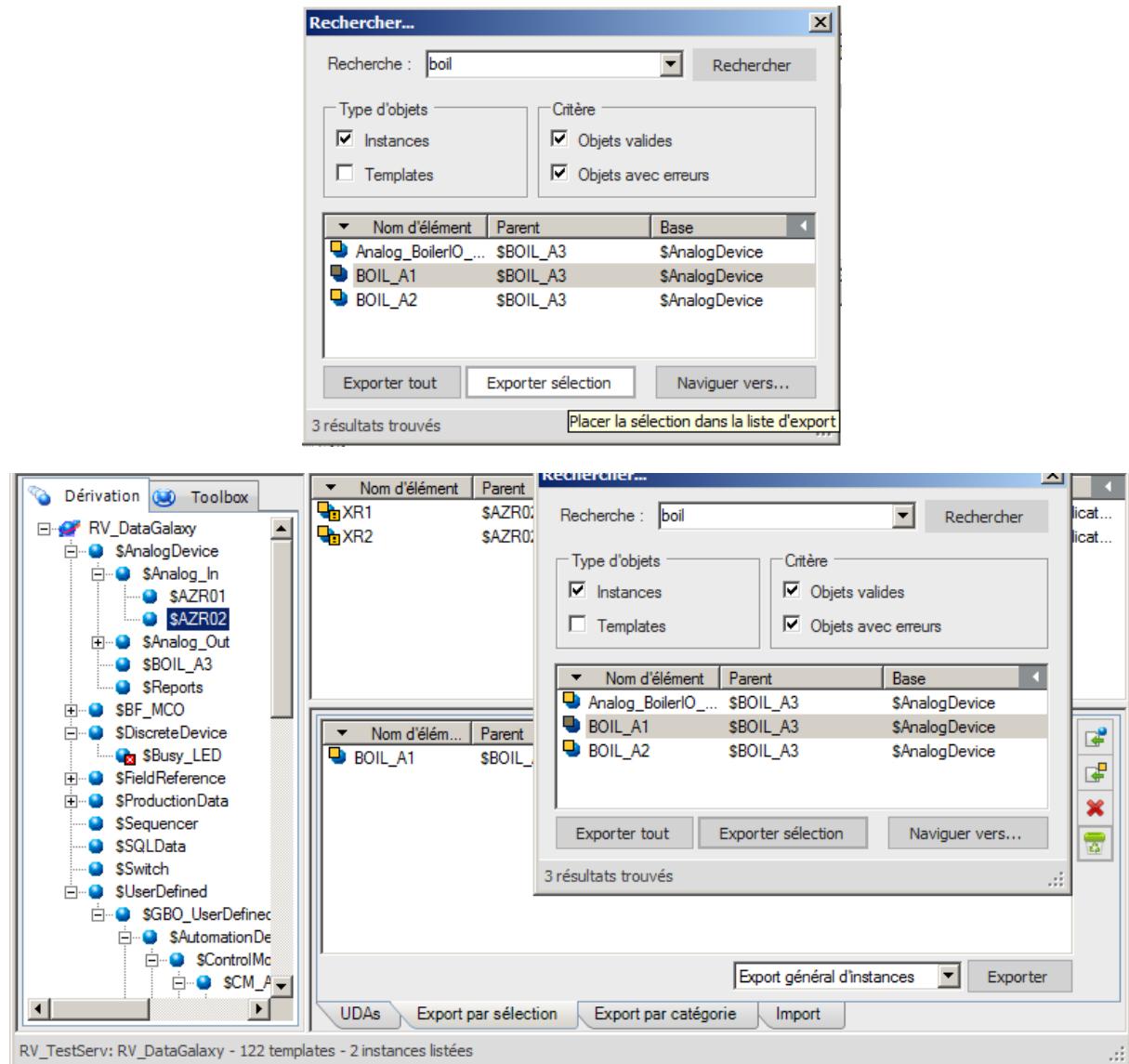


Lorsque le bouton de navigation est pressé, une boîte de dialogue propose un choix : remplacer la sélection courante de l'arbre de templates, ou y ajouter d'autres éléments. Chacune des deux méthodes peut trouver son utilité dans des contextes différents. Le double-clic sur un objet de la liste de résultats est un raccourci de la première méthode, sans passer par la boîte de dialogue.

En termes d'implémentation, la méthode de sélection de templates dans l'arbre courant est similaire à celle utilisée pour la restauration des *snapshots*. Elle appelle une méthode que j'ai ajoutée à la librairie de gestion d'arbres multi-sélections.

Placement dans la liste d'export

L'une des méthodes d'export consiste à utiliser une liste d'objets, permettant un export personnalisé. Cette liste est décrite au point 5.2.8. La fenêtre de recherche permet d'y transférer directement des résultats. Deux boutons offrent le choix entre un transfert de tous les éléments ou uniquement de ceux sélectionnés.



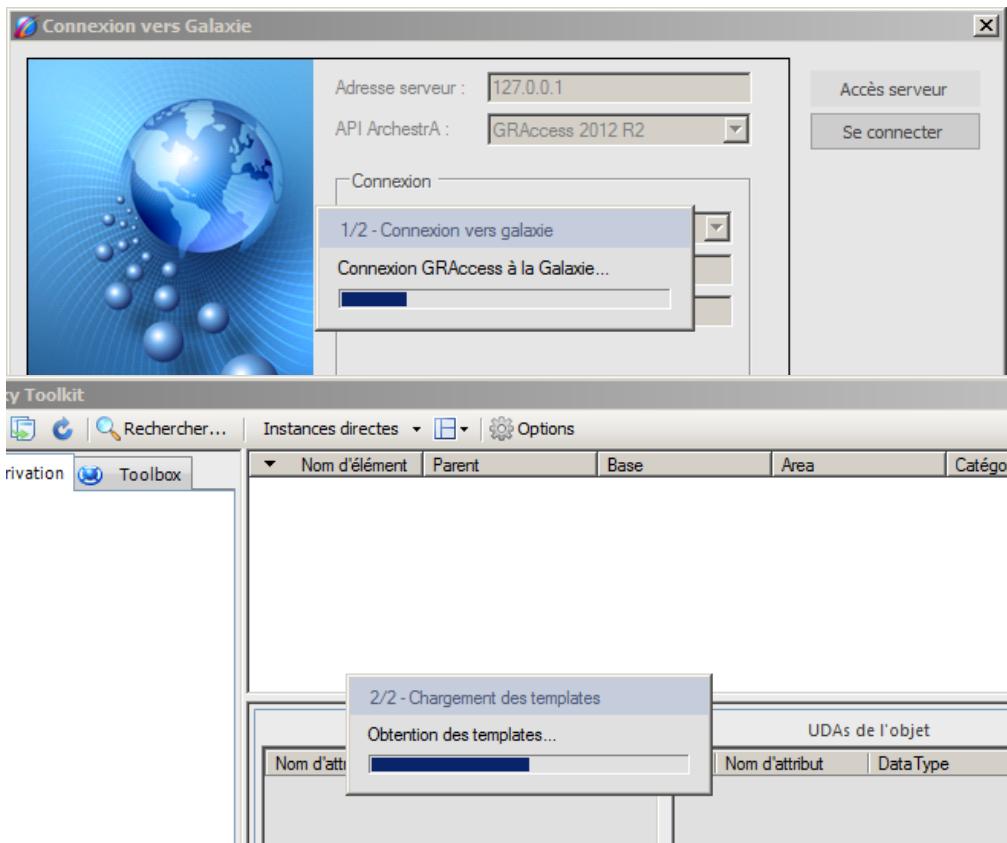
La fenêtre de recherche informe la fenêtre principale, via un événement, qu'elle doit ajouter des éléments à la liste d'export. Ces objets sont contenus dans une variable de liste statique de la classe de recherche, ce qui les rend accessibles à la fenêtre principale. Dès la réception du signal, celle-ci transfère les éléments vers la liste d'export.

5.2.6 - Gestion des tâches et file d'attente

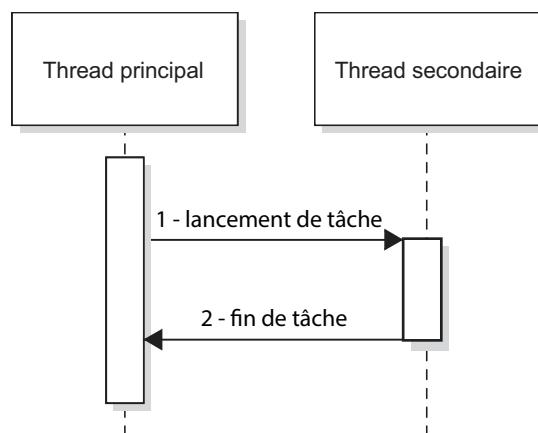
Une particularité de mon application est l'exécution asynchrone des tâches d'obtention, d'import ou encore d'export de données. Celle-ci a l'avantage de ne pas bloquer l'interface graphique. Elle offre aussi la possibilité de gérer et afficher une boîte de progression, qui informe l'utilisateur sur l'action en cours et sur l'état d'avancement.

Cette boîte de progression fournit des indications à propos du nombre d'étapes en attente, du nom de l'étape actuelle et de la sous-opération en cours.

Dès le démarrage de l'application, une boîte de progression apparaît pendant le chargement.



Grâce à la nature asynchrone des tâches, le thread principal n'est pas bloqué.



Gestion de la file d'attente

La file d'attente respecte un schéma producteur/consommateur. Chaque fois qu'une action est demandée par l'utilisateur, elle s'ajoute derrière la dernière tâche ayant la même priorité. La première tâche de la file est exécutée. Une fois terminée, elle est retirée de la file.



La fonction d'ajout d'une nouvelle tâche reçoit comme paramètres un objet *Task*, ainsi que plusieurs informations complémentaires : le nom de la tâche, la référence d'une fonction appelée après l'exécution, le type de tâche, sa priorité, et le fait qu'elle puisse ou non être interrompue. L'objet *Task* contient les instructions à exécuter.

L'ajout d'une tâche consiste donc à l'insérer dans la file, selon sa priorité. Si la file est vide, la tâche est directement exécutée.

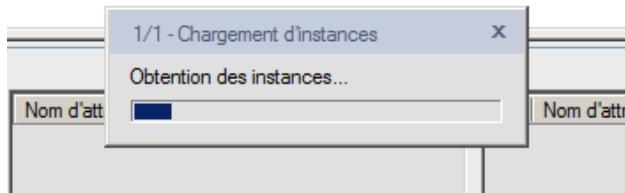
```
// insertion de tâche
_mutexQueue.WaitOne();
try
{
    // instancier tâche et définir callback
    TaskListItem newTask = new TaskListItem(action, callback, name, type,
                                              prior, isInterruptable);
    newTask.AsyncTask.ContinueWith(t =>
    {
        TaskListItem.RemoveCompletedTask(t.Exception);
    }, TaskScheduler.FromCurrentSynchronizationContext());

    // file vide -> ajout simple et exécution
    if (_tasksQueue.Count == 0)
    {
        // boîte de progression
        // [...]
        // ajout de tâche
        _tasksQueue.AddFirst(newTask);
        _tasksQueue.First.Value.AsyncTask.Start(); // lancement
    }
    // file déjà utilisée -> ajout/remplacement selon priorité et si remplaçable
    else
    {
        InsertTask(newTask);
        // mise à jour boîte de progression
        // [...]
    }
}
catch (Exception exc) { /* [...] */ }
_mutexQueue.ReleaseMutex();
```

Lorsqu'une tâche s'achève, elle est redirigée vers une fonction se chargeant de la retirer de la file, de démarrer la tâche suivante, et d'appeler la fonction associée à la fin de la tâche terminée.

Interruption d'une tâche

La plupart des tâches réalisées par mon application sont interruptibles. Cela signifie que l'utilisateur dispose d'une petite croix, qui permet d'y mettre fin.



En vérité, il n'est pas possible d'interrompre une tâche depuis l'extérieur. Seule la tâche proprement dite peut décider de s'arrêter. La méthode d'arrêt sur demande consiste donc à utiliser un indicateur d'annulation, que la tâche doit consulter régulièrement. Elle peut ainsi choisir de se terminer, lorsqu'une demande d'arrêt est détectée.

En utilisant un objet *CancellationToken*, la gestion des annulations est très simple. La fonction qui suit permet de réaliser une demande d'interruption.

```
///<summary>Demander l'arrêt de la tâche courante (si disponible)</summary>
public static void CancelCurrentTask()
{
    if ((Object)_tasksQueue != null && _tasksQueue.Count > 0)
        _tasksQueue.First.Value.Canceler.Cancel();
}
```

Pour être informée de cette demande, la tâche doit consulter l'objet *CancellationToken*.

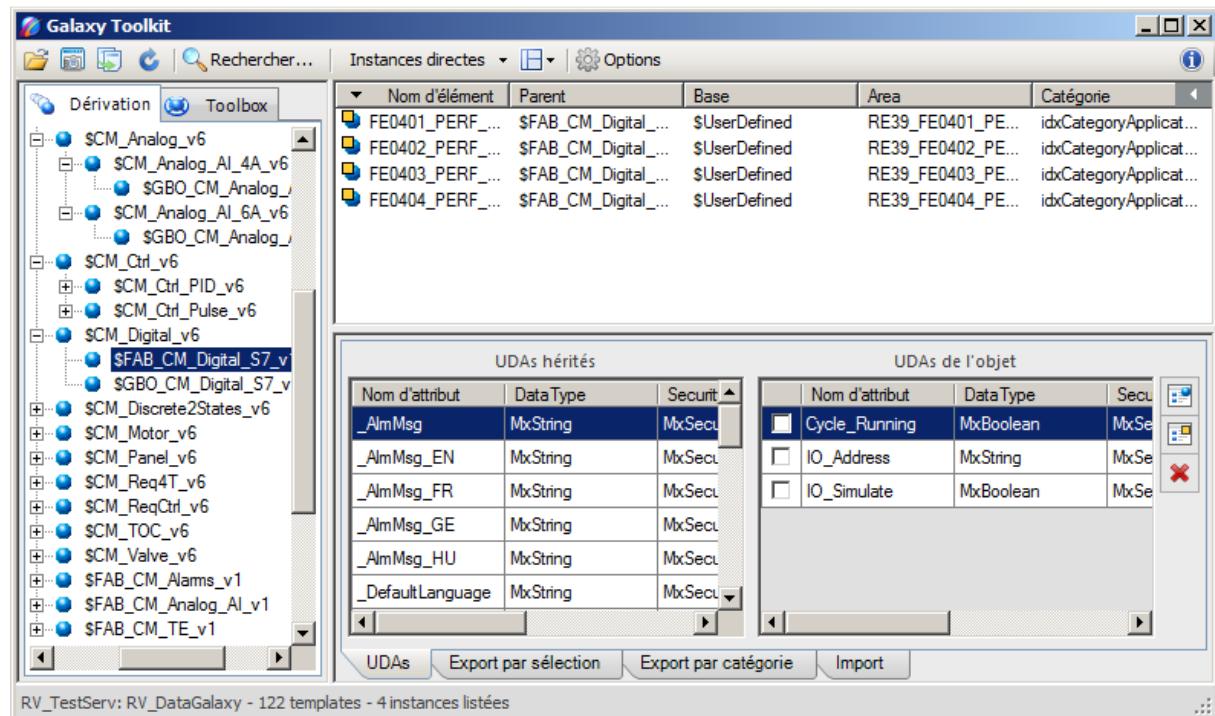
```
///<summary>Vérifier si la tâche courante doit être arrêtée</summary>
public static bool IsTaskCanceled()
{
    if ((Object)_tasksQueue == null || _tasksQueue.Count == 0)
        return false;
    return _tasksQueue.First.Value.Canceler.Token.IsCancellationRequested;
}
```

Elle peut alors choisir de s'arrêter. En effectuant des vérifications régulières du *CancellationToken*, la tâche devient interruptible.

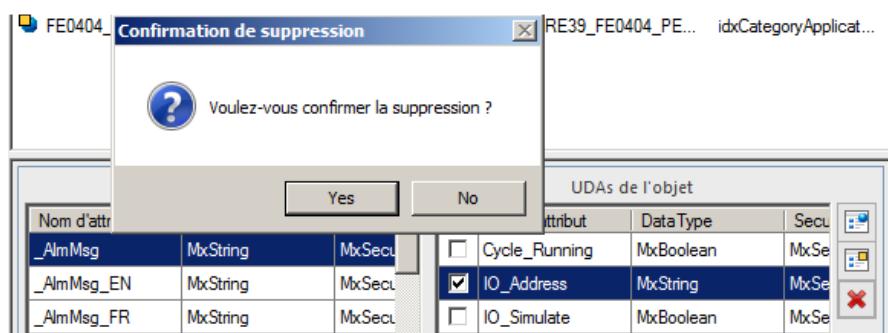
5.2.7 - Affichage et suppression d'attributs UDAs

L'obtention des attributs *UDAs* via GRAccess est détaillée dans le chapitre d'analyse. Son principe repose sur la lecture d'un « faux » attribut de l'objet concerné, qui contient une liste XML de tous les *UDAs* propres ou hérités (selon l'attribut). En déserialisant cette liste, le nom et le type des attributs *UDAs* sont obtenus. Il suffit alors d'utiliser leur nom pour les repérer dans la liste de tous les attributs de l'objet. La connaissance de leur type permet de décoder leur valeur efficacement.

L'onglet *UDAs* de la zone contextuelle de mon application permet de visualiser la liste des *UDAs* propres et hérités d'un objet. Il permet également la sélection et la suppression des *UDAs* propres à l'élément. En revanche, les *UDAs* hérités ne peuvent pas être supprimés. En effet, ils n'appartiennent pas à l'objet qui en hérite, ce qui ne permet pas leur élimination de la part de ce dernier.



Une boîte de dialogue demande une confirmation en cas de suppression, afin d'éviter les fausses manœuvres. Au niveau de l'implémentation, la suppression d'un *UDA* fait simplement appel à une méthode prévue à cet effet dans GRAccess, nommée *DeleteUDA*. Son utilisation est très simple et ne demande, comme seule précaution, qu'une vérification de la réussite de l'opération.



5.2.8 - Import et export de données

Export par sélection

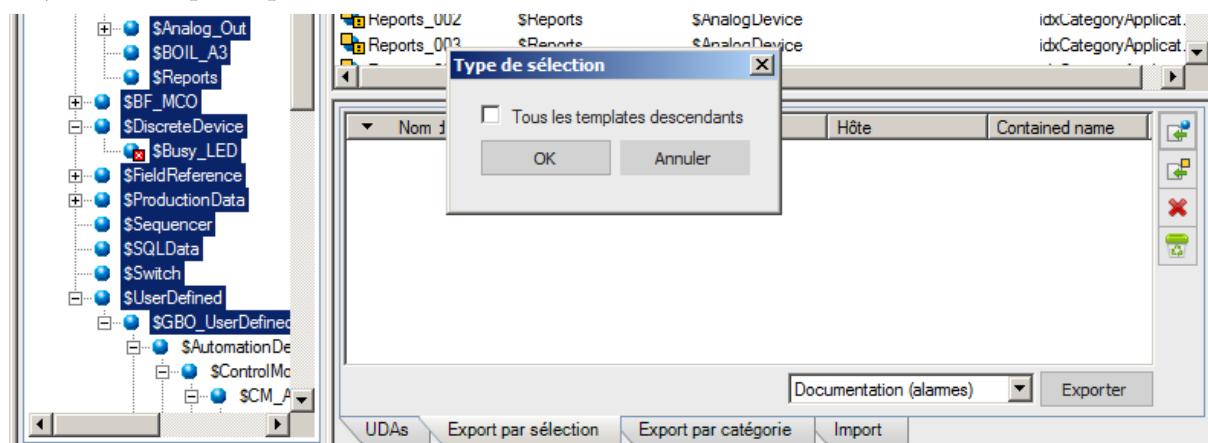
La zone contextuelle de mon application propose deux méthodes d'export : par sélection ou par catégorie. L'export par sélection consiste à remplir une liste avec les éléments souhaités, avant de lancer la procédure. Il permet donc une sélection personnalisée des objets.

Le menu de la liste d'export par sélection propose quatre actions :

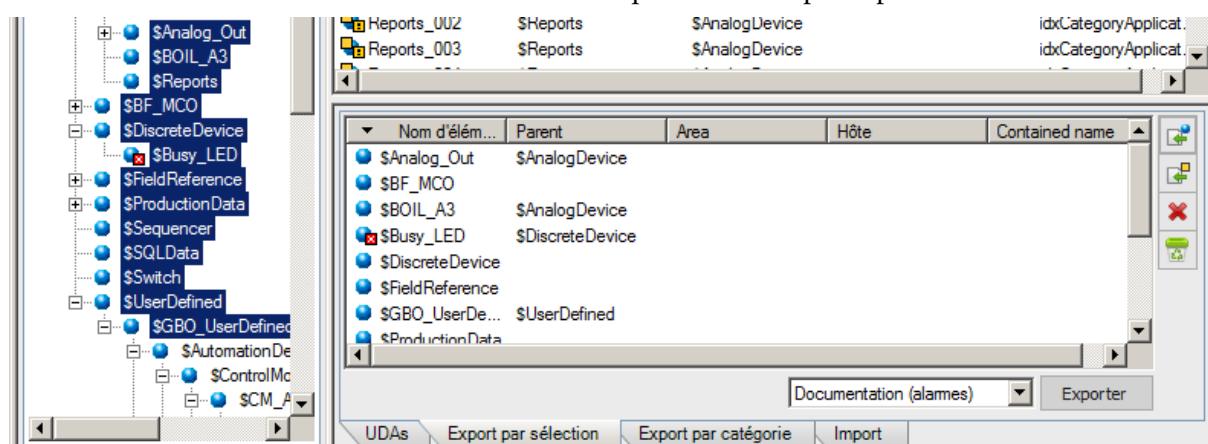


- Ajouter à la liste les templates sélectionnés dans l'arbre courant.
- Ajouter à la liste les éléments sélectionnés dans la liste d'instances.
- Retirer un objet de la liste d'export.
- Vider la liste d'export.

L'ajout de templates permet d'inclure tous les descendants de la sélection, si nécessaire.

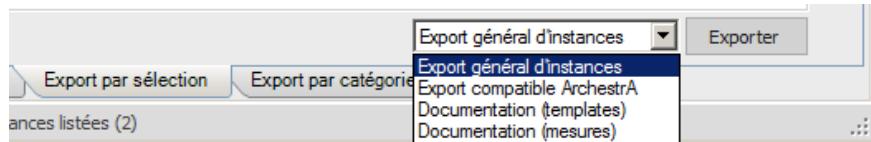


La liste d'export affiche tous les éléments qu'elle contient. Son utilisation est similaire à celle de la liste d'instances et des résultats de recherche, sauf qu'elle ne comporte pas de filtres.



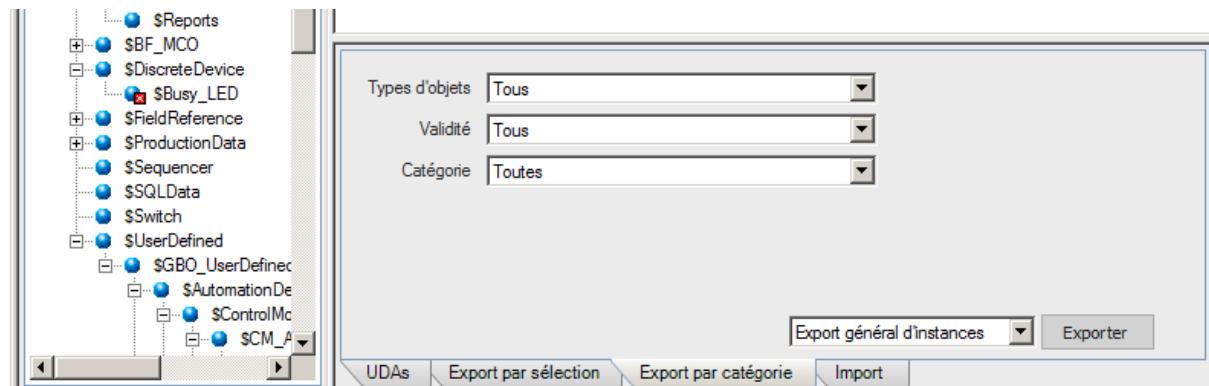
Il est possible d'utiliser plusieurs « raccourcis » avec la souris. Le double-clic sur un template de l'arbre ou sur un élément de la liste d'instances utilise automatiquement cet objet dans l'onglet ouvert de la zone contextuelle. Si l'onglet d'export est ouvert, l'objet est ajouté dans sa liste. Si c'est l'onglet « *UDAs* » qui est actif, le double-clic se contente d'afficher ceux de l'élément. La liste d'instances peut également réaliser un *drag and drop* vers la zone contextuelle.

Une liste de sélection, située en bas à droite de la zone contextuelle, permet de choisir le type d'export réalisé : l'export général (importable), l'export compatible avec ArchestrA ou l'export de rapports de documentation (templates, mesure ou alarmes).

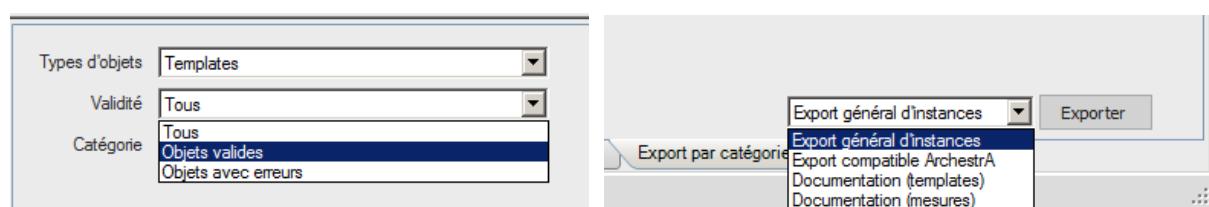


Export par catégorie

L'export par catégorie ne permet pas le choix d'éléments au cas par cas. En revanche, il offre la possibilité d'exporter l'ensemble des objets répondant à certains critères : un type défini (templates ou instances), un statut (valide ou erroné) et une catégorie. C'est aussi un moyen efficace d'exporter l'entièreté d'une galaxie, si aucun critère n'est utilisé.



La liste de sélection, située en bas à droite de la zone contextuelle, offre les mêmes fonctionnalités que celle de l'onglet d'export par sélection.



Export général d'instances

L'export général permet de créer des fichiers Excel pouvant être importés par la suite. L'édition du fichier dans le tableur Excel permet l'ajout et la modification rapide d'un grand nombre d'instances. Le fichier généré par l'export général dispose d'un onglet principal, listant les templates (parents des instances) et leur nombre d'enfants. L'apparence de cette feuille Excel est similaire à celle du cahier des charges. J'y ai toutefois ajouté des informations complémentaires, afin de faciliter le travail des personnes chargées de l'édition du document : la liste des valeurs possibles pour certains paramètres (*DataType*, *Security*, *Locked*, *Category*, ...).

Chaque template dispose de sa propre feuille Excel, où sont listées ses instances. Le nom de l'onglet est censé correspondre au nom du template. En pratique, si le nom est trop long, il est tronqué. J'ai donc mis au point un mécanisme de gestion de troncature, afin d'éviter tout risque de conflit en cas de noms similaires. J'ai également ajouté une case dans la feuille du template, qui contient son nom complet. Cela permet de l'identifier lors d'un import.

	\$Chaux_10_50					UDA	AL2O3	CAO	
	[Import]	Instances	Area	Container	Contained Name	Description	Locked	0	0
							Security	1	1
							DataType	Float	Float
							Is Array	false	false
x	FR1_CHAUX_10_50	FR1_Rapports				Matière 10 50		0,0	0,0
x	FR2_CHAUX_10_50	FR2_Rapports				Matière 10 50		0,0	0,0
x	FR3_CHAUX_10_50	FR3_Rapports				Matière 10 50		0,0	0,0
x	FR4_CHAUX_10_50	FR4_Rapports				Matière 10 50		0,0	0,0

L'obtention des données pour réaliser cet export n'est pas différente des méthodes expliquées précédemment. Les instances sont obtenues de la même façon que dans la liste d'instance. Les *UDAs* sont récupérés de la même manière qu'avec l'onglet « *UDAs* » de la zone contextuelle.

La création du document Excel est réalisée à l'aide de la librairie *EPPlus*. Son utilisation est similaire aux exemples fournis dans le chapitre d'analyse. La principale difficulté rencontrée était la représentation des vecteurs de données. Séparer les éléments par des virgules aurait posé problème pour les vecteurs de nombre flottants. Les points et les deux-points auraient été problématiques avec les vecteurs de valeurs temporelles. La plupart des séparateurs conventionnels auraient posé problème pour les vecteurs de chaînes de caractères.

Le séparateur utilisé est la double barre verticale : `||`. Ce séparateur ne saurait apparaître dans aucun autre type de données que les chaînes de caractères. Par ailleurs, son utilisation dans un vecteur de chaînes de caractères (descriptions, noms, messages...) est peu probable.

Import de données

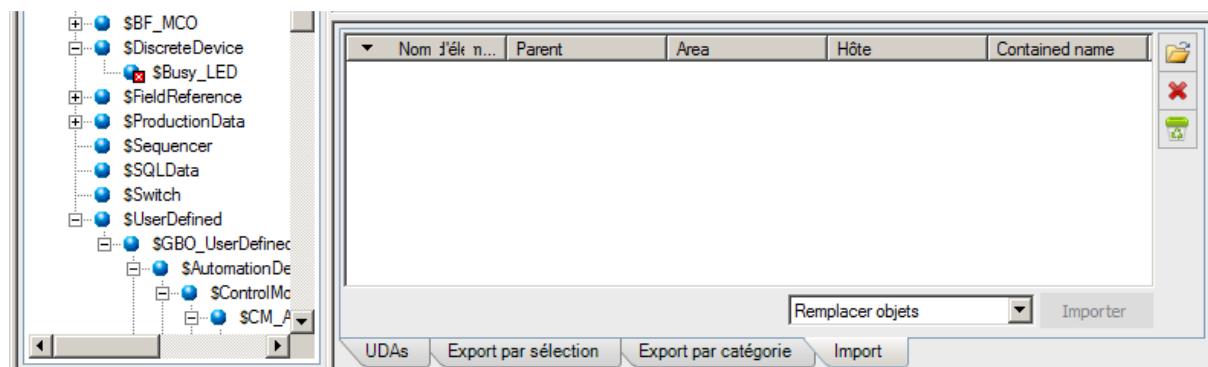
La zone contextuelle de mon application propose un onglet dédié à l'import. Il contient une liste de validation d'éléments. Seuls les documents issus de l'export général (ou créés de manière similaire) peuvent être importés.

Le menu de la liste d'import propose trois actions :



- Ouvrir un fichier d'import et copier ses éléments dans la liste de validation.
- Supprimer des objets de la liste de validation.
- Vider la liste de validation.

La liste de validation permet de choisir les éléments à prendre en compte, parmi ceux trouvés dans le fichier ouvert. Elle permet également de contrôler les erreurs. En effet, en cas d'échec d'import de certains objets, ceux-ci restent présents dans la liste (contrairement aux éléments dont l'import est réussi, qui en sont retirés).



Il peut sembler étrange de trouver des templates dans la liste d'import (voir ci-dessous), alors que seules des instances avaient été exportées. Pourtant, l'ajout ou la modification de colonnes d'attributs *UDAs* dans une feuille Excel de template aboutit à sa modification. La mise à jour du template sera dès lors nécessaire, avant de pouvoir importer les instances.



La liste de sélection, située en bas de la zone contextuelle, permet de choisir l'attitude à adopter vis-à-vis des objets déjà existants : les écraser ou les ignorer.



5.2.9 - Export compatible avec ArchestrA

Les exports réalisés par ArchestrA IDE ont une structure très simple. Il s'agit de fichiers CSV dont le contenu ne s'apparente pas à un tableau, mais plutôt à une série de longues listes d'attributs.

Comme le montre l'exemple ci-dessous, les instances sont regroupées par template. Pour chaque groupe, la première ligne indique le nom du template. La seconde ligne est une liste de tous les attributs/extensions qui existent dans ArchestrA, quels qu'ils/elles soient. Les lignes suivantes sont chacune dédiées à une instance spécifique. Elles contiennent une liste de toutes les valeurs associées aux attributs/extensions mentionné(e)s dans la deuxième ligne.

```
; Created on: 19-05-2016 18:38:29 from Galaxy: RV_DataGalaxy

:TEMPLATE=$AnalogDevice
:TagName,Area,SecurityGroup,Container,ContainedName,ShortDesc,ExecutionRel[...]
AnalogDevice_001,,Default,,,Description d'objet.,None, [...]
AnalogDevice_002,,Default,,,Description d'objet.,None, [...]

:TEMPLATE=$DiscreteDevice_001
:TagName,Area,SecurityGroup,Container,ContainedName,ShortDesc,ExecutionRel[...]
[...]
```

Compte tenu de la nature du fichier exporté, il n'est pas possible de réutiliser les objets présents dans le cache mémoire de mon application : ils ne contiennent que les attributs les plus importants, pas une liste complète de toutes valeurs.

La technique la plus simple pour réaliser cet export consiste à récupérer, via GAccess, l'ensemble des instances à exporter. Il s'agit dès lors d'une simple requête basée sur une liste de noms. Une fois les objets récupérés, la totalité de leurs attributs sont recopiés dans une chaîne de caractères propre à chacun d'entre eux.

Chaque instance dispose de sa propre chaîne de caractères, contenue dans une liste, elle-même située dans un dictionnaire dont les clés sont le nom des templates. De cette manière, l'écriture du fichier CSV est grandement simplifiée. Pour récupérer les attributs d'une instance, deux étapes sont nécessaires :

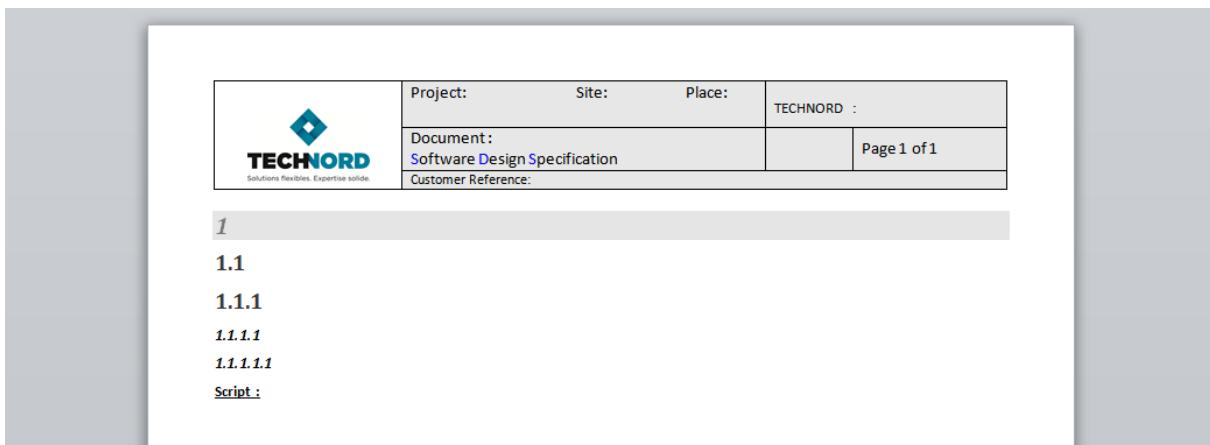
- Copier, une à une, les propriétés élémentaires de l'objet. Leur nombre est plutôt faible.
- Parcourir l'intégralité des attributs contenus dans la propriété *Attributes* de l'objet. Cette propriété renferme un dictionnaire de données, où sont repris tous les attributs. Malheureusement, de « faux » attributs s'y trouvent également, remplissant des rôles particuliers. Ceux-ci doivent absolument être ignorés. Il est donc important de réaliser une liste d'exclusion. Afin de gagner du temps lors des vérifications, la liste d'exclusion est implementée sous forme de *HashSet*. Contrôler si un attribut appartient à cette liste se fait alors de manière directe, sans nécessiter son parcours complet.

5.2.10 - Rapports de documentation Word

Création du document

L'obtention des données pour la génération de rapports au sujet des templates n'est pas très différente des méthodes décrites précédemment. En plus de récupérer les attributs *UDAs*, il est nécessaire de prendre en charge les *field attributes*, les extensions d'*UDAs* et les *scripts*. Cependant, leur méthode d'obtention repose exactement sur le même principe que celui des *UDAs*. Il s'agit du même type de listes XML à déserialiser, permettant d'identifier les éléments ciblés.

La librairie *DocX* est utilisée pour créer le document Word. Son utilisation est assez simple et correspond aux exemples cités dans le chapitre 4. Au niveau des styles toutefois, la librairie ne permet pas la création de styles globaux. Chaque paragraphe doit être mis en forme séparément. Il existe cependant un moyen de contourner ce problème : fournir un canevas de document contenant déjà des styles globaux. En procédant de cette manière, il devient possible d'associer un style global à un paragraphe. L'image ci-dessous montre un aperçu du canevas réalisé.



Le document généré contient plusieurs tableaux. La création de tableaux avec *DocX* est simple, comme le montre l'exemple ci-dessous.

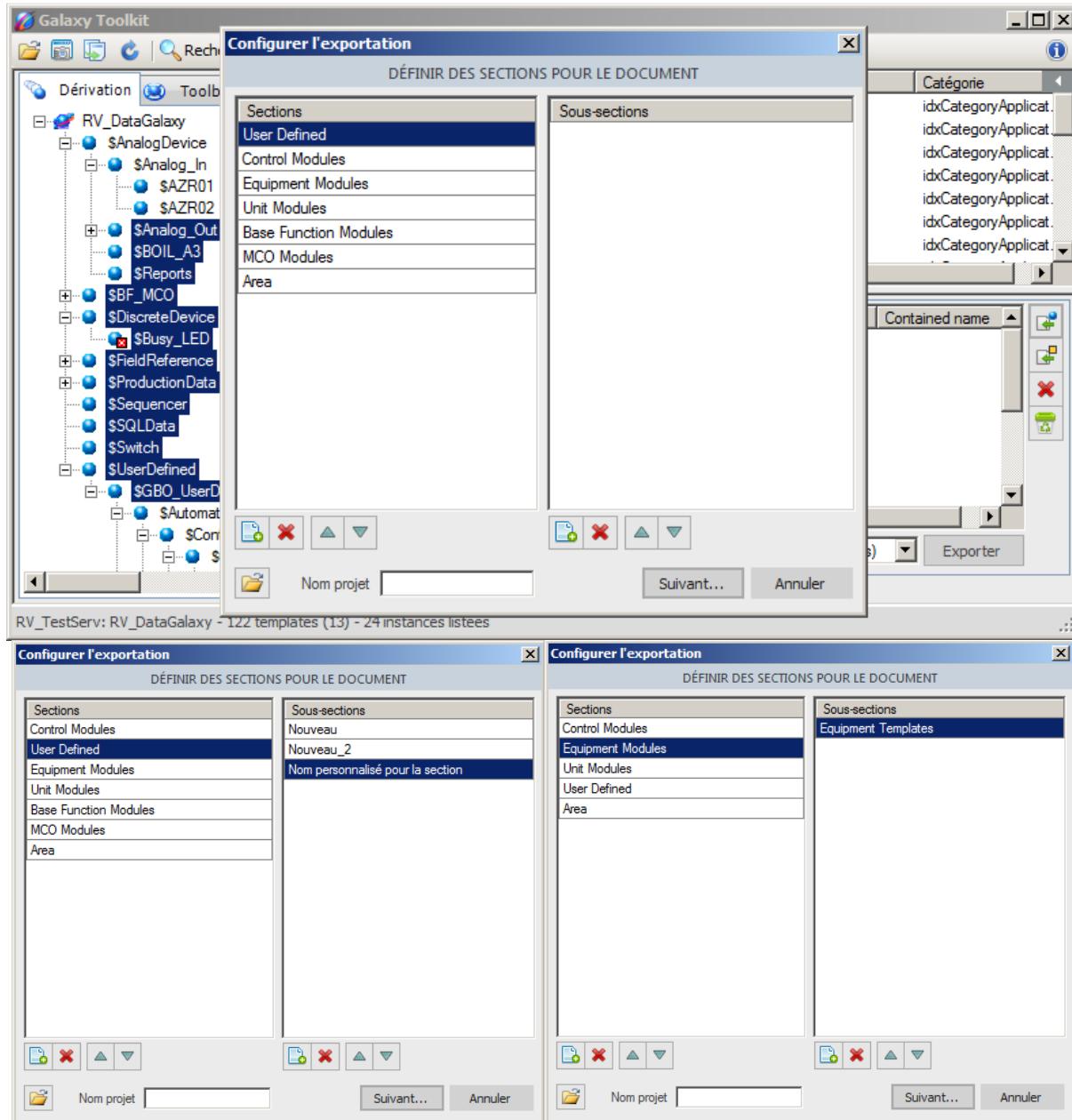
```
Table scrtble = document.AddTable(3, 2);
scrtble.Rows[0].Cells[0].Width = 110;
scrtble.Rows[0].Cells[0].FillColor = Color.LightGray;
scrtble.Rows[0].Cells[0].Paragraphs[0].Append("Name");
```

Le résultat de l'export est en tout point similaire à l'exemple fourni dans le cahier des charges.

Sections du document

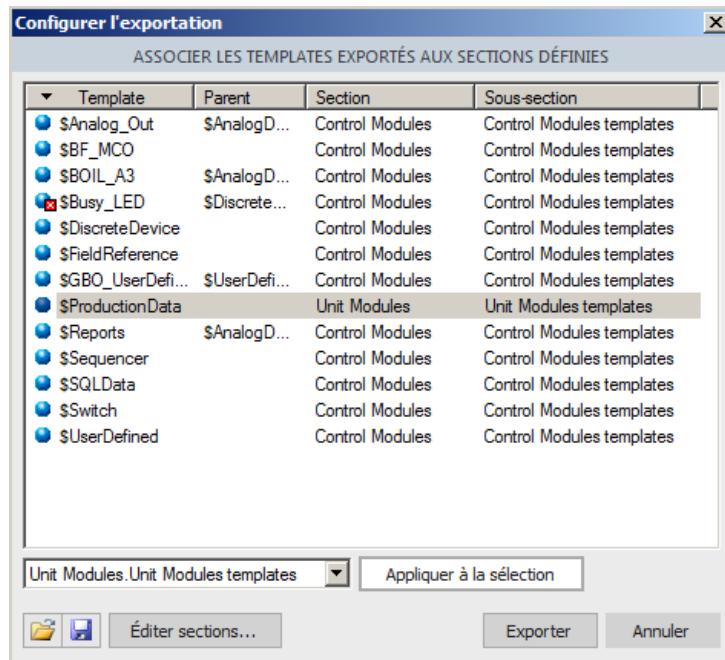
Une particularité de ce rapport est la présence de sections à l'intérieur du document. Les templates sont en effet répartis dans des sortes de chapitres. J'ai donc ajouté à mon application une fenêtre permettant de choisir les sections et d'y ranger les templates.

La première étape consiste à définir un ensemble de sections et leurs sous-sections. Il est donc possible d'ajouter et de supprimer des sections, et aussi de modifier leur nom et leur ordre.



Lorsque le choix des sections est terminé, le bouton « Suivant » permet de passer à la seconde étape : l'association de chaque templates à une section. Un champ permet de définir un nom de projet, qui apparaîtra dans l'en-tête du document généré. Le bouton d'ouverture de document JSON, situé en bas à gauche, est expliqué à la page suivante.

La seconde phase consiste à associer chaque template à une section et à une sous-section. Elles sont visibles dans une liste de sélection, sous la forme *Section.Sous-section*. Celle-ci permet de modifier le rangement des templates sélectionnés.



Il est possible d'enregistrer les sections, les sous-sections et le classement des templates dans un fichier JSON. Le format JSON a été choisi pour sa simplicité d'édition (voir ci-dessous).

```
"nom_projet" : {
    "User Defined" : {
        "User Defined templates" : [
            "$Test",
            "$Test_ext",
            "$UserDefined",
            "$UserDefined_001"
        ]
    },
    "Control Modules" : {
        "Control Modules templates" : [
            "$AnalogDevice"
        ]
    },
    "Equipment Modules" : {
        "Equipment Modules templates" : [
            "$DiscreteDevice"
        ]
    }
}
```

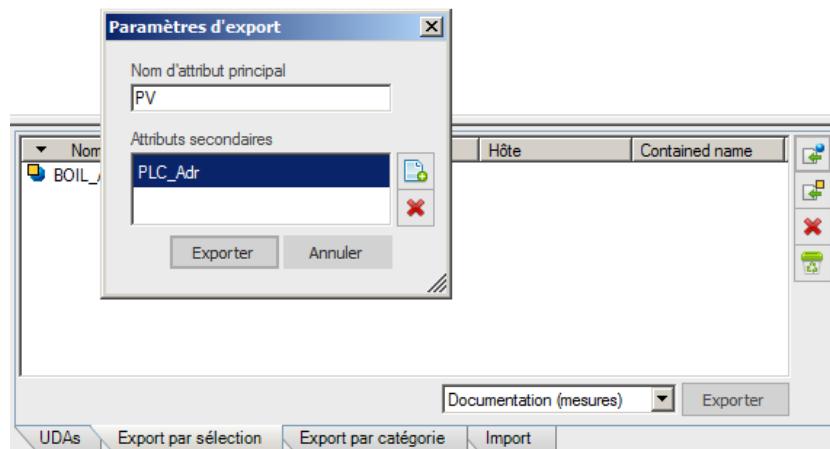
Le fichier JSON peut être chargé en utilisant le bouton d'ouverture, situé en bas à gauche. Il est également possible de le charger directement depuis la liste d'export, à condition qu'elle soit vide. La demande d'export agit alors comme une demande d'ouverture du fichier JSON.

5.2.11 - Rapports de documentation Excel

Rapport de mesures

Toutes les instances utilisées pour réaliser des mesures possèdent un attribut spécial portant toujours le même nom, généralement « PV ». Dès lors, pour obtenir les données des mesures, il suffit de vérifier la présence de cet attribut principal dans chaque instance.

Les instances possédant l'attribut principal de mesure sont aussi équipées d'une série de variantes découlant du même nom. Par exemple, la description est contenue dans « PV.ShortDesc » si l'attribut principal se nomme « PV ». Chez Technord, un attribut *UDA* est utilisé en supplément, pour l'adressage. J'ai donc mis au point une fenêtre permettant de préciser le nom de l'attribut principal, ainsi que le nom d'un ou plusieurs *UDAs* additionnels.



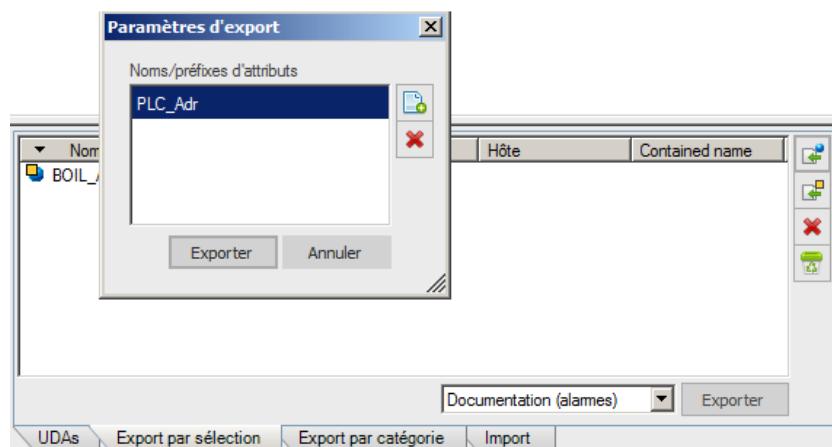
Après avoir récupéré l'ensemble des données, il ne reste plus qu'à les exporter dans un document Excel. Comme dans le cas des rapports de templates, un canevas de fichier est utilisé. Il permet d'utiliser une mise en page conçue à l'avance. Cette méthode se révèle beaucoup plus rapide à exécuter que la génération complète d'un document. Par ailleurs, elle permet d'avoir un canevas facilement éditabile. Elle m'a donc paru préférable à une génération totale du document.

Le résultat de l'export ressemble au document ci-dessous.

Concern: <i>List Measurement</i> Document:																	
Customer Reference:																	
Name (A2)	Area A ²	Name (Intouch)	Description	PLC_Adress	TopicName	EngUnits	MinEU	MaxEU	MinRaw	MaxRaw	RawType	Low Limit	High Limit	Force Storage Period	Value	Deadband	Package
PV1515.PV	RX59_D409	PV1515	D409 : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	10000	Linear	N/A	N/A	300000	0,1		
PV1525.PV	RX59_D410	PV1525	D410 : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	10000	Linear	N/A	N/A	300000	0,1		
PV1609.PV	RX59_D404	PV1609	D404 : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	10000	Linear	N/A	N/A	300000	0,1		
PV1907.PV	RX59_D405	PV1907	D405 : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	10000	Linear	N/A	N/A	300000	0,1		
PV2118.PV	RX59_D501	PV2118	D501 : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	10000	Linear	N/A	N/A	300000	0,1		
PV3207.PV	RX59_D601A	PV3207	D601A : Sortie ANA pression de sortie des gaz	W8042		%	0	100	0	4000	Linear	N/A	N/A	300000	0,1		

Rapport d'alarmes

L'obtention des données des alarmes est plus compliquée que celle des mesures. En effet, il existe de nombreuses catégories d'alarmes : *UDAs*, extensions d'*UDAs* et extensions de *field attributes*. Une série de « faux » attributs, présent dans la liste *Attributes* de chaque objet, contiennent des valeurs booléennes, indiquant la présence ou l'absence d'un type d'alarme. Si une alarme est présente, sa méthode d'obtention dépend de son type. Les alarmes d'extensions portent toujours les mêmes noms, quelle que soit leur instance. Elles sont donc faciles à récupérer. Les alarmes d'*UDAs*, en revanche, nécessitent la désrialisation d'une liste pour être identifiées.

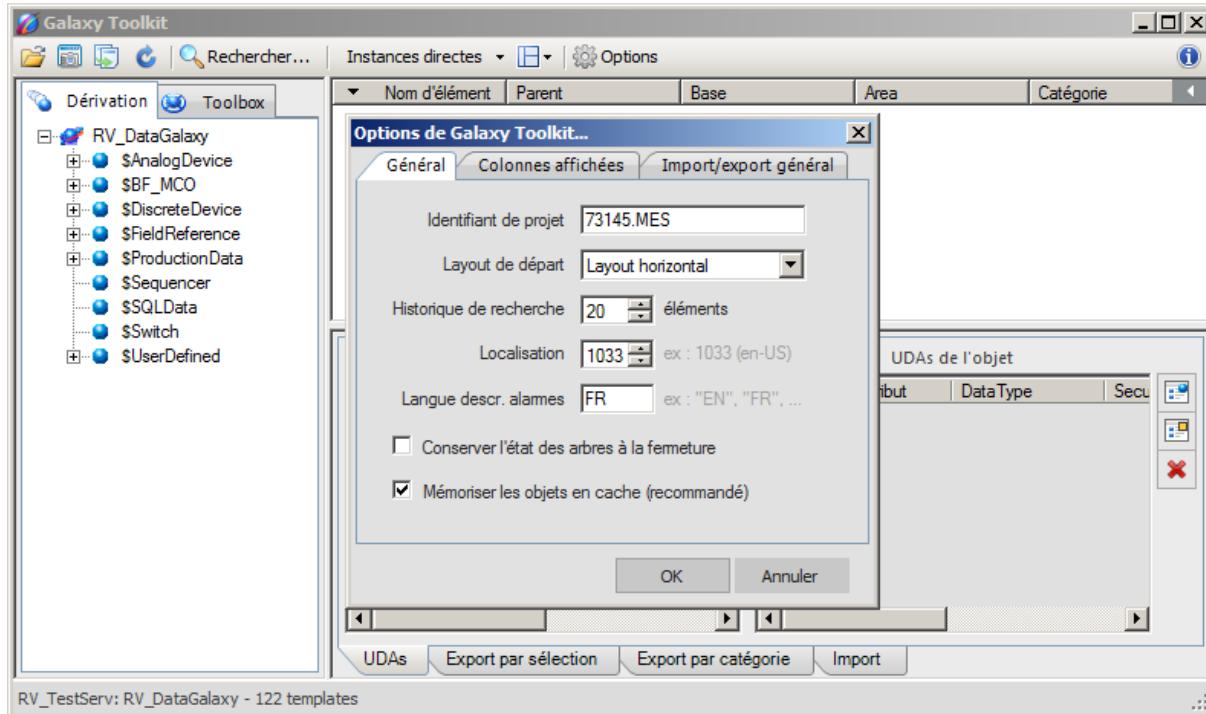


Au niveau de la création du document, l'export d'alarmes est similaire à celui de mesures.

A	B	C	D	E	F	G	H	I
 TECHNORD Solutions flexibles. Expertise solide.			Concern: List Alarm			Document:		
						Project:	Site:	Place:
			Customer Reference:					
Tag A2	PLC_Ad	Topic Name	Area A2	Active State	Priority	Description	Package	Remark
D604_AL_01.Alarm_01	B1452		RX59_D604	On	500	XV3431D : Disc xv - purge filtre B601AC		
D604_AL_01.Alarm_02	B1452		RX59_D604	On	500	XV3432D : Disc xv - VP sur filtre B601AC		
D604_AL_01.Alarm_03	B1452		RX59_D604	On	500	XV3433D : Disc xv - arrivée ACP		
D604_AL_01.Alarm_04	B1452		RX59_D604	On	500	XV3434D : Disc xv - VP sur filtre B601AB		
D604_AL_01.Alarm_05	B1452		RX59_D604	On	500	XV3435D : Disc xv - vent filtre B601AE		
D604_AL_01.Alarm_06	B1452		RX59_D604	On	500	XV3436D : Disc xv - purge filtre B601AB		

5.2.12 - Fenêtre d'options et configuration

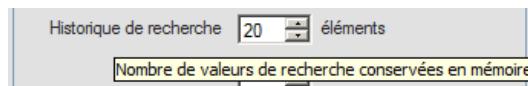
Galaxy Toolkit dispose d'un certain nombre d'options permettant d'assurer sa flexibilité. La configuration est lue et sauvegardée dans un fichier *.ini, sous la forme : OPTION=VALEUR.



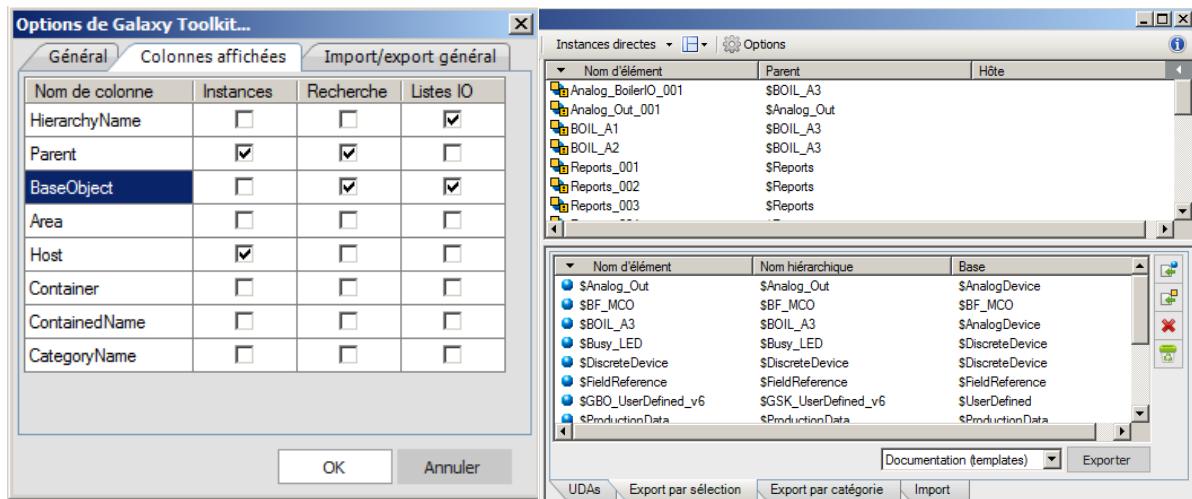
Les options générales permettent de choisir :

- des valeurs qui seront utilisées pour la génération de rapports (le nom identifiant pour la liste de templates, et le suffixe de langue pour les attributs d'alarmes) ;
- une option de localisation, utilisée pour les imports de descriptions d'instances (*MxInternationalizedString*) ;
- des options d'ergonomie (disposition de départ, longueur de l'historique de recherche, *snapshot* automatique lors de l'ouverture et de la fermeture).

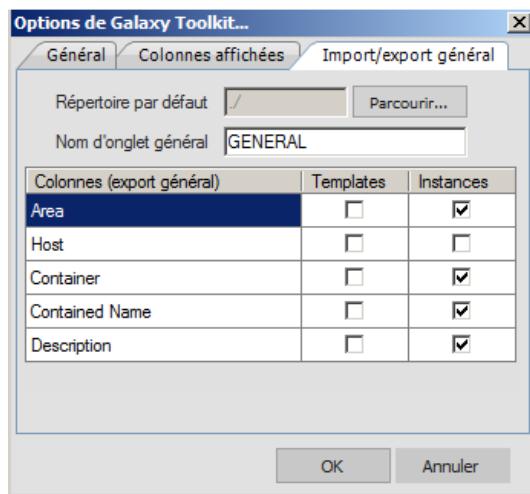
Des indications complémentaires sont fournies au survol de la souris.



Le second panneau d'options permet de choisir quelles colonnes afficher dans les listes de l'application (liste d'instances, résultats de recherche, listes d'import et d'export).



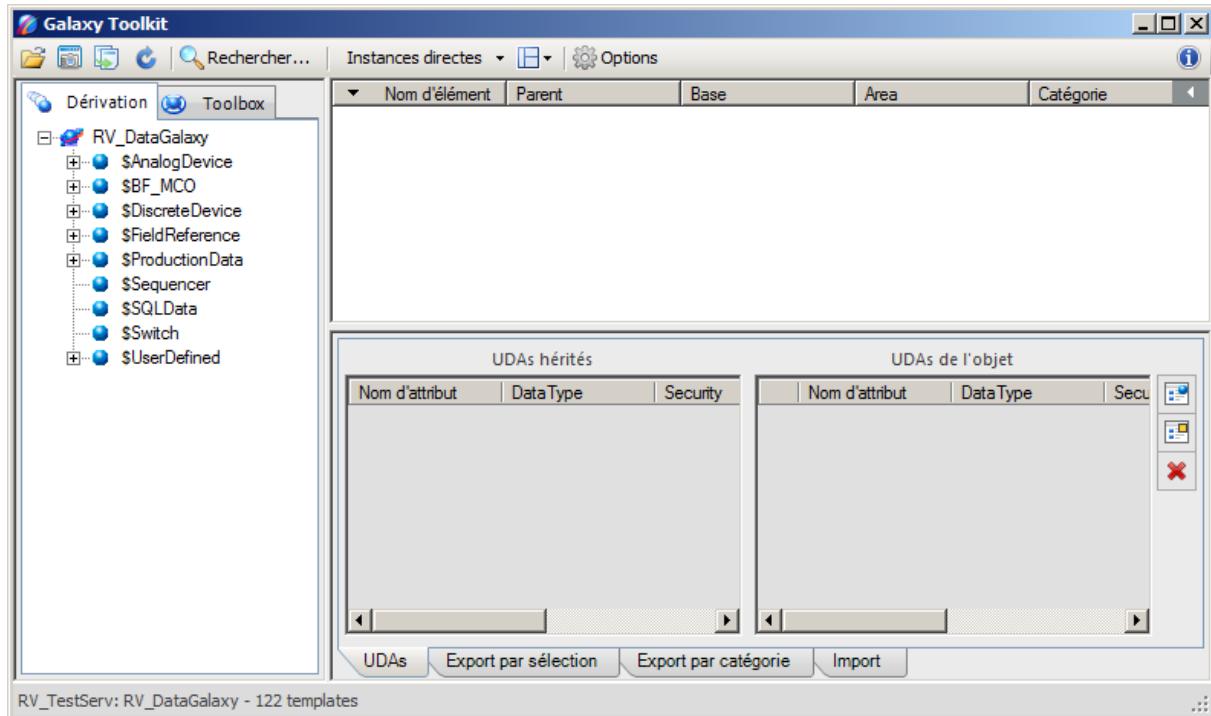
Enfin, le dernier panneau définit des paramètres pour l'import et l'export général, tels que le nom de l'onglet principal du fichier Excel et le choix des colonnes ajoutées au document.



5.3 – Design et ergonomie

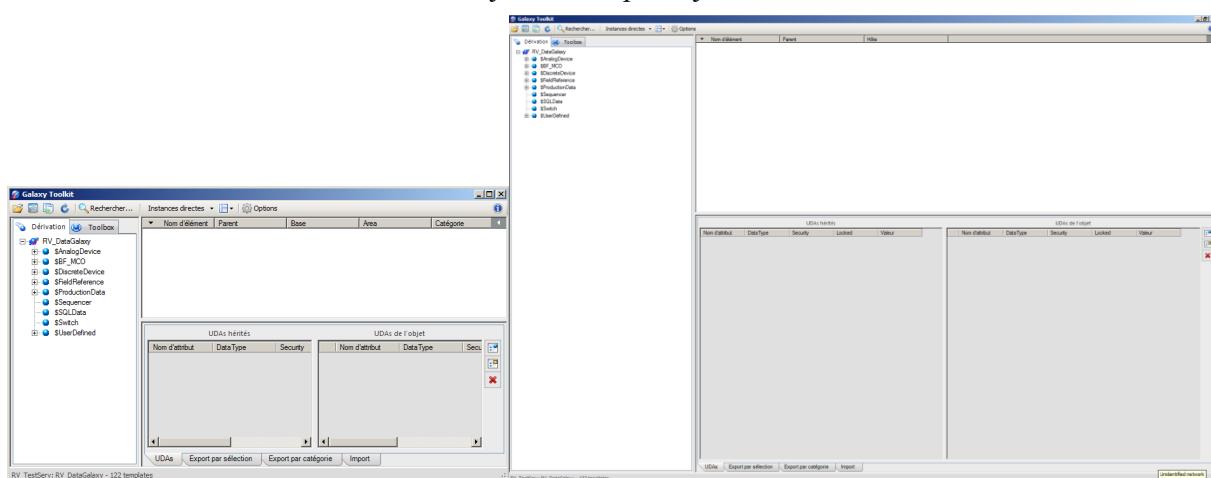
5.3.1 - Interface graphique

L'interface graphique de mon application respecte les conventions : une barre de menu au-dessus, les arbres de données à gauche et un espace de travail à droite. Ces conventions offrent au projet une utilisation intuitive. L'aspect des onglets et des boutons a été retravaillé, afin de leur procurer une apparence moderne (et ce, même sur les vieilles versions de Windows).



La première caractéristique de l'interface graphique est d'être « élastique » (*responsive*). En effet, le contenu des fenêtres s'adapte lorsqu'elles sont redimensionnées. L'application est donc utilisable sur des écrans de toutes les tailles et peut être agrandie ou rétrécie selon les préférences de l'utilisateur. Les séparateurs entre les zones principales peuvent aussi être déplacés.

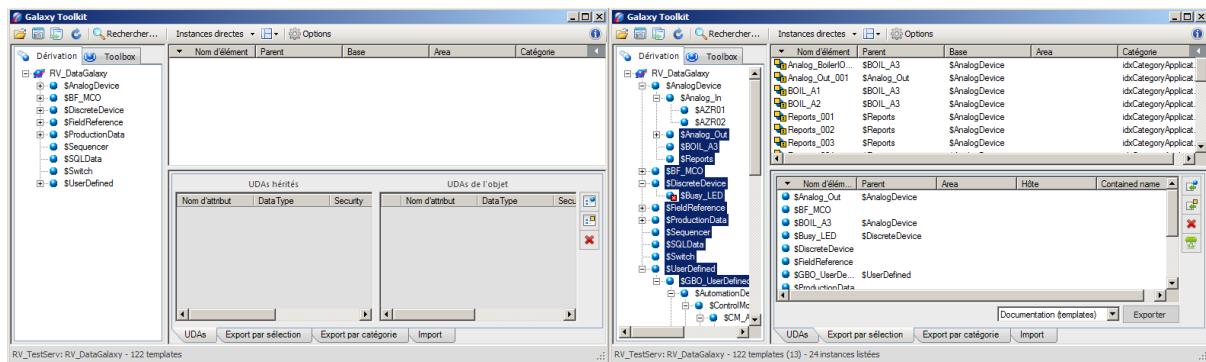
Le contenu de la fenêtre s'adapte en fonction de sa taille.



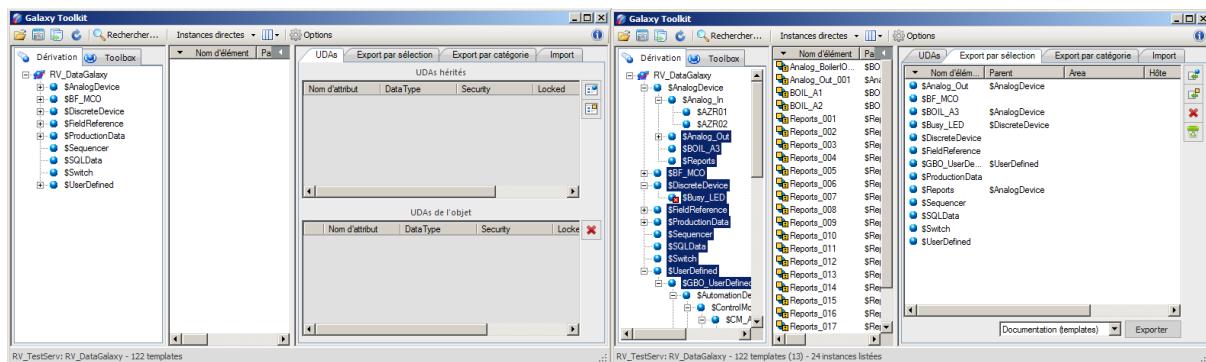
Un bouton de la barre de menu permet de changer la disposition des blocs situés dans l'espace de travail (liste d'*instances* et zone contextuelle). Les deux sections peuvent être disposées l'une au-dessus de l'autre, ou l'une à côté de l'autre. Un raccourci clavier (*Ctrl + L*) permet également de changer rapidement de disposition.



La disposition des blocs l'un au-dessus de l'autre permet à l'utilisateur de voir de nombreuses colonnes d'informations à propos de chaque objet. Par contre, elle limite le nombre d'objets visibles simultanément.



La disposition des blocs l'un à côté de l'autre permet à l'utilisateur de voir un grand nombre d'objets dans les listes de données. Cela peut être pratique pour réaliser des sélections d'éléments. En revanche, le nombre de colonnes d'informations visibles est limité.



Chacun des deux types de dispositions présente des avantages. L'utilisateur est libre de choisir l'agencement de départ dans les options du logiciel.

5.3.2 - Amélioration de l'expérience utilisateur

Aux yeux d'un utilisateur, la manière d'utiliser une application importe autant que les possibilités de celle-ci. Il ne suffit pas de réaliser une série de fonctionnalités sans respecter aucun standard de mise en forme, ni d'organisation. L'utilisateur doit pouvoir effectuer des actions en se basant sur des principes auxquels il est habitué. Pour répondre à cette problématique, mon application fait appel à un certain nombre d'astuces, listées ci-dessous.

- **Mémorisation des dernières informations de connexion :**

L'utilisateur n'est pas obligé de saisir systématiquement les mêmes informations à chaque connexion : elles sont mémorisées pour la fois suivante.

- **Affichage d'informations au survol des boutons et des champs :**

Chaque bouton et chaque contrôle de formulaire de l'application dispose d'informations affichées au survol de la souris, couramment appelées « infos bulles ». Ces informations permettent à l'utilisateur d'être certain de l'action réalisée par chaque élément.

- **Affichage « grisé » des boutons et contrôles inactifs :**

Lorsqu'un bouton ou contrôle ne peut être utilisé, son apparence s'adapte pour le signaler. Ce principe augmente l'affordance¹ des objets concernés.

- **Disposition de la fenêtre principale (*layout*) adaptable :**

La fenêtre principale peut être disposée de deux manières, comme expliqué au point 5.3.1. L'utilisateur peut configurer la disposition de départ et la changer à tout moment. Par ailleurs, les séparateurs (*splitters*) entre les différentes zones sont déplaçables.

- **Raccourcis pour le clavier et touches fonctionnelles :**

Les raccourcis pour le clavier offrent un gain de temps intéressant, lors de l'utilisation régulière d'une application. Mon projet permet de s'en servir pour ouvrir une fenêtre de recherche (*Ctrl + F*) et pour modifier la disposition de l'application (*Ctrl + L*). D'autres touches remplissent des usages fonctionnels : la suppression d'un élément dans une liste (*Delete*), la confirmation rapide d'une recherche (*Enter*), ...

- **Boîtes de dialogue de confirmation, lors d'une suppression :**

En cas de fausse manœuvre, l'utilisateur a la possibilité d'annuler la suppression.

- **Contenu des fenêtres élastique (*responsive*) :**

L'utilisateur peut modifier la taille des fenêtres selon ses désirs. Le contenu s'y adapte.

- **Dans une liste multi-colonnes, sélection des items sur toute la largeur :**

Plus un élément est large, plus il est facile de cliquer dessus (loi de *Fitts*).

- **Historique de recherche mémorisé par ordre de saisie :**

Chaque nouvelle recherche s'ajoute en haut de la liste d'historique. Si une ancienne valeur est réutilisée, elle se replace également au début de la liste.

- **Possibilité de double-clic et de *drag and drop*, en plus des boutons :**

Les utilisateurs sont habitués à l'utilisation de « raccourcis » à l'aide de la souris. Le respect de ces conventions rend l'utilisation d'une application plus intuitive.

¹ Affordance : la capacité d'un objet à suggérer sa propre utilité.

5.3.3 - Tests d'ergonomie

Une fois les fonctionnalités de mon projet terminées, la réalisation de tests d'ergonomie ont permis de m'assurer de la qualité de son utilisation et de corriger certains aspects selon les résultats. Les utilisateurs ont reçu les énoncés de trois scénarios à réaliser. Ils ont ensuite complété les actions demandées, sans intervention de ma part. Il est important de préciser que chacun des utilisateurs testait mon application pour la première fois.

Premier scénario

« Trouvez l'instance nommée *COM2_CPT1_ENTREFER2* et réalisez un export général importable (*xlsx*) de cette instance. Importez ensuite le fichier que vous venez d'exporter. »

Résultats de l'utilisateur numéro 1 – premier scénario

Durée du test : 2 minutes, 45 secondes

Satisfaction de l'utilisateur : 4/5

L'utilisateur a utilisé la recherche pour trouver l'instance. Il n'a pas remarqué la présence du bouton « Exporter la sélection ». Il a procédé de la même manière que dans ArchestrA IDE : en double-cliquant sur l'objet pour naviguer vers lui (électionner son template parent dans l'arbre courant et sélectionner l'instance dans la liste). Il a été perturbé par le fait que la liste ne défilait pas automatiquement vers l'élément sélectionné. La suite s'est déroulée sans encombre.

Ce test m'a permis de comprendre la nécessité d'un défilement automatique (*scroll*) de la liste des instances lors d'une navigation depuis la recherche. Cela permet à l'élément sélectionné d'être automatiquement visible. J'ai donc ajouté cette fonctionnalité.

Résultats de l'utilisateur numéro 2 – premier scénario

Durée du test : 2 minutes, 17 secondes

Satisfaction de l'utilisateur : 5/5

L'utilisateur a utilisé la recherche pour trouver l'instance. Il a utilisé le bouton d'export, plaçant directement l'instance dans la liste d'export. La suite s'est déroulée sans encombre.

Ce test n'a pas posé de difficultés.

Second scénario

« Exportez le rapport de documentation au sujet des templates (*docx*) pour tous les éléments de la galaxie. Conservez les sections de document par défaut et placez le template *\$UserDefined* dans une autre section que la sienne. »

Résultats de l'utilisateur numéro 1 - second scénario

Durée du test : 3 minutes, 57 secondes

Satisfaction de l'utilisateur : 4/5

L'utilisateur a compris rapidement que la méthode la plus simple pour exporter toute une galaxie était l'export par catégories, sans activer aucun filtre. La fenêtre de classement des templates dans les sections du document s'est ouverte. L'utilisateur a été perturbé par la nomenclature utilisée. Les sections du document portaient le nom de « catégories », ce qui portait à confusion avec la propriété de catégorisation des objets ArchestrA. De plus, le bouton de validation des sections, portant le nom « OK », ne lui permettait pas de deviner l'existence d'une deuxième étape (le placement des templates dans chaque section). Il est toutefois parvenu à terminer le test, après quelques recherches.

Ce test a mis en évidence l'ambiguïté du terme « catégories », et la contre-affordance¹ du bouton « OK ». J'ai donc décidé de renommer les « catégories » du document en « sections ». Quant au bouton, il a reçu la nouvelle appellation « Suivant... », ce qui a permis de mettre en valeur la présence d'une seconde étape. Ces modifications ont été apportées avant la réalisation du test par le second utilisateur, ce qui explique que ce dernier n'ait pas été perturbé.

Résultats de l'utilisateur numéro 2 - second scénario

Durée du test : 3 minutes, 24 secondes

Satisfaction de l'utilisateur : 5/5

L'utilisateur a mis plus de temps pour comprendre que la méthode adéquate consistait à utiliser l'export par catégories. Mais une fois la fenêtre de classement des templates ouverte, il n'a eu aucun mal à comprendre son fonctionnement. La suite du test s'est déroulée sans encombre.

Ce test n'a pas posé de difficulté. Il prouve que les changements de nomenclature apportés se sont révélés efficaces.

¹ Affordance : la capacité d'un objet à suggérer sa propre utilité. Par exemple, colorer un lien hypertexte en bleu et le souligner permet d'augmenter son affordance.

Contre-affordance : le fait qu'un objet suggère une autre utilité que la sienne. Par exemple, colorer un lien hypertexte en noir, sans soulignement, est contre-affordant : il ne ressemble plus à un lien.

Troisième scénario

« Chargez la sauvegarde de l'état des arbres (*snapshot*) nommée 'test3.snap' (située sur le bureau Windows) et attendez la fin du chargement des instances. Parmi les instances listées, trouvez les instances *AE0004* et *AE11104* (sans utiliser la recherche). Exportez les mesures et les alarmes de ces deux instances. »

Résultats de l'utilisateur numéro 1 – troisième scénario

Durée du test : 1 minute, 48 secondes

Satisfaction de l'utilisateur : 5/5

L'utilisateur a trouvé directement le bouton de chargement d'un *snapshot*. Il n'a pas eu besoin d'utiliser de filtres pour trouver les instances demandées. La liste était en effet relativement courte. Les exports de mesures et d'alarmes se sont déroulés sans encombre.

Ce test n'a posé aucune difficulté.

Résultats de l'utilisateur numéro 2 – troisième scénario

Durée du test : 2 minutes, 6 secondes

Satisfaction de l'utilisateur : 5/5

L'utilisateur a pris le temps d'observer les différents boutons du menu, afin de lire les informations affichées au survol de la souris. Il a ensuite trouvé le bouton de chargement d'un *snapshot*. Il n'a pas utilisé de filtres pour trouver les instances demandées. Les exports de mesures et d'alarmes se sont déroulés sans encombre.

Ce test n'a posé aucune difficulté.

Conclusions

La réalisation des tests d'ergonomie a permis de souligner certains défauts de l'application, que je n'aurais pas constatés seul. Ceux-ci ont été corrigés en conséquence. Il s'agissait essentiellement de problèmes de contre-affordance, liés à une nomenclature ambiguë. L'absence de défilement automatique d'une liste (en cas de navigation depuis la recherche) était également problématique.

5.4 – Problèmes rencontrés

Installation d'ArchestrA et de GRAccess

L'installation d'ArchestrA a posé quelques difficultés. Un certain nombre d'exigences doivent être satisfaites pour la mener à bien.

Tout d'abord, l'installation d'ArchestrA nécessite la présence d'une version précise de Microsoft SQL Server. Si une autre version est déjà sur le système (par exemple, celle installée avec Visual Studio), l'installateur ne détecte pas correctement la présence de la version qui lui correspond. Il était donc nécessaire d'installer ArchestrA avant Visual Studio. Par ailleurs, un *service pack* précis doit également être présent pour SQL Server. S'il s'agit d'une version plus récente que celle demandée, elle n'est pas détectée non plus. Il en va de même s'il s'agit du même *service pack* en mode 32 bits au lieu de 64 bits.

Une fois la bonne version de SQL Server en place, il devient aisément d'installer ArchestrA et ensuite GRAccess. Cependant, si le système ne dispose pas de droits d'administrateur, le lancement de GRAccess ne peut se faire. Ma première tentative d'utilisation de GRAccess, sur le système hôte de mon ordinateur, s'est donc soldée par un échec. Le système de sessions synchronisées de Technord ne permettait pas de créer un nouvel utilisateur disposant de ces droits. La solution a été de recommencer la procédure d'installation sur une machine virtuelle.

Lenteur de GRAccess

Lors de mes premiers essais avec GRAccess, je n'ai pu m'empêcher de constater la lenteur extrême de l'API pour récupérer des éléments. Plusieurs dizaines minutes étaient parfois nécessaires pour récupérer une longue liste d'instances.

Après avoir analysé chaque opération, j'ai réalisé qu'une partie du problème provenait de la récupération de la liste des erreurs ArchestrA associées aux objets. Cette liste avait pour seule utilité de m'indiquer si un objet était erroné ou non. En effectuant de nouvelles recherches, je me suis aperçu que les objets disposaient d'une propriété nommée *ValidationStatus*. Celle-ci permettait de connaître cette information, sans devoir récupérer la liste des erreurs. Le simple fait d'utiliser cette propriété et de ne plus vérifier la liste d'erreurs m'a permis de rendre la récupération d'objets six fois plus rapide.

Cependant, j'ai constaté que l'affichage de longues listes d'éléments prenait également un temps anormalement long. Lors de chaque mise à jour des listes, l'ensemble des éléments y étaient insérés un à un, au moyen de la méthode *Add* prévue à cet effet. En remplaçant ce système par un ajout groupé, via la méthode *AddRange*, l'affichage des listes est devenu pratiquement instantané, aussi longues soient-elles.

En dépit de toutes ces améliorations, la récupération d'objets via GRAccess restait un peu lente. Pour pallier à ce problème, j'ai mis au point un système de « cache », permettant de stocker en mémoire les objets déjà chargés. La première récupération d'un élément a vu sa durée inchangée, mais les récupérations ultérieures sont devenues presque instantanées.

Drag and drop d'objets – conflit d'événements

La liste des *instances* de mon application permet de réaliser une série d'opérations avec la souris. Il est possible de sélectionner des éléments et de double-cliquer pour les utiliser automatiquement dans la zone contextuelle. Souhaitant ajouter une possibilité de *drag and drop* (glisser et déplacer), je me suis aperçu que cela provoquait un conflit entre les événements de la souris. En effet, après l'appel de la méthode *DoDrag* au début d'un clic de souris (*MouseDown*), les événements de sélection et de double-clic n'étaient plus pris en compte.

Pour résoudre ce problème, j'ai réalisé le déclenchement du *drag and drop* en deux temps. Lors d'une pression sur la souris (*MouseDown*), j'ai cessé de démarrer directement la méthode *DoDrag*. À la place, j'ai simplement utilisé le gestionnaire de l'événement pour activer une variable booléenne. En cas de relâchement du clic (*MouseUp*), cette variable était remise à son état initial. Il devenait donc possible, pour les autres événements, de savoir si la souris était pressée ou non.

La deuxième étape consistait à utiliser un nouvel événement permettant de savoir si la souris quittait la zone de la liste, autrement dit, cessait de la survoler (*MouseLeave*). J'ai chargé cet événement d'appeler la méthode *DoDrag*, à condition que la souris soit pressée. Le problème de conflit a ainsi été résolu, permettant de cumuler à la fois la sélection, le double-clic et le *drag and drop*.

Chapitre 6

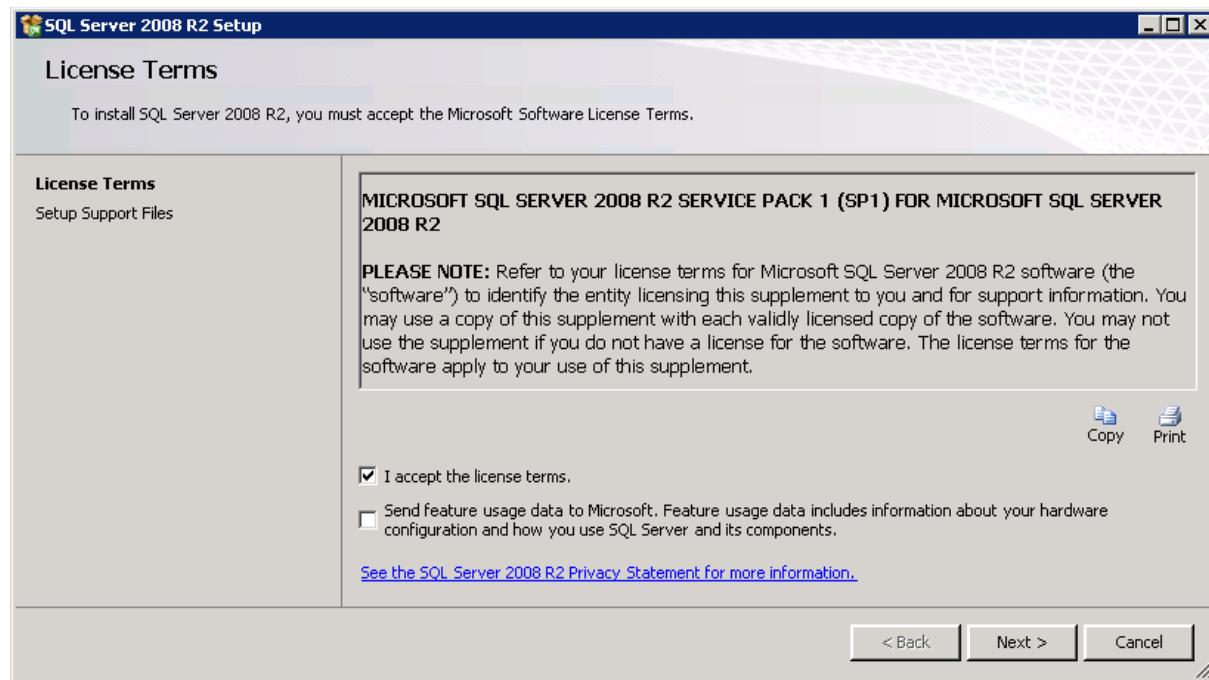
Procédure d'installation

6.1 – Prérequis

Pour utiliser mon application, ainsi que l'API GRAccess d'ArchestrA, il est nécessaire de répondre à un certain nombre de critères :

- Utiliser un système d'exploitation compatible avec ArchestrA 2012 R2 :
 - *Microsoft Windows Server 2003 SP2 Standard/Enterprise* ;
 - *Microsoft Windows Server 2008 SP2 Standard/Enterprise* (32/64-bit) ;
 - *Microsoft Windows Server 2008 R2 SP1 Standard/Enterprise* (64-bit).
- Posséder les droits d'administrateur sur la machine.
- Installer *ArchestrA System Platform 2012 R2*, s'il n'est pas déjà présent sur la machine.

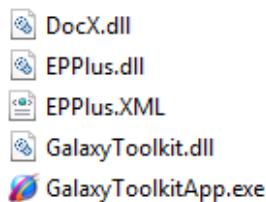
Pour pouvoir installer *ArchestrA System Platform*, il est nécessaire de disposer d'une version de *Microsoft SQL Server* supportée par le logiciel. Il est recommandé de ne pas cumuler plusieurs versions différentes car ArchestrA peut alors avoir du mal à les identifier. Sur un système 32 bits, la seule version supportée est *SQL Server 2008 SP3*. Sur un système 64 bits, il est recommandé d'utiliser *SQL Server 2008 R2 SP1 x64*. Il est indispensable d'installer le *Service-Pack* approprié.



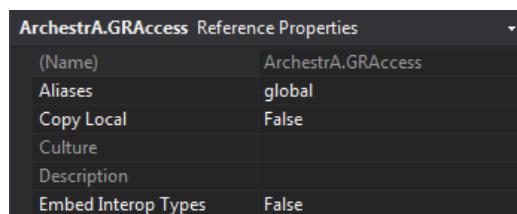
Une fois SQL Server installé, il est aisément d'installer *ArchestrA System Platform 2012 R2*. Il n'est pas nécessaire de sélectionner la fonctionnalité *In Touch*, qui n'est pas utilisée pour le projet.

6.2 – Installation de l'application

En dehors des prérequis, la seule installation à réaliser est celle de *GRAccess 2012 R2*. L'utilitaire ne nécessite aucune précaution particulière lors de sa mise en place.



Pour utiliser mon application, il suffit de copier l'ensemble de ses fichiers dans un même dossier, puis de démarrer le fichier exécutable *GalaxyToolkitApp.exe*. Les fichiers à copier sont au nombre de cinq (voir l'image ci-contre).

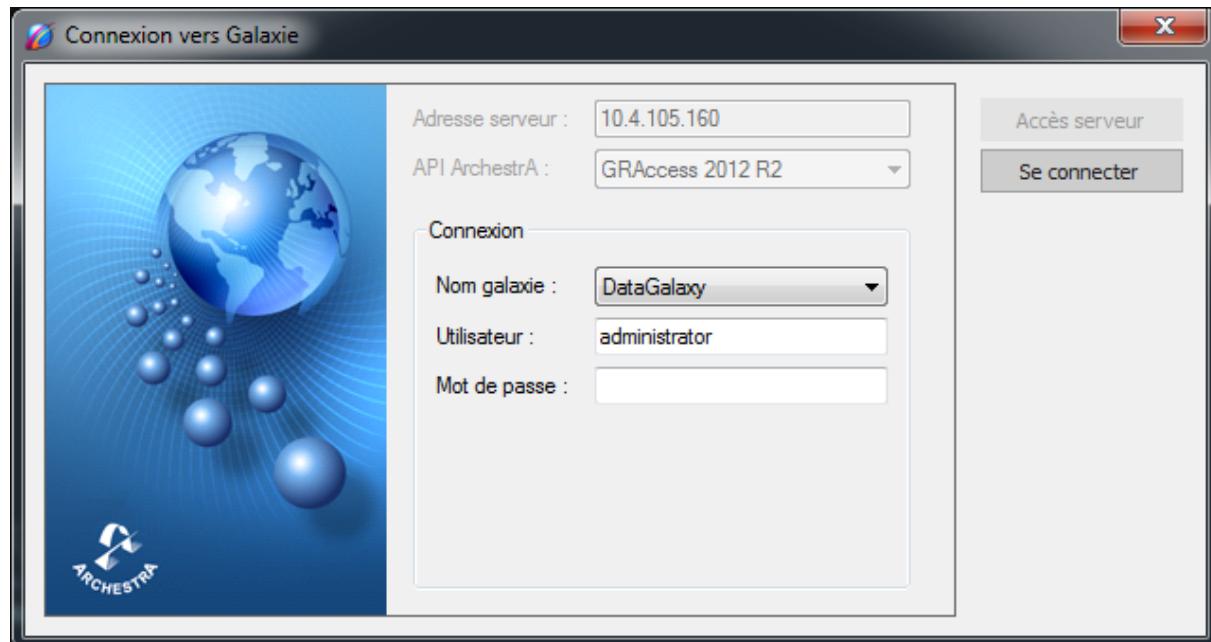


Pour recompiler l'application, il est important de s'assurer que la référence vers l'API GRAccess soit configurée comme suit : les attributs « Copy Local » et « Embed Interop Types » doivent valoir *false*.

6.3 – Lancement de l'application

Lors du démarrage de l'application, deux étapes sont nécessaires pour établir une connexion avec ArchestrA. La première consiste à ouvrir un accès avec GRAccess, en précisant l'adresse IP du serveur ArchestrA. Bien qu'elle ne contienne pour l'instant qu'un élément, une liste de sélection a été prévue pour permettre le choix de la version d'ArchestrA concernée. Cette liste trouvera son utilité lorsque l'application supportera plusieurs versions.

La seconde étape consiste à choisir vers quelle « galaxie » (projet ArchestrA) se connecter, et à fournir un nom d'utilisateur et un mot de passe. Sitôt cette étape complétée, la fenêtre principale de l'application s'ouvrira.



Chapitre 7

Avenir du projet

Le projet réalisé sera utilisé par le département MES-IT de Technord et partagé avec d'autres implantations de la société, notamment celle de Tournai. De nouvelles fonctionnalités sont susceptibles d'y être ajoutées, telles que celles reprises dans la liste ci-dessous :

- Support d'autres versions d'ArchestrA

L'application actuelle prend en charge la version 2012 R2 d'ArchestrA (et par extension, de GAccess). L'organisation du code permet toutefois d'ajouter le support d'autres versions, par exemple ArchestrA 2014. Une énumération est prévue pour le choix de version dans l'interface graphique, à laquelle il suffit d'ajouter les valeurs souhaitées. Côté données, le patron de conception *factory* a été prévu pour gérer les versions. Il suffit donc d'ajouter de nouvelles classes dérivant de la même classe abstraite conceptuelle. Quant au contenu de ces classes spécifiques, il est minimaliste : les traitements de données indépendants de la version sont gérés par la classe abstraite, et les données sont stockées dans des conteneurs indépendants. Le code à réécrire pour une nouvelle classe est donc relativement court.

- Conversion de données d'une version à l'autre

Si les données exportées par l'application peuvent ensuite être importées vers la même version d'ArchestrA, rien n'empêche qu'elles puissent aussi l'être vers une autre version. L'ajout d'options d'export pourrait permettre d'opter pour une réorganisation des données, en vue d'une conversion.

- Nouveaux types de documentation

La gestion des différents exports est centralisée. Ceci permet d'ajouter de nouveaux types avec très peu de modifications du code. Il suffirait ensuite d'écrire une fonction récupérant les bons éléments et une autre se chargeant de créer le fichier d'export, quelle que soit sa nature.

- Nouvelles options et fonctionnalités

De nouveaux paramètres pourraient être ajoutés pour augmenter la flexibilité des fonctionnalités actuelles. Il est également envisageable de concevoir d'autres éléments, que ce soit pour la gestion des données, pour leur affichage ou pour améliorer le confort d'utilisation.

Pour faciliter la maintenance du code, il m'a semblé judicieux de faciliter sa compréhension. Pour ce faire, le code a été fortement documenté. Des en-têtes détaillés ont été spécifiés pour chaque fonction, décrivant son rôle et chaque paramètre. Ces informations, respectant la norme XML, peuvent aussi être consultées au survol de souris de chaque appel de fonction ou de chaque paramètre. En dehors des en-têtes, de nombreux commentaires se chargent d'expliquer les principes de chaque portion de code.

L'utilisation du paradigme orienté objet a également permis d'offrir une structure du code facile à comprendre. Il permet aussi l'utilisation des héritages, afin de créer de nouvelles classes basées sur les classes existantes (par exemple, si de nouveaux types de données devaient être pris en charge). Cette découpe du code permet enfin la réutilisabilité de certaines classes, telles que les arbres de données, pour d'autres projets.

Chapitre 8

Conclusion

À l'issue de mes quatorze semaines de stage, le bilan est très positif. Les différents objectifs du projet ont été atteints et les responsables du stage sont très satisfaits du résultat. Ce stage a également été une expérience positive pour moi. Le fait que mon projet soit destiné à être utilisé de manière régulière rendait sa réalisation plus motivante. L'apprentissage de nouvelles techniques au cours de sa réalisation l'a également rendu plus intéressant.

Le but principal du stage était de mettre au point une application permettant de gérer, importer et exporter les données du système de supervision ArchestrA. Non seulement les objectifs principaux et secondaires ont été atteints, mais également un ensemble de tâches supplémentaires : tests et optimisation des performances, mécanisme de file d'attente, étude d'ergonomie, ainsi qu'une série d'autres fonctionnalités et options. Ces éléments additionnels permettent d'améliorer l'utilisabilité, l'efficacité, les performances et la flexibilité de l'application.

L'immersion en entreprise m'a permis d'utiliser, dans un projet concret, les connaissances apprises au cours des trois dernières années. D'une part, j'ai eu l'opportunité d'élargir mes compétences techniques, que ce soit en termes de techniques de programmation, d'analyse, de logiciels ou encore de méthodes de tests et de recherche. D'autre part, j'ai pu me familiariser avec le milieu professionnel et acquérir des compétences d'organisation et de communication pour m'y intégrer efficacement. J'ai mis au point une méthode de travail organisée de manière claire et structurée tout au long du projet, me permettant à la fois d'optimiser mon temps de travail et de prévoir de manière réaliste la durée de chaque étape. Cette technique de gestion de projets me servira autant pour mes projets personnels que professionnels. Enfin, l'évolution du cahier des charges et la modification de certains objectifs au cours du stage ont permis de développer ma capacité d'adaptation. Outre l'apprentissage, le stage m'a permis de consolider mes orientations professionnelles.

Si le projet réalisé répond aux besoins actuels de Technord, il reste toutefois possible d'y ajouter de nouvelles fonctionnalités. Les décisions prises au sujet de l'organisation du projet assurent son évolutivité et sa facilité de maintenance. Le paradigme orienté objet permet l'ajout de nouvelles classes dérivées des classes existantes, pour de nouveaux types d'objets. Le patron de conception *factory* utilisé pour l'obtention des données depuis ArchestrA permet la prise en charge de versions supplémentaires. La séparation de la gestion de données et de l'aspect visuel permet d'altérer l'un sans nécessiter de modifier l'autre. Le projet a été pensé pour évoluer et sera certainement encore amélioré dans les années à venir.

Bibliographie

Livres

- BERSINI H., *Programmation orientée objet*, Eyrolles, 2013, 6^e éd., 650 pages.
- DE KORT W., *Exam Ref 70-483 – Programming in C#*, O'Reilly, 2013, 379 pages.
- COLLECTIF ENI, *Design patterns – 14 patrons de conception détaillés*, Eni, 2010, 12 pages.
- GABAY J. & GABAY D., *UML2 – Analyse et conception*, Dunod, 2008, 240 pages.
- BOUCHER A., *Ergonomie web*, Eyrolles, 2009, 2^e éd., 456 pages.

Notes de cours

- LÉONARD A., *Algorithmique*, HEPL, 2012, 90 pages.
- VILVENS C., *Java : Programmation de base*, HEPL, 2013, 541 pages.

Documentation non publique, format PDF

- WONDERWARE, *ArchestrA System Platform 2012 with InTouch 2012 - Getting Started Guide*, Invensys, 2011, rev. C, 72 pages.
- WONDERWARE, *GRAccess Toolkit API - User's Guide*, Invensys, 2010, 264 pages.
- TECHNORD, *Présentation de Technord*, Technord, 2015, 41 pages.

Sites web – entreprise et ArchestrA

Technord Groupe, <http://www.technord.com>, 07/05/2016.

Wikipedia the free encyclopedia, *Wonderware*, <https://en.wikipedia.org/wiki/Wonderware>, 21/05/2016.

Wonderware France, *Wonderware System Platform*, http://www.wonderware.fr/Wonderware.html?content_id=68, 21/05/2016.

Wonderware PacWest, *Wonderware System Platform*, <https://wonderwarepacwest.com/solutions/system-platform>, 21/05/2016.

Schneider Electric Software, *Wonderware Supervisory HMI Software Solutions*, <http://software.schneider-electric.com/products/wonderware/hmi-and-supervisory-control>, 21/05/2016.

Schneider Electric Software, *Wonderware System Platform*, <http://software.schneider-electric.com/pdf/brochure/wonderware-system-platform>, 21/05/2016.

ArchestrA.info, *GRAccess 3.1 SP1*, http://archestra.info/index.php/GR_Access_3.1_SP1, 22/05/2016.

Sites web – librairies et leur utilisation

CodePlex, *EPPlus – Create advancedExcel spreadsheets on the server*,
<http://epplus.codeplex.com>, 23/05/2016.

CodePlex, *DocX*,
<https://docx.codeplex.com>, 23/05/2016.

Cathals Corner, *Why did you create DocX*,
<http://cathalscorner.blogspot.be/2010/06/cathal-why-did-you-create-docx.html>, 23/05/2016.

Cathals Corner, *DocX and Tables*,
<http://cathalscorner.blogspot.be/2010/06/docx-and-tables.html>, 20/04/2016.

SourceForge, *.NET Multi-Select TreeView*,
<https://sourceforge.net/projects/mstreeview>, 16/02/2016.

CodeProject, *Painting Your Own Tabs – Second Edition*,
<http://www.codeproject.com/Articles/91387/Painting-Your-Own-Tabs-Second-Edition>,
26/04/2016.

Sites web – développement

Microsoft Developer Network – MSDN, <https://msdn.microsoft.com>, 25/04/2016.

Stack Overflow, <http://stackoverflow.com>, 29/04/2016.

CodeProject, <http://www.codeproject.com>, 29/04/2016.

Net-Informations.com, *C# SQL Server Connection*,
<http://csharp.net-informations.com/data-providers/csharp-sql-server-connection.htm>,
24/02/2016.

ConnectionString.com, *.NET Framework Data Provider for SQL Server connection strings*,
<https://www.connectionstrings.com/sqlconnection>, 24/02/2016.

Microsoft, *Comment faire : trier un contrôle ListView par colonne dans Visual C# .NET*,
<https://support.microsoft.com/fr-fr/kb/319401>, 07/03/2016.

TechOnTheNet, *MSEExcel: Formulas and functions*,
<http://www.techonthenet.com/excel/formulas>, 04/04/2016.

Pingfu, *How to receive WndProc messages in WPF*,
<https://pingfu.net/csharp/2015/04/22/receive-wndproc-messages-in-wpf.html>, 28/04/2016.

Dot Net Perls, *C# Stopwatch Examples*,
<http://www.dotnetperls.com/stopwatch>, 02/05/2016.

Annexes

Annexe 1 – Diagramme des classes

Diagramme des relations entre les classes – modèle et outils

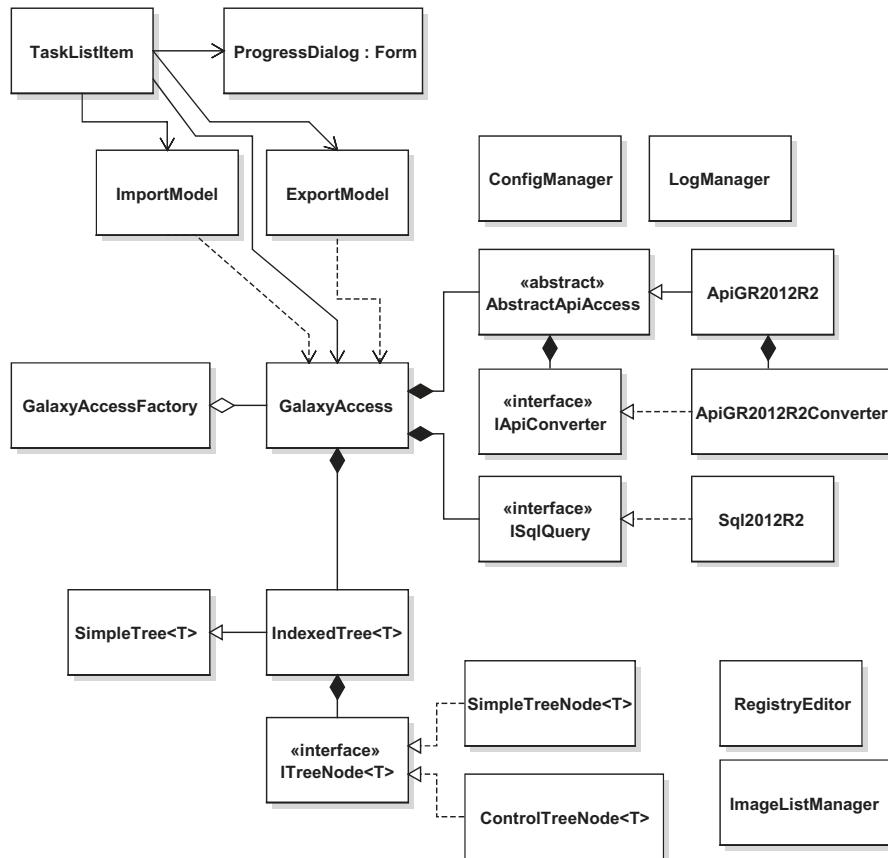


Diagramme des relations entre les classes – objets de transfert et structure

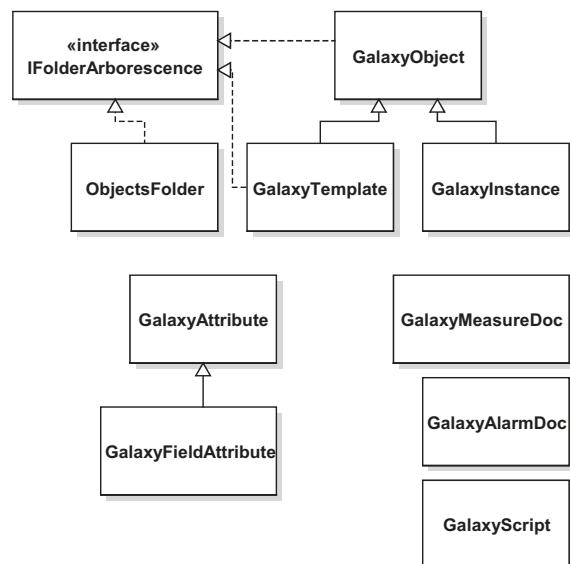


Diagramme des relations entre les classes - interface graphique

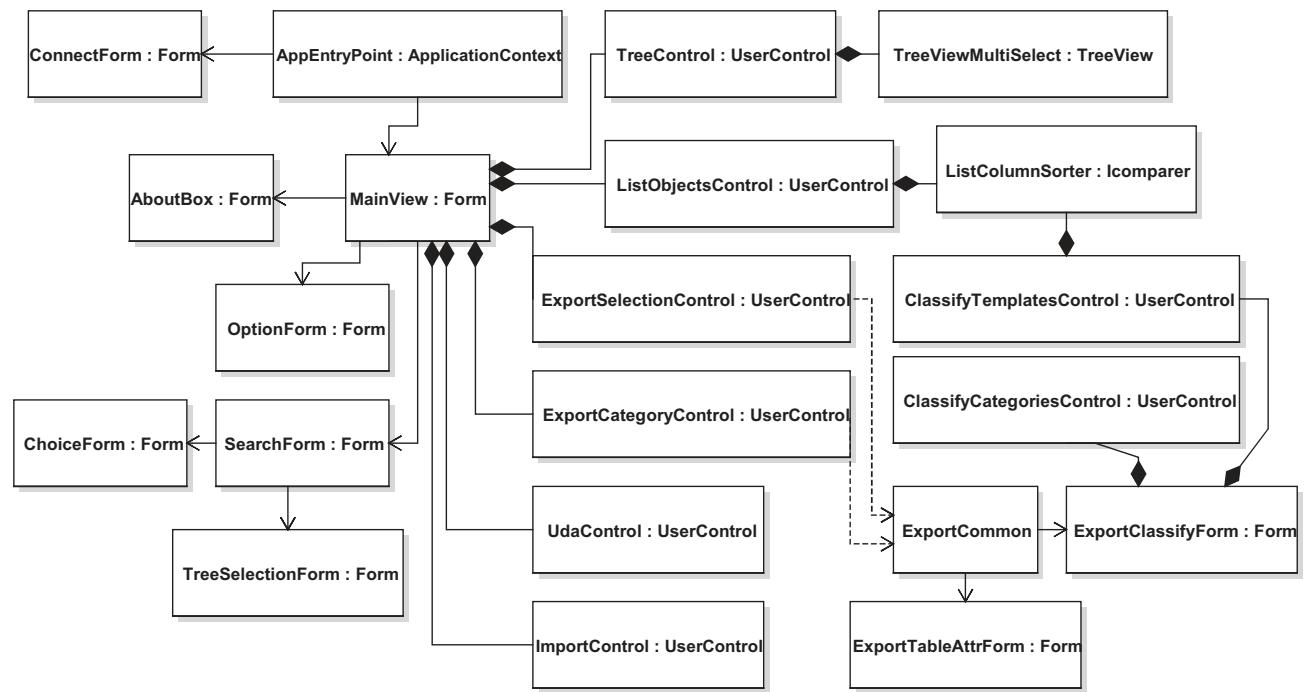
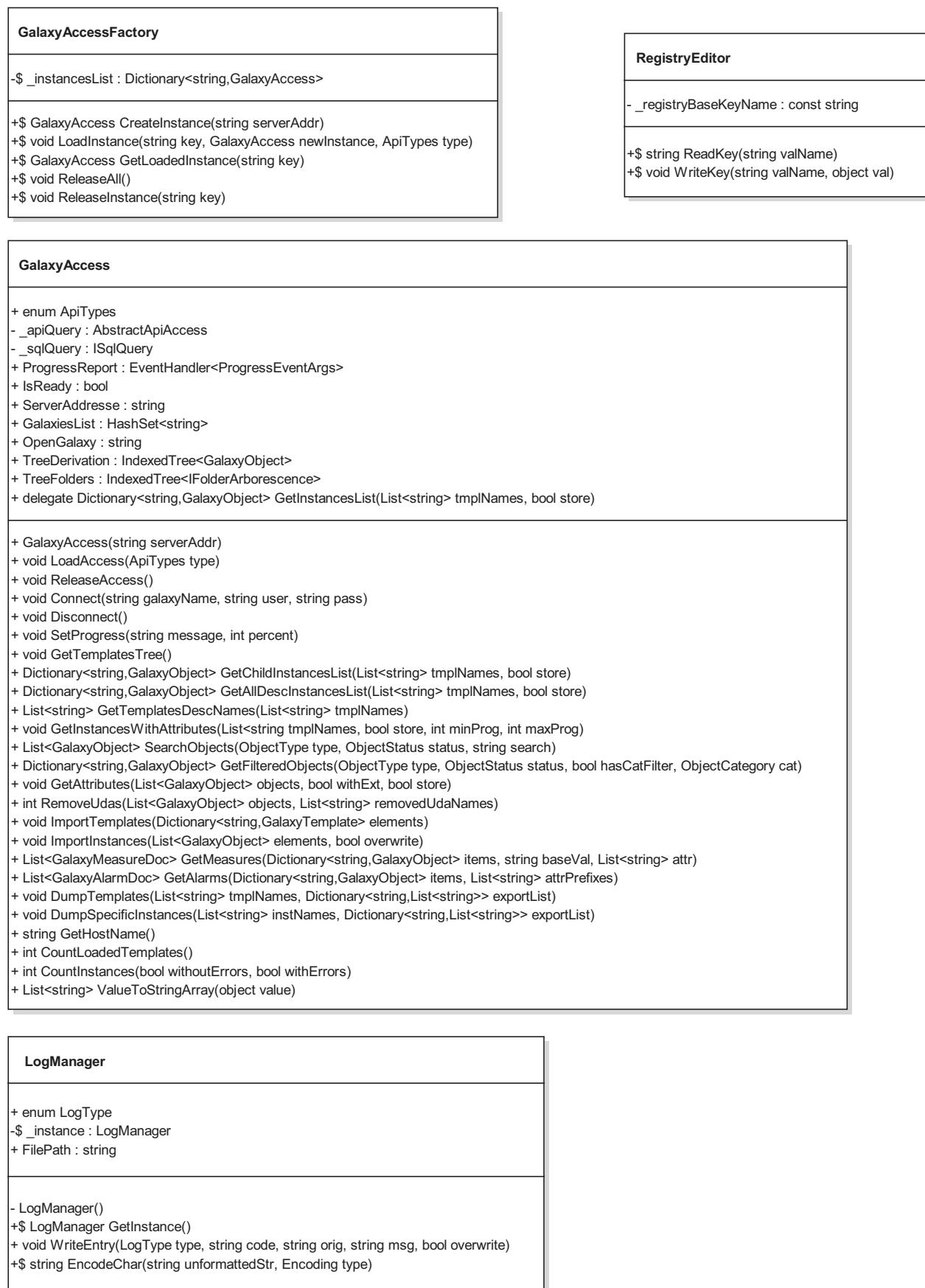


Diagramme des classes principales – modèle et outils



Annexes

```
«abstract»  
AbstractApiAccess  
  
+ GalaxiesNames : HashSet<string>  
+ OpenGalaxyName : string  
+ IsLogged : abstract bool  
+ Converter : IApiConverter  
  
+ AbstractApiAccess()  
+abstract void LoadContext(string serverAddr)  
+abstract void ReleaseContext(bool forceDisconnect)  
+abstract void Connect(string galaxyName, string user, string pass)  
+abstract void Disconnect(bool force)  
+abstract Dictionary<string,ITreeNode<GalaxyObject>> GetAllTemplates()  
+abstract Dictionary<string,GalaxyObject> GetInstancesByTemplate(string tmplName, Dictionary<string, GalaxyObject> instList,  
Dictionary<string, GalaxyObject> tmplStoredInst, bool withAttr)  
+abstract void GetAttributes(Dictionary<string, GalaxyObject> objectsToComplete, bool withExt)  
+abstract int RemoveUdas(List<GalaxyObject> objects, List<string> removedUdaNames)  
+abstract List<GalaxyMeasureDoc> GetMeasures(List<string> instList, string baseVal, List<string> attr)  
+abstract List<GalaxyAlarmDoc> GetAlarms(List<string> instList, List<string> attrPrefixes)  
+abstract void ImportTemplate(GalaxyTemplate template)  
+abstract int ImportInstances(List<GalaxyObject> instances, bool overwrite)  
+abstract void DumpTemplates(List<string> tmplNames, Dictionary<string, List<string>> exportList)  
+abstract void DumpInstances(List<string> instNames, Dictionary<string, List<string>> exportList)
```

```
«interface»  
IApiConverter  
  
+ GalaxyObject ApiToGalaxyObject(bool isTmpl, IgObject source, GalaxyObject existingToComplete, bool hadDetails, bool hasAttr, bool hasExt)  
+ Dictionary<string, GalaxyAttribute> XmlToAttributesList(string xmlSource)  
+ void XmlToExtensionsList(string xmlSrc, out Dictionary<string, List<ExtensionTypes>> outAttrExt, out HashSet<string> outScriptsNames)  
+ void XmlToFieldAttributesList(string xmlSrc, out HashSet<string> outDiscrete, out HashSet<string> outAnalog)  
+ List<string> ValueToStringArray(object value)
```

```
«interface»  
ISqlQuery  
  
+ Success : bool  
+ ServerAddress : string  
+ DatabaseName : string  
  
+ void SetAccess(string database, string serverAddr)  
+ bool CheckAccess()  
+ int CountObjects(ObjectType type, ObjectStatus status, string sqlCond)  
+ List<GalaxyObject> SearchObjects(ObjectType type, ObjectStatus status, string search)  
+ Dictionary<string, ObjectsFolder> GetGalaxyFolders()  
+ List<GalaxyObject> GetFilteredObjects(ObjectType type, ObjectStatus status, bool hasCatFilter, ObjectCategory cat)
```

```
ImportModel  
  
+$ int ReadFile(string appKey, string filePath, out Dictionary<string, GalaxyObject> outData)  
+$ int ReadTemplateSheets(ExcelPackage package, Dictionary<string, GalaxyObject> resultsToComplete)  
+$ bool CheckValueIntegrity(DataTypes type, string value)  
+$ void ImportData(string appKey, Dictionary<string, GalaxyObject> elements, bool overwrite)
```

Annexes

ApiGR2012R2 : AbstractApiAccess

```
+ IsLogged : override bool
- _accessCtx : GRAccessAP
- _galaxies : IGalaxies
- _openGalaxy : IGalaxy

+ ApiGR2012R2()
+ void LoadContext(string serverAddr)
+ void ReleaseContext(bool forceDisconnect)
+ void Connect(string galaxyName, string user, string pass)
+ void Disconnect(bool force)
+ Dictionary<string,ITreeNode<GalaxyObject>> GetAllTemplates()
+ Dictionary<string, GalaxyObject> GetInstancesByTemplate(string tmplName, Dictionary<string, GalaxyObject> instList,
    Dictionary<string, GalaxyObject> tmplStoredInst, bool withAttr)
+ void GetAttributes(Dictionary<string, GalaxyObject> objectsToComplete, bool withExt)
+ int RemoveUdas(List<GalaxyObject> objects, List<string> removedUdaNames)
- int RemoveUdasFromApiObjects(IgObjects elements, List<string> removedUdaNames)
+ List<GalaxyMeasureDoc> GetMeasures(List<string> instList, string baseVal, List<string> attr)
+ List<GalaxyAlarmDoc> GetAlarms(List<string> instList, List<string> attrPrefixes)
- SetFieldAlarmValues(IgObject inst, string almName, string descr, List<string> attrPrefixes)
- SetUdaAlarmValues(IgObject inst, string almName, string descr, List<string> attrPrefixes)
+ void ImportTemplate(GalaxyTemplate template)
+ int ImportInstances(List<GalaxyObject> instances, bool overwrite)
- void ImportInstanceValues(IInstance instance, GalaxyObject values)
- MxValue SetUdaValue(DataTypes type, string valueStr)
+ void DumpTemplates(List<string> tmplNames, Dictionary<string, List<string>> exportList)
+ void DumpInstances(List<string> instNames, Dictionary<string, List<string>> exportList)
- void DumpInstancesPerTemplate(string templateName, List<IgObject> inst, Dictionary<string, List<string>> exportList)
- List<string> DumpBaseData(IgObject inst)
```

ApiGR2012R2Converter : IApiConverter

```
+ GalaxyObject ApiToGalaxyObject(bool isTmpl, IgObject source, GalaxyObject existingToComplete, bool hadDetails, bool hasAttr, bool hasExt)
- GalaxyObject ApiToAttributesLists(IgObject objectToConvert, GalaxyObject result)
- GalaxyObject ApiToExtensionsLists(IgObject objectToConvert, GalaxyObject result)
- GalaxyObject ApiToFieldAttributesLists(IgObject objectToConvert, GalaxyObject result)
- GalaxyAttribute ApiToAttribute(IAttribute attr)
+ Dictionary<string, GalaxyAttribute> XmlToAttributesList(string xmlSource)
+ void XmlToExtensionsList(string xmlSrc, out Dictionary<string, List<ExtensionTypes>> outAttrExt, out HashSet<string> outScriptsNames)
+ void XmlToFieldAttributesList(string xmlSrc, out HashSet<string> outDiscrete, out HashSet<string> outAnalog)
+ List<string> ValueToStringArray(object value)
```

SqI2012R2 : ISqlQuery

```
+ Success : bool
+ ServerAddress : string
+ DatabaseName : string
- ConnectionString : string

+ SqI2012R2(string serverAddr)
+ void SetAccess(string database, string serverAddr)
+ bool CheckAccess()
+ int CountObjects(ObjectType type, ObjectStatus status, string sqlCond)
+ List<GalaxyObject> SearchObjects(ObjectType type, ObjectStatus status, string search)
+ Dictionary<string, ObjectsFolder> GetGalaxyFolders()
+ List<GalaxyObject> GetFilteredObjects(ObjectType type, ObjectStatus status, bool hasCatFilter, ObjectCategory cat)
- int CountObjectsQuery(string queryCond, SqlParameter[] param)
- List<GalaxyObject> GetObjectsQuery(string queryCond, SqlParameter[] param)
- Dictionary<string, ObjectsFolder> GetFoldersQuery(string queryCond, SqlParameter[] param)
- bool CheckFolderParent(Dictionary<string, ObjectsFolder> index, string parentName)
```

Annexes

<p>«interface»</p> <p>ITreeNode<T></p> <hr/> <p>+ Content : T + Parent : ITreeNode<T> + ParentName : string + ChildNodes : List<ITreeNode<T>></p> <hr/> <p>+ void ListNodesNames(List<string> names) + void ListNodes(List<ITreeNode<T>> nodes) + void AddChild(ITreeNode<T> node) + void RemoveChild(ITreeNode<T> node)</p>	<p>SimpleTreeNode<T> : ITreeNode<T>, IComparable<SimpleTreeNode<T>></p> <hr/> <p>+ Content : T + Parent : ITreeNode<T> + ParentName : string + ChildNodes : List<ITreeNode<T>></p> <hr/> <p>+ SimpleTreeNode(T nodeContent, string parent) + SimpleTreeNode(T nodeContent, ITreeNode<T> parent) + void ListNodesNames(List<string> names) + void ListNodes(List<ITreeNode<T>> nodes) + void AddChild(ITreeNode<T> node) + void RemoveChild(ITreeNode<T> node)</p>
	<p>ControlTreeNode<T> : Forms.TreeNode, ITreeNode<T>, IComparable<ControlTreeNode<T>></p> <hr/> <p>+ Content : T + Parent : ITreeNode<T> + ParentName : string + ChildNodes : List<ITreeNode<T>></p> <hr/> <p>+ ControlTreeNode(T nodeContent, int imgIndex, string parent) + ControlTreeNode(T nodeContent, ControlTreeNode<T> parent, int imgIndex) + void ListNodesNames(List<string> names) + void ListNodes(List<ITreeNode<T>> nodes) + void AddChild(ITreeNode<T> node) + void RemoveChild(ITreeNode<T> node)</p>
	<p>SimpleTree<T> : IEnumerable<ITreeNode<T>></p> <hr/> <p>+ RootNode : ITreeNode<T> + ChildNodes : List<ITreeNode<T>></p> <hr/> <p>+ SimpleTree(ITreeNode<T> rootNode) +virtual List<string> ListNodesNames() +virtual List<ITreeNode<T>> ListNodes() +virtual void BuildFromIndex(Dictionary<string,ITreeNode<T>> lookup, bool updateArg) + Dictionary<string,ITreeNode<T>> BuildFromList(List<ITreeNode<T>> nodesList)</p>
	<p>IndexedTree<T> : SimpleTree<T>, IEnumerable<ITreeNode<T>></p> <hr/> <p>+ NodesIndex : Dictionary<string,ITreeNode<T>></p> <hr/> <p>+ IndexedTree(ITreeNode<T> rootNode) + List<string> ListNodesNames() + List<ITreeNode<T>> ListNodes() + void BuildFromIndex(Dictionary<string,ITreeNode<T>> lookup, bool updateArg) + void Insert(ITreeNode<T> node) + void Remove(ITreeNode<T> node)</p>

Annexes

<p>TaskListItem</p> <pre>+ enum TaskTypes -\$ _tasksQueue : LinkedList<TaskListItem> -\$ _mutexQueue : Mutex + SessionNumber : int + TaskName : string + TaskType : TaskTypes + Priority : int + IsInterruptable : bool + CancelToken : CancellationTokenSource - AsyncTask : Task + delegate void TaskResult() - Callback : TaskResult - TaskListItem(int sess, Task act, TaskResult callback, string name, TaskTypes type, int prior, bool isInt) + ~TaskListItem() +\$ void SetNewTask(int sess, Task act, TaskResult callback, string name, TaskTypes type, int prior, bool isInt) +\$ bool IsTaskCanceled() +\$ void StopTaskIfCanceled() -\$ void InsertTask(TaskListItem taskVal) +\$ void CancelCurrentTask() +\$ void RemoveCompletedTask(Exception errIndic) +\$ void ClearTasks() +\$ int CountWaitingTasks()</pre>	<p>ConfigManager</p> <pre>+ enum ListColumns + ColumnNames : string[] + enum ExportedTemplateTypes + ExportedTemplateName : string[] + ExportedTemplateSizes : int[] + enum ExportedInstanceTypes + ExportedInstanceName : string[] + ExportedInstanceSizes : int[] -\$ _instance : ConfigManager + AutoSnapshot : bool + ConfigFilePath : string + EntrepriseId : string + InternationalLocale : int + AttrDescrLang : string + IsAltStartupLayout : bool + UseCache : bool + DefaultFolderPath : string + SearchMaxHistory : int + SearchColumns : bool[] + InstancesColumns : bool[] + ImportExportColumns : bool[] + ExportGeneralTabName : string + ExpGenTemplatesValues : bool[] + ExpGenInstancesValues : bool[]</pre>
<p>ProgressDialog : Form</p> <pre>delegate void SetProgressionCallback(string label, int percent) + Handler : EventHandler<ProgressEventArgs> - _progressDivider : int - _progressStep : int -\$ _instancesList : List<ProgressDialog> - ProgressDialog() +\$ ProgressDialog GetInstance(int sessionNumber) +\$ void ClearInstances() + void SetProgression(string label, int percent) +\$ void IncrementProgressionStep(string stepTitle, bool isCancellable) + void IncrementDivision() + void SetTasksNumber(int number) - void ProgressReport_handler(object sender, ProgressEventArgs e) + void ShowForm(string title, string label, int divider, bool isCancellable) +\$ void HideAll() - void btnCancel_Click(object sender, EventArgs e)</pre>	<pre>- ConfigManager() +\$ ConfigManager GetInstance() + void Save() + void Load() + Dictionary<string,string> GetListColumns(bool[] listColumns) + Dictionary<string,string> GetInstancesColumns() + Dictionary<string,string> GetSearchColumns() + Dictionary<string,string> GetImportExportColumns()</pre>
<p>ExportModel</p> <pre>+ enum ExportTypes +\$ ExportTitles : string[] + struct TemplateExportCategory_t : { string, List<GalaxyObject> } +\$ List<GalaxyObject> GetDataGeneral(string appKey, Dictionary<string, GalaxyObject> elements) +\$ Dictionary<string,List<string>> GetDataArchestra(string appKey, Dictionary<string, GalaxyObject> elements) +\$ void GetDataDocTemplates(string appKey, List<GalaxyObject> elements) +\$ List<GalaxyMeasureDoc> GetDataDocMeasures(string appKey, Dictionary<string, GalaxyObject> elements, string baseVal, List<string> attr) +\$ List<GalaxyAlarmDoc> GetDataDocAlarms(string appKey, Dictionary<string, GalaxyObject> elements, List<string> attrPrefixes) +\$ void ExportGeneral(string appKey, string filePath, List<GalaxyObject> objects) -\$ int ExportGeneralInfos(ExcelWorksheet sheet) -\$ void ExportGeneralTemplate(GalaxyAccess access, ExcelWorksheet tmplSheet, GalaxyTemplate tmpl) +\$ void ExportArchestra(string appKey, string filePath, Dictionary<string,List<string>> data) +\$ void ExportDocTemplates(string filePath, List<string> listOrder, Dictionary<string,List<TemplateExportCategory_t>> obj, string projTitle) +\$ void ExportDocMeasures(string filePath, List<GalaxyMeasureDoc> objects) +\$ void ExportDocAlarms(string filePath, List<GalaxyAlarmDoc> objects) -\$ string SetDefaultFile(string fileName) -\$ void SetExcelTitleColor(ExcelRange range)</pre>	<p>ImageListManager</p> <pre>+\$ IconsCollections : Dictionary<string,ImageList> +\$ ImageList AddIconsFromSpritesheet(string title, string imageRes, int iconSize)</pre>

Diagramme des classes principales – interface graphique



Annexes

<p>TreeControl : UserControl</p> <pre>+ enum TreeChoice + enum SnapshotStatus - _appKey : string - _sessionNumber : int - _isFirstLoad : bool + LoadedEvent : EventHandler<DataEventArgs> + SelectionEvent : TreeViewEventHandler + DoubleClickEvent : EventHandler - _isModifyingSelection : bool - _SavedNodesNames : List<string>[] + _SelectionNames : List<string> + SelectedObjects : List<GalaxyObject></pre> <pre>+ TreeControl() + void Init(string appKey, int sessionNumber) + void InitData() - void OnTemplatesTreesLoaded() - void treeTabs_SelectedIndexChanged(object sender, EventArgs e) - void TreeSelection_handler(object sender, TreeViewEventArgs e) - void SelectDerivationByName(bool replaceCur, List<string> selectionNames) - void SelectFoldersByName(bool replaceCur, List<string> selectionNames) + void SelectByName(TreeChoice tree, bool replaceCur, List<string> selectionNames) - void SelectByTreeAndName(List<string>[] selectionNames) + void SaveCurrentSelectedNames() + void SynchronizeSelection() + void LoadSnapshot(bool isAutoLoad) + void SaveSnapshot(bool isAutoSave) - List<string> GetExpandedNodesNameList<T>(IndexedTree<T> source) - void GetExpandedNodeName<T>(ControlTreeNode<T> top, List<string> expandedNodes) - void TreeDerivation_DoubleClick(object sender, EventArgs e) - void TreeFolders_DoubleClick(object sender, EventArgs e)</pre>	<p>ListObjectsControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int - _displaySession : int + LoadedEvent : EventHandler + SelectionEvent : EventHandler + DoubleClickEvent : EventHandler - _columnSorter : ListColumnSorter - _columns : Dictionary<string, string> - _columnsProperties : PropertyInfo[] + DataSource : Dictionary<string, GalaxyObject> - _dataModel : List<ListViewItem> - _listMutex : Mutex - _isSelectionBusy : bool + SelectionCount : int + SelectionNames : List<string> + SelectedObjects : List<GalaxyObject> - _isFilterPanelOpen : bool - _isMouseDown : bool</pre>
<p>ExportCategoryControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int</pre> <pre>+ ExportCategoryControl() + void Init(string appKey, int sessionNumber) - void btnExport_Click(object sender, EventArgs e)</pre>	<p>ListColumnSorter : IComparer</p> <pre>- _objectCompare : CaseInsensitiveComparer + SortColumn : int + Order : SortOrder</pre> <pre>+ ListColumnSorter() + int Compare(object x, object y)</pre>
<p>ExportSelectionControl : UserControl</p> <pre>+ enum ExportTypes - _appKey : string - _sessionNumber : int - _displaySession : int - _templatesTreeRef : TreeControl - _instancesListRef : ListObjectsControl</pre> <pre>+ ExportSelectionControl() + void Init(string appKey, int sess, int disp, TreeControl treeRef, ListObjectsControl listRef) + void RemoveSelection() - void btnRemove_Click(object sender, EventArgs e) - void btnGetTree_Click(object sender, EventArgs e) - void btnGetList_Click(object sender, EventArgs e) + void InsertTargets(List<GalaxyObject> newObjects) - void btnEmpty_Click(object sender, EventArgs e) + void ClearTargetsList() + void UpdateColumns() - void listSelection_DragEnter(object sender, DragEventArgs e) - void listSelection_DragDrop(object sender, DragEventArgs e) - void btnExport_Click(object sender, EventArgs e)</pre>	<p>ImportControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int + ErrorCount : int + ReadData : Dictionary<string, GalaxyObject> + ImportedEvent : EventHandler</pre> <pre>+ ImportControl() + Init(string appKey, int sessionNumber, int displaySession) + void RemoveSelection() - void btnRemove_Click(object sender, EventArgs e) - void btnClear_Click(object sender, EventArgs e) + void ClearImportList() + void UpdateColumns() - void btnFileOpen_Click(object sender, EventArgs e) + void OnFileImported() - void btnImport_Click(object sender, EventArgs e) + void OnImportComplete()</pre>
<p>ExportCommon</p> <pre>+\$ void LaunchExport(string appKey, int sess, bool isSelect, int type, Dictionary<string, GalaxyObject> obj)</pre>	

Annexes

<p>ExportSelectionControl : UserControl</p> <pre>+ enum ExportTypes - _appKey : string - _sessionNumber : int - _displaySession : int - _templatesTreeRef : TreeControl - _instancesListRef : ListObjectsControl</pre> <pre>+ ExportSelectionControl() + void Init(string appKey, int sess, int disp, TreeControl treeRef, ListObjectsControl listRef) + void RemoveSelection() - void btnRemove_Click(object sender, EventArgs e) - void btnGetTree_Click(object sender, EventArgs e) - void btnGetList_Click(object sender, EventArgs e) + void InsertTargets(List<GalaxyObject> newObjects) - void btnEmpty_Click(object sender, EventArgs e) + void ClearTargetsList() + void UpdateColumns() - void listSelection_DragEnter(object sender, DragEventArgs e) - void listSelection_DragDrop(object sender, DragEventArgs e) - void btnExport_Click(object sender, EventArgs e)</pre>	<p>ImportControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int + ErrorCount : int + ReadData : Dictionary<string, GalaxyObject> + ImportedEvent : EventHandler</pre> <pre>+ ImportControl() + Init(string appKey, int sessionNumber, int displaySession) + void RemoveSelection() - void btnRemove_Click(object sender, EventArgs e) - void btnClear_Click(object sender, EventArgs e) + void ClearImportList() + void UpdateColumns() - void btnFileOpen_Click(object sender, EventArgs e) + void OnFileImported() - void btnImport_Click(object sender, EventArgs e) + void OnImportComplete()</pre>
<p>UdaControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int - _displaySession : int - _templatesTreeRef : TreeControl - _instancesListRef : ListObjectsControl - _selectedObjects : List<GalaxyObject> - _selectionMutex : Mutex + LayoutOrientation : Orientation</pre> <pre>+ UdaControl() + void Init(string appKey, int sess, int disp, TreeControl treeRef, ListObjectsControl listRef) - void btnGetTree_Click(object sender, EventArgs e) - void btnGetList_Click(object sender, EventArgs e) + void SetObjectsList(List<GalaxyObject> selected) - void LoadUdas(bool mutex) - void OnUdasLoaded() - void btnRemove_Click(object sender, EventArgs e) - void OnUdasRemoved</pre>	<p>ExportCategoryControl : UserControl</p> <pre>- _appKey : string - _sessionNumber : int</pre> <pre>+ ExportCategoryControl() + void Init(string appKey, int sessionNumber) - void btnExport_Click(object sender, EventArgs e)</pre>
	<p>ExportTableAttrForm : Form</p> <pre>- _busy : bool - _exportType : int + BaseValue : string + ListAttributes : List<string></pre> <pre>+ ExportTableAttrForm(int exportType) - void btnOk_Click(object sender, EventArgs e) - void btnCancel_Click(object sender, EventArgs e) - void btnAddAttr_Click(object sender, EventArgs e) - void btnRemoveAttr_Click(object sender, EventArgs e)</pre>
<p>ExportCommon</p> <pre>+\$ void LaunchExport(string appKey, int sess, bool isSelect, int type, Dictionary<string, GalaxyObject> obj)</pre>	