as2lib - a little introduction

Christoph Atteneder, Martin Heidegger,
Michael Herrmann, Alexander Schliebner and Simon Wacker.
Special thanks to Nicolas Desy for translation.

1. June 2004

Table des matières

1 Historique du projet **2**

2 Core Package **3**
2.1 BasicInterface 3
2.2 BasicClass 3

3 Output Handling **5**

4 Exception Handling **8**

5 Event Handling **12**

6 Reflections **15**

7 Data Holding **18**

8 Overloading **22**

9 Test Cases **24**

10 Speed Tests **27**

11 Preview **30**
11.1 Connection Handling 30
11.2 as2lib console 30
11.2.1 Requirements 30
11.2.2 The as2lib console 31
11.3 Roadmap 32

Chapitre 1

Historique du projet

as2lib, une librairie *Open Source* pour *ActionScript 2*, a été fondé en Septembre 2003 dans le but de créer de meilleures possibilités de programmation en *ActionScript 2*. L'équipe de développement travaille sur les problèmes majeurs de Flash et essaie de résoudre ces problèmes quotidiens avec Flash. Une des caractéristiques les plus importantes d'*as2lib* est que c'est publié sous la licence *MPL* (Mozilla Public License). Cela veut dire que c'est gratuit pour les projets personnels et commerciaux. Après avoir défini les concepts de bases et les tâches à accomplir, l'équipe a séparé le travail en package pour les 5 membres de l'équipe.

Nom	*Rôle*	*Site web*	*Nationalité*
Atteneder Christoph	Développeur	www.cubeworx.com	Autriche
Heidegger Martin	Gestion de projet	www.traumwandler.com	Autriche
Herrmann Michael	Développeur		Autriche
Schliebner Alexander	Co-fondateur	www.schliebner.de	Allemagne
Wacker Simon	Chef Développeur	www.simonwacker.com	Allemagne

TAB. 1.1 – Membre actif de *as2lib*

FIG. 1.1 – www.as2lib.org

Chapitre 2

Core Package

Motivation : Fournir des fonctionnalités spécifiques dans toutes les classes afin de simplifier le développement et la réparation de bug.

Solution : Toutes les classes, interfaces et packages d'as2lib suivent le même guide. Les classes de fondation sont dans le package ***core***[1].

2.1 BasicInterface

Afin d'obtenir une meilleure définition des classes, les fonctionnalités d'*as2lib* sont définies sous forme d'*interface* de façon intensive. Toutes les interfaces d'*as2lib* héritent de l'interface *BasicInterface*, ce qui assure d'avoir ces fonctionnalités dans toutes les classes :

- **getClass()** :*ClassInfo* - Cette méthode offre l'information exacte sur la classe au moment où la méthode est invoqué. L'information retournée est de type *ClassInfo* et contient le nom de la classe, méthodes, propriétés, le nom complet incluant le package et les classes Mères.
- **toString()** :*String* - Cette méthode retourne la représentation de l'objet sous forme de String.

La logique de la méthode getClass est fournie par la classe *BasicClass* (voir fig. 2.1, S. 4).

2.2 BasicClass

La classe de base d'*as2lib* est *BasicClass*. Toutes les classes d'*as2lib* sont directement ou indirectement dérivées de *BasicClass*. Elle implémente *BasicInterface* et fournie, à l'aide des classes *ReflectUtil*[2] et *ObjectUtil*[3], la logique de ces méthodes qui doivent être implémenté :

[1] org.as2lib.core.*
[2] org.as2lib.env.util.ReflectUtil
[3] org.as2lib.util.ObjectUtil

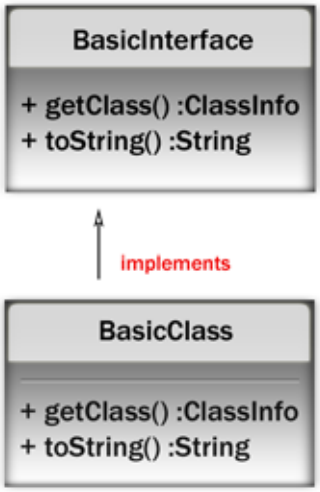

FIG. 2.1 – Partie principale du package *org.as2lib.core*

- **getClass()** :*ClassInfo* - Pour la définition, voir la documentation de *BasicInterface*. La création de l'information d'une classe est possible grâce au package reflection d'*as2lib* (org.as2lib.env.reflect) voir 6.
- **toString()** :*String* - Cette méthode retourne la représentation de l'objet sous forme de String.

Chapitre 3

Output Handling

Motivation : La sortie normale d'une application Flash est fait via l'opération interne :

```
trace(expression);
```

La sortie d'un trace est néanmoins seulement visible qu'à l'intérieur d'un environnement de développement qui supporte cette opération. Dans tous les autres cas (ex : dans un application Web), il n'y a pas de standard de défini. Une librairie devrait fournir une sortie standardisée pour les utilisateurs et les développeurs dans tous les cas possibles.

Solution : Pour avoir plusieurs types de sortie dans tous les environnements[1], il suffit d'utiliser la classe *Out*[2]. Cela donne la possibilité par exemple de sauvegarder les messages d'erreur côté serveur, afin de s'assurer que l'erreur soit perceptible seulement pour le développeur et non pour le client. La classe *Out* traite les requêtes selon la configuration de un ou plusieurs *OutputHandler*[3]. *As2lib* offre une sortie standardisée pour un nombre illimité d'interfaces.

Usage : Un exemple simple de l'utilisation de la gestion de sortie d'*as2lib* est vu dans la figure 3.1,p. 6. Après avoir créé une instance de la classe *Out* et fournie le *TraceHandler*[4], on peut procédé à l'utilisation.

La séquence de chaque action est déterminée par leurs numéros. On peut, comme dans la figure 3.1, p. 6, utilisé des sorties prédéfini comme le *TraceHandler* ou *ExternalConsoleHandler*[5] ou bien faire sa propre implémentation

[1] Flash peut rouler dans le player, dans *Macromedia Central*, dans un application compilé (*.exe)

[2] org.as2lib.env.out.Out

[3] org.as2lib.env.out.OutputHandler, org.as2lib.env.out.handler.*

[4] org.as2lib.env.out.handler.TraceHandler

[5] org.as2lib.env.out.handler.ExternalConsoleHandler

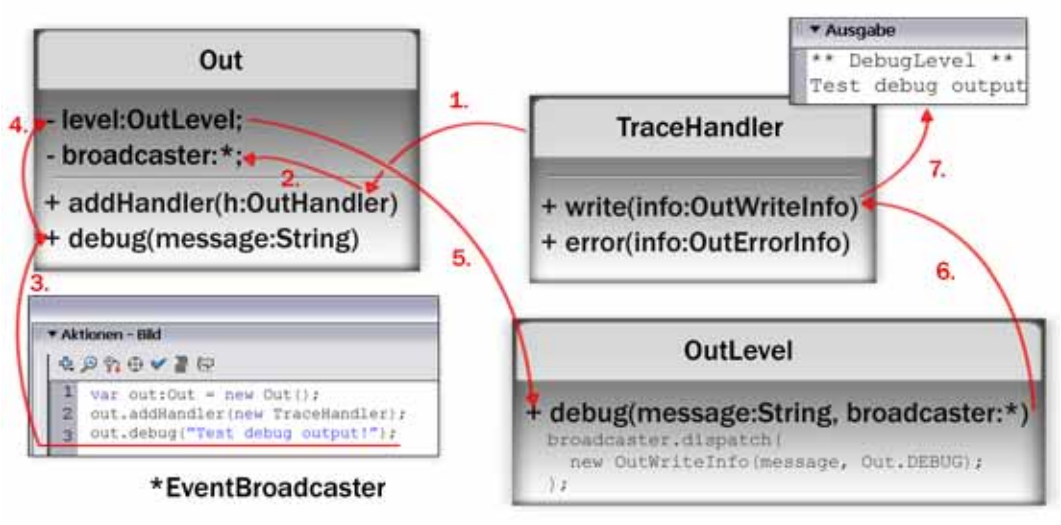

FIG. 3.1 – Utilisation de la gestion de sortie d'*as2lib*

de l'interface *OutputHandler*(ex. : sortie avec le component Alert de Macromedia)[6] :

```
import org.as2lib.env.event.EventInfo;
import org.as2lib.env.out.OutHandler;
import org.as2lib.env.out.info.OutWriteInfo;
import org.as2lib.env.out.info.OutErrorInfo;
import org.as2lib.env.out.OutConfig;
import org.as2lib.core.BasicClass;
import mx.controls.Alert;

class test.org.as2lib.env.out.handler.UIAlertHandler
  extends BasicClass implements OutHandler {

  public function write(info:OutWriteInfo):Void {
    Alert.show(info.getMessage(),
      getClass().getName());
  }

  public function error(info:OutErrorInfo):Void {
    Alert.show(
      OutConfig.getErrorStringifier().execute(info),
      getClass().getName()
```

[6]Le component Alert de *Macromedia* doit être dans la librairie et vous devez posséder *Flash MX 2004 Professional*, pour accéder à la classe Alert.

```
      );
   }
}
```

Par la définition d'un niveau de sortie (ex. : *aOut.setLevel(Out.DEBUG)*), il est possible d'empêcher la sortie de certaines informations. Les niveaux possibles sont :

- Out.ALL
- Out.DEBUG
- Out.INFO
- Out.WARNING
- Out.ERROR
- Out.FATAL
- Out.NONE

Cette graduation permet un débuguage clair durant le développement et une réécriture rapide de la sortie lorsque l'application est terminée. Durant le développement (ex. : avec *DEBUG*), toutes les informations ayant un niveau inférieur à celui sélectionné sont utilisés (*DEBUG*) : *DEBUG*, *INFO*, *WARNING*, *ERROR* et *FATAL*. Il n'y a que *LOG* qui est omis.

```
var aOut = new Out();

aOut.setLevel(Out.DEBUG);

aOut.log("log me Please!");
aOut.debug("debug me Please!");
aOut.info("inform me Please!");
aOut.warning("warn me Please!");
aOut.error(new Exception("Output Error", this));
aOut.fatal(new FatalException("Fatal Output Error",
  this));
```

Lorsque l'application est finie, sortir seulement les erreurs fatales peut être fait avec une seule ligne de code.

```
aOut.setLevel(Out.FATAL);
```

Chapitre 4

Exception Handling

Motivation : Les erreurs non attrapées sont affichées ainsi[1] :

```
trace(Error.toString());
```

De plus, l'information affichée est peu ou pas du tout informative (Il n'y a que "Error" ou le String envoyé au constructeur qui est affiché. Voir fig. 4.1, p. 8). L'affichage des messages d'erreurs est seulement possible qu'avec *Flash MX 2004*.

FIG. 4.1 – La sortie d'un erreur (*throw new Error("Un erreur est survenu !") ;*) dans Flash MX 2004.

Solution : *as2lib* contient des classes basées sur la classe native de *Macromedia*, et implémente les méthodes de l'interface *Throwable*[2]. Les nouvelles fonctionnalités de la gestion des exceptions sont :

- Toutes les opérations qui sont invoqué avant que l'exception survienne sont sauvegardées dans un Stack afin d'accélérer la recherche des erreurs. Avec l'aide de *Reflections*, voir chapitre 6, le nom du message d'erreur est généré automatiquement.
- Les exceptions peuvent être facilement englobable dans d'autres exceptions.

Voici les exceptions prédéfinies qui sont fournis :

[1] Documentation en ligne livedocs.macromedia.com/flash/mx2004/main/12_as217.htm
[2] org.as2lib.env.except.Throwable

- *AbstractException* : Toutes les exceptions dérive cette classe. Elle implémente les méthodes que l'interface Throwable définie (voir fig. 4.2, p. 9).
- *Throwable* : Interface qui force l'implémentation des méthodes suivantes :
  - **getStackTrace(Void) :***Stack* - Retourne toutes les opérations invoqué avant que l'exception survienne.
  - **initCause(cause :Throwable) :***Throwable* - Déclare la raison de l'exception et ne peut être utiliser qu'une seule fois. Cette méthode est normalement utilisée afin de ne pas perdre l'information lorsqu'une exception est la cause d'une autre exception.
  - **getCause(Void) :***Throwable* - Retourne la cause de l'exception.
  - **getMessage(Void) :***String* - Retourne le message qui a été envoyé au constructeur de l'exception.
- *Exception* : Est un implémentation standard de l'interface Throwable et hérite de la classe AbstractException.
- *FatalException* : FatalException a une priorité plus élevée que Exception.
- *As2lib* défini déjà quelque exceptions :
  - IllegalArgumentException
  - IllegalStateException
  - UndefinedPropertyException
  - ...

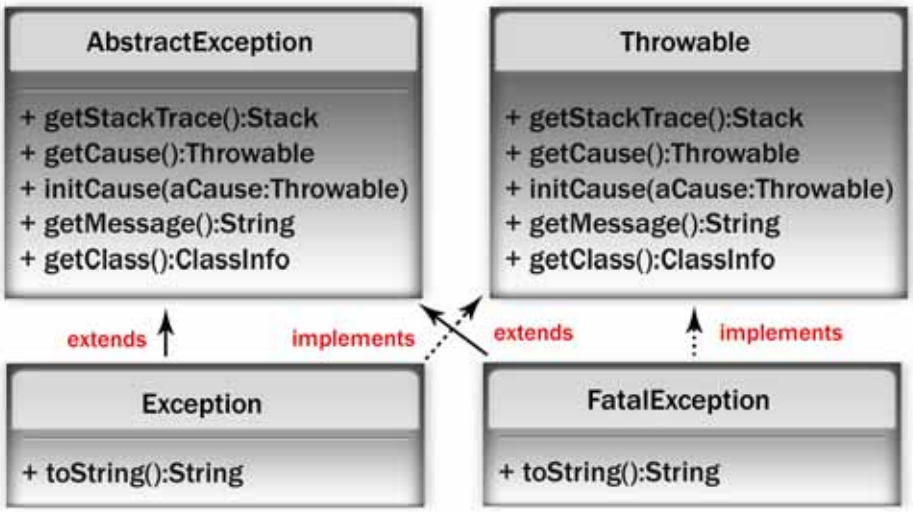

FIG. 4.2 – Basic hierachy of the *as2lib Exception* package

Usage : Dans le cas d'une valeur incorrecte envoyé en paramètre, on lance une IllegalArgumentException de la façon suivante :

```
import org.as2lib.env.except.IllegalArgumentException;
...
throw new IllegalArgumentException(''Wrong Parameter.'',
  this,
  arguments);
```

Pour toutes les exceptions, trois paramètres sont nécessaires :

1. *message* - ex : "Parametre invalide." - Un texte qui rend la raison de l'exception claire.
2. *thrower* - ex. : "this" - Référence à la classe ou l'object qui à généré l'exception.
3. *arguments* - Une variable intrinsic de Flash, qui contient les paramètres envoyés à la méthode.

Voici comment attraper une IllegalArgumentException avec un bloc *try-catch*.

```
import org.diplomarbeit.ExceptionTest;
import org.as2lib.env.except.IllegalArgumentException;
import org.as2lib.env.out.Out;

try {
  var myOut:Out = new Out();
  var myET:ExceptionTest = new ExceptionTest();
  myOut.info(myET.getString());
}
catch(e:IllegalArgumentException){
  myOut.error(e);
}
```

Vous pouvez vous définir autant d'exception que vous voulez. Si par exemple, vous voulez lancer une OutOfTimeException, vous devez faire l'implémentation suivante :

```
import org.as2lib.env.except.Exception;

class OutOfTimeException extends Exception {

public function OutOfTimeException(message:String,
  thrower, args:FunctionArguments) {
  super (message, thrower, args);
}
}
```

La classe Exception est héritée donc le constructeur de la classe Exception doit être invoqué.

Chapitre 5

Event Handling

Motivation : Les événements en *Flash* sont très intensif sur la performance et sont une partie essentielle de l'interface de l'usager. La plupart des développeurs utilisent cette fonctionnalité à l'aide de la classe intrinsic *AsBroadcaster* (une fonctionnalité non documenté de Flash) ou la classe *EventDispatcher*[1] de *Macromedia*. Il n'y a pas de définition exacte de disponible pour un *EventListener* ou pour les événements ou les arguments. Il manque des informations et définitions à propos des événements pour les développeurs.

Solution : Voici les problèmes à faire face :

- Les développeurs d'objets doivent définir quels événements seront attrapés.
- Les développeurs d'écouteur doivent implémenter tous les événements.
- Les développeurs d'objets doivent être capable de déclencher des événements.
- Les développeurs d'objets doivent pouvoir ajouter de l'information à un événement.

As2lib supporte la gestion des événements car c'est une partie essentielle du développement d'une application. Si vous utilisez la classe intrinsic *AsBroadCaster*, cela peut devenir une implémentation déficiente. La classe *EventDispatcher* de *Macromedia* n'est pas gratuite, ex : elle est seulement disponible lorsque vous achetez *Macromedia Flash*, et elle ne supporte pas toutes les fonctionnalités requises.

L'interface la plus importante du package *event* que le développeur utilise est *EventBroadcaster*[2]. Vous pouvez ajouter autant d'écouteurs que vous voulez,

```
addListener(listener:EventListener)
```

[1] mx.events.EventDispatcher
[2] org.as2lib.env.event.EventBroadcaster

et les enlever si vous le souhaitez.

```
removeListener(listener:EventListener)
```

Les objets écouteurs doivent implémenter l'interface *EventListener*. Pour vos propres projets, il est recommandé d'utiliser et implémenter votre propre interface *EventListener*. Si vous aimez utiliser *EventBroadcaster*, voici comment procédé. La figure 5.1 à la page 14 montre un exemple simple de la gestion des événements avec *as2lib*. Mis à part la classe SimpleEventListener, il n'y a que des classes d'*as2lib* qui sont utilisés.

```
import org.as2lib.env.event.EventListener;
import org.as2lib.core.BasicClass;

class com.myproject.SimpleEventListener
  extends BasicClass implements EventListener {

  public function onTest(){
    trace("onTest");
  }
}
```

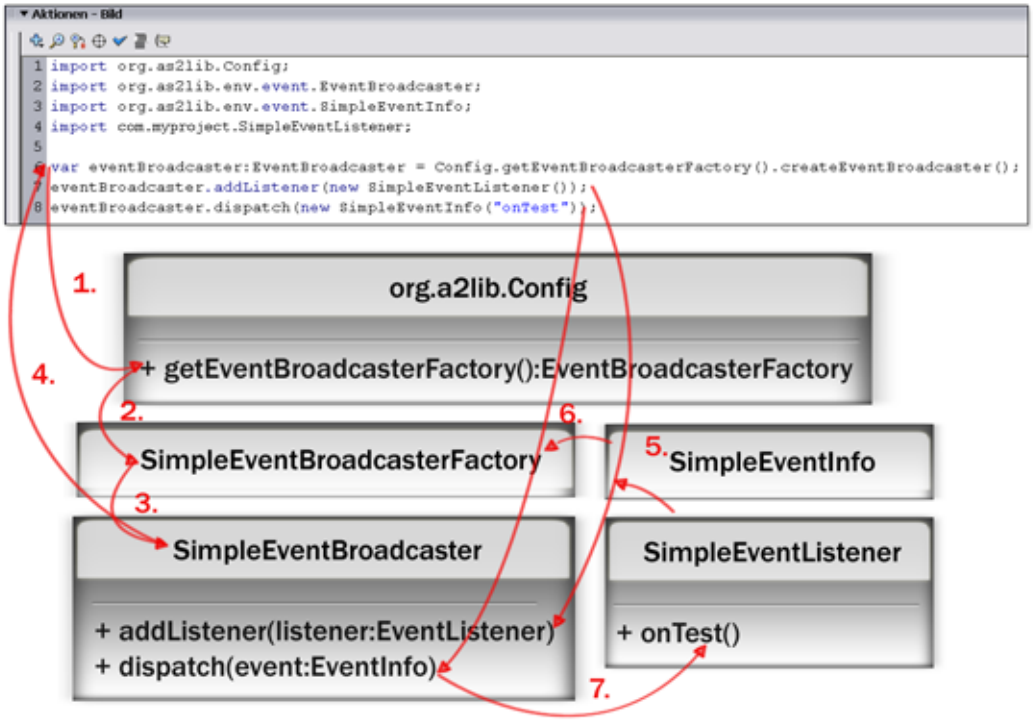

FIG. 5.1 – Exemple du procédé de la gestion des événements avec *as2lib*.

Chapitre 6

Reflections

Motivation : En *Java*, il est possible d'obtenir de l'information à propos d'une classe : son nom, méthodes et propriétés via Relfections. Cette fonctionnalité par défaut de Java n'est pas intégré en *ActionScript 2* et devrait l'être par *as2lib*.

Solution : Afin d'utiliser ces fonctionnalités dans *Flash*, les *Reflections* ont été implémentés d'après le schéma suivant. Les classes ActionScript sont recherchées à partir du "root package" *_global*. Si un objet est trouvé, il est reconnu en tant que package. Les sous-objets de type function sont marqués comme des classes.

Un usage parmi plusieurs pour *Reflections* avec *as2lib* est par exemple *BasicClass*, voir chapitre 2. La méthode *getClass* de *BasicClass* utilise la méthode *getClassInfo* de la classe *ReflectUtil*[1] et retourne une instance de *ClassInfo* qui fournit toutes les informations importantes d'une classe. Le package *reflect*[2] utilise différents algorithmes pour obtenir l'information d'une classe. La collection qui contient toutes les algorithmes est située dans le package algorythm de *as2lib*. Cette fonctionnalité peut, bien sûr, être accédé directement par la classe *ReflectUtil*.

```
import org.as2lib.env.util.ReflectUtil;
import org.as2lib.env.reflect.ClassInfo;
import edu.test.TestClass;

var aClass:TestClass = new TestClass();
var aClassInfo:ClassInfo = ReflectUtil.getClassInfo(
        aClass);
```

L'instance de *ClassInfo* qui a été créé contient des méthodes qui permettent

[1] org.as2lib.env.util.ReflectUtil
[2] org.as2lib.env.reflect

au développeur d'obtenir d'autre information à propos de la classe. Ses méthodes sont :

- **getName**() :*String* - Nom de la classe ex : "'TestClass"'.
- **getFullName**() :*String* - Nom complet de la classe incluant le package ex : "'edu.test.TestClass"'.
- **getRepresentedClass**() :*Function* - Référence vers la classe dont on a obtenu l'information.
- **getConstructor**() : *ConstructorInfo* - Retourne le constructeur de la classe entouré dans un instance de la classe *ConstructorInfo*.
- **getSuperClass**() :*ClassInfo* - Information à propos de la classe supérieure, si elle existe.
- **newInstance**() - Crée une nouvelle instance de la classe dont on a obtenu l'information.
- **getParent**() :*PackageInfo* - Information à propos du package dans lequel la classe est située.
- **getMethods**() :*Map* - Une Map, voir chapitre 7 , qui contient de l'information a propos de chaque méthode de la classe.
- **getMethod(methodName :String)** :*MethodInfo* - Retourne l'information à propos de la méthode dont le nom à été passé en paramètre sous la forme d'une instance de la classe MethodInfo.
- **getMethod(concreteMethod :Function)** :*MethodInfo* - Retourne l'information à propos de la méthode qui a été passé en paramètre sous la forme d'une instance de la classe MethodInfo.
- **getProperties**() :*HashMap* - - HashMap contenant l'information à propos des propriétés de la classe qui ont été défini en tant que setters et getters.
- **getProperty(propertyName :String)** :*PropertyInfo* - Retourne l'information à propos de la propriété dont le nom à été passé en paramètre sous la forme d'une instance de la classe PropertyInfo.
- **getProperty(concreteProperty :Function)** :*PropertyInfo* - Retourne l'information à propos de la propriété qui a été passé en paramètre sous la forme d'une instance de la classe PropertyInfo.

La connexion entre les classes Info démontré à l'aide de la figure 6.1 à la page 17.

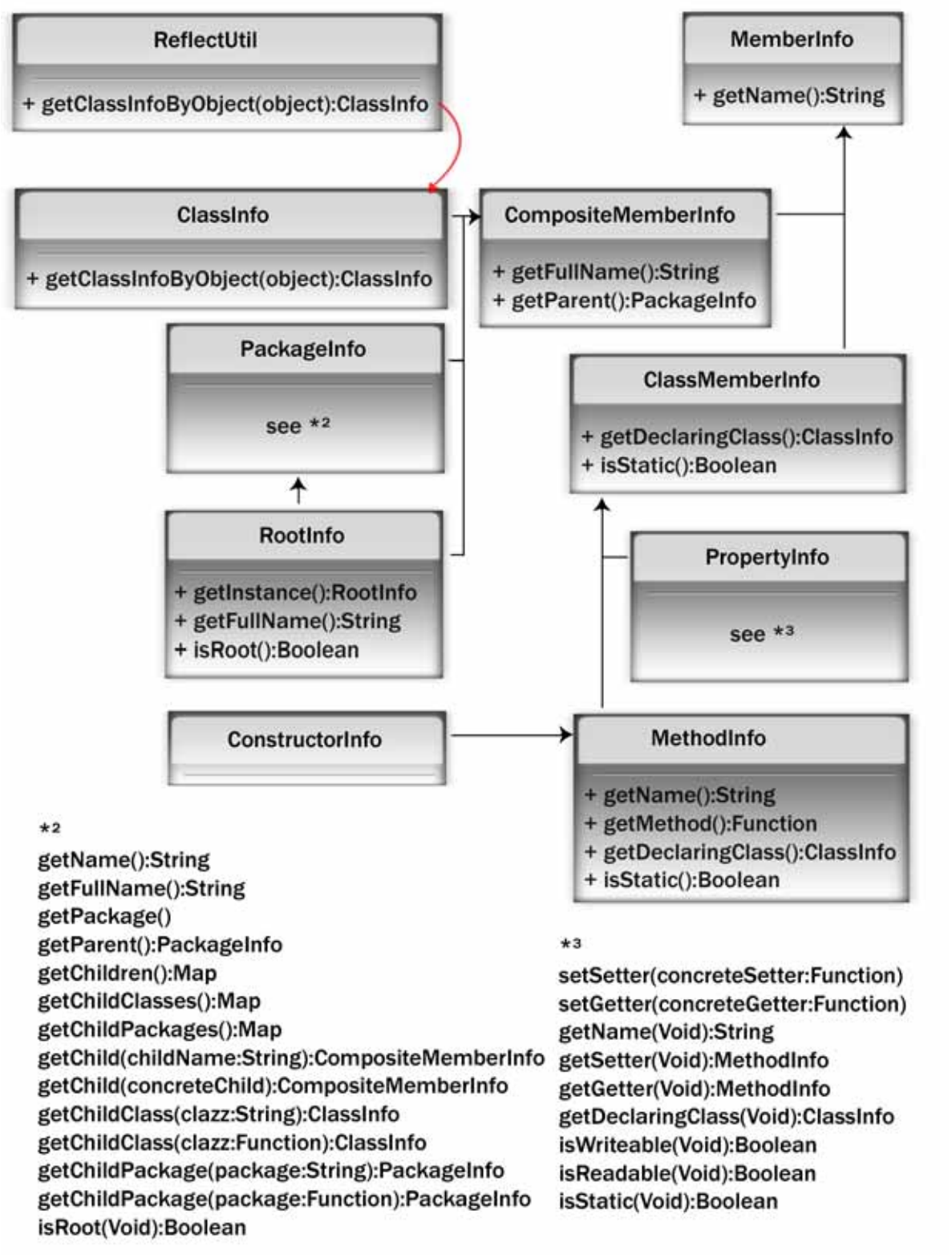

FIG. 6.1 – Hierarchy of the Info Classes in the *reflect* package

Chapitre 7

Data Holding

Motivation : Un problème typique de l'utilisation d'*ActionScript 2* est qu'il y a des types de données (ex : Array) qui peuvent contenir différents types de données (ex : String, Number,...). Cette notation flexible peut résulté, spécifiquement en équipe, en une mauvaise utilisation de ces collections de données.

Solution : *As2lib* ne fait pas que fournir une classe Array à typage fort (TypedArray[1]), mais aussi beaucoup d'autre types de collections.

TypedArray : La classe TypedArray fournit le typage fort à un Array.

```
import org.as2lib.data.holder.TypedArray;

var myA:TypedArray = new TypedArray(Number);
myA.push(2);
myA.push("Allo");
```

Dans ce bout de code, un Array de type Number est créé. Si vous essayez d'ajouter un String dans le TypedArray (*myA.push*("*Allo*")) le compilateur lancera l'exception suivante :

```
** FatalLevel **
org.as2lib.env.except.IllegalArgumentException:
Type mismatch between [Allo] and [[type Function]].
  at TypedArray.validate(Allo)
```

De plus, vous pouvez envoyer un Array déjà existant comme deuxième paramètre. La classe TypeArray fournit les mêmes fonctionnalités que la classe Array de *Macromedia*. Voici d'autre type de collection de *as2lib* :

[1] org.as2lib.data.holder.TypedArray

- **HashMap** : Un type de donnée qui contient des clés et leurs valeurs. Toutes ses méthodes sont communes avec un *HashMap* standard (voir Java).

```
import org.as2lib.data.holder.HashMap;

var aPerson:Person = new Person("Christoph",
  "Atteneder");
var bPerson:Person = new Person("Martin",
  "Heidegger");

var nickNames:HashMap = new HashMap();
nickNames.put(aPerson,"ripcurlx");
nickNames.put(bPerson,"mastaKaneda");

trace(nickNames.get(aPerson));
trace(nickNames.get(bPerson));
```

Output :
```
ripcurlx
mastaKaneda
```

- **Stack** : On peut ajouter des valeurs dans un Stack avec la méthode *push* et les enlever avec la méthode *pop*. On ne peut accéder qu'à la dernière valeur ajoutée.

```
import org.as2lib.data.holder.SimpleStack;

var myS:SimpleStack = new SimpleStack();
myS.push("uuuuuuup?!");
myS.push("what's");
myS.push("Hi");
trace(myS.pop());
trace(myS.pop());
trace(myS.pop());
```

Output :
```
Hi
what's
uuuuuuup?!
```

- **Queue** : En contraste avec le Stack, on ne peut accéder qu'à la première valeur ajoutée. On peut ajouter des valeurs avec la méthode *enqueue* et les enlever avec la méthode *dequeue*. Pour accéder à un élément sans l'enlever, il faut utiliser la méthode *peek*.

```
import org.as2lib.data.holder.LinearQueue;
```

```
var aLQ:LinearQueue = new LinearQueue();
aLQ.enqueue("Hi");
aLQ.enqueue("whats");
aLQ.enqueue("uuuuup?!");
trace(aLQ.peek());
trace(aLQ.dequeue());
trace(aLQ.dequeue());
trace(aLQ.dequeue());
```

Output :

```
Hi
Hi
whats
uuuuup?!
```

De plus il y a aussi des *Itérateurs*[2] fournit :

- **ArrayIterator** : À cause du fait que les autres itérateurs sont basés à l'interne sur les Array, tous les itérateurs utilisent indirectement un *ArrayIterator*[3].
- **MapIterator** : Si vous invoqué la méthode *iterator()* sur un *HashMap*, ça vous retourne un *MapIterator*.

Voici comment affiché tous éléments d'un HashMap à l'aide d'un *MapIterator*[4] :

[2] An iterator makes it easier to access elements of a collection without knowing its structure.

[3] org.as2lib.data.io.iterator.ArrayIterator

[4] org.as2lib.data.iterator.MapIterator

```
import org.as2lib.data.holder.HashMap;

var aPerson:Person = new Person("Christoph",
  "Atteneder");
var bPerson:Person = new Person("Martin",
  "Heidegger");

var nickNames:HashMap = new HashMap();
nickNames.put(aPerson,"ripcurlx");
nickNames.put(bPerson,"mastaKaneda");

var it:Iterator = myH.getIterator();

while(it.hasNext()){
  trace(it.next());
}
```

Output :

```
Christoph,Atteneder
Martin,Heidegger
```

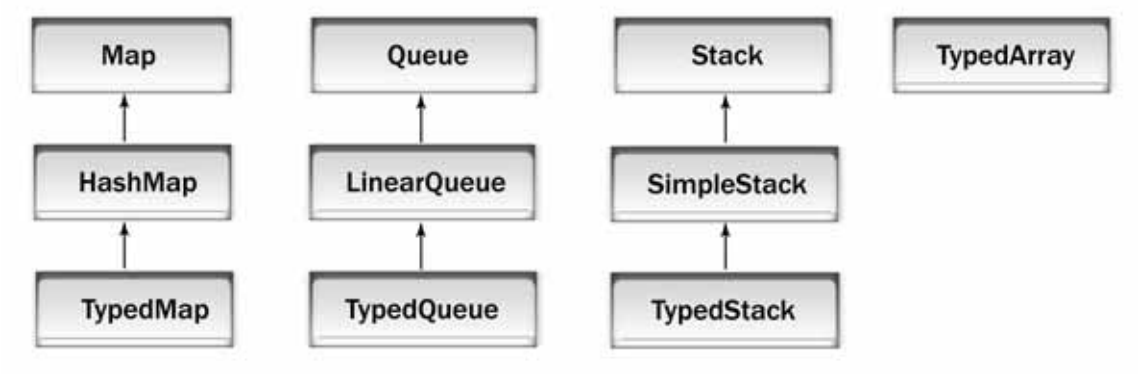

FIG. 7.1 – Les collections qui se trouvent dans le package *holder*.

La figure 7.1 à la page 21 montre la structure de le hiérarchie des collections qui se trouvent dans le package *holder*.

Chapitre 8

Overloading

Motivation : En Java, la surcharge de méthode est supportée, cela veut dire que la classe peut avoir un ou plusieurs constructeurs dont leurs seuls différences sont les paramètres qui leurs sont envoyés. Essayé d'implémenter plus d'un constructeur ou méthode avec le même nom mais des paramètres différents donne l'erreur suivante en *ActionScript 2* :

> "Une classe doit comporter un seul constructeur. " ou bien :
> "Impossible d'utiliser un nom de membre plusieurs fois."

Solution : *As2lib* rend la surcharge possible en *ActionScript 2* à l'aide du package *Overload*[1]. Si par exemple, une classe a besoin de 3 constructeurs, *as2lib* fournit une solution simple.

```
import org.as2lib.env.overload.Overload;

class TryOverload {
  private var aString:String;
  private var aNumber:Number;

  public function TryOverload(){
    var overload:Overload = new Overload(this);
    overload.addHandler([Number,
    String],
    setValues);
    overload.addHandler([Number],
    setNumber);
    overload.addHandler([String],
    setString);
    overload.forward(arguments);
  }
```

[1] org.as2lib.env.overload

```
    public function setValues(aNumber:Number,
    aString:String){
      this.aNumber = aNumber;
      this.aString = aString;
    }
    public function setNumber(aNumber:Number){
      this.aNumber = aNumber;
    }
    public function setString(aString:String){
      this.aString = aString;
    }
}
```

Dans la clase *TryOverload*, un constructeur est déclaré sans spécifier les paramètres de réception. Créé une instance de la classe *TryOverload* crée également un objet de type Overload qui reçoit des traiteurs pour chaque constructeur additionnels qui sont requis. Dans ce cas spécifique, il y a un constructeur pour Number et String, un constructeur pour un Number et un autre pour un String. Finalement les arguments sont envoyés à l'objet Overload qui invoque la méthode appropriée. Si les paramètres envoyés ne correspondent à aucun traiteur, une exception de type UnknownOverloadHandlerException est lancée.

Un test sur la classe *TryOverload* ressemble a ceci :

```
var aOverload:TryOverload = new TryOverload("Hallo");
var bOverload:TryOverload = new TryOverload(6);
var cOverload:TryOverload = new TryOverload(6,"y");
```

Chapitre 9

Test Cases

Un *Test Unitaire* est une classe/méthode qui vérifie une classe/méthode spécifique afin d'assurer que son comportement est correct. La plupart des développeurs doivent créer un test unitaire pour leurs propres classes et sont assisté par un API de test unitaire qui offre des fonctionnalités de test automatique.

Motivation : Bien qu'il existe déjà un API de test pour *ActionScript 2*, *as2unit*[1], notre API de test a été créée pour les raisons suivantes :

- *As2unit* - *As2unit* n'offre pas beaucoup de fonctionnalités (*as2unit* 7 méthodes de test - *as2lib* 15 méthodes de test).
- *As2unit* est dans le dépit de la révélation de leur code source, pas encore open source.
- La documentation officiel de *as2unit* n'est toujours pas disponible.
- *As2unit* est seulement disponible sous forme de component.
- *As2unit* ne peut tester qu'une seule classe à la fois.
- La sortie d'*as2unit* ne peut être fait qu'avec un trace.

Solution : Il y a deux actions qui ne devrait pas être nécessaire (incluant le component Flash, régler les paramètres) pour effectuer les test unitaire, comme s'est le cas présentement avec *as2unit*. Il devrait être possible pour le développeur d'effectuer un test unitaire à partir d'une invocation de méthode. Un appel direct de la classe autant qu'un appel direct à un package entier est possible.

[1] www.as2unit.org

```
import org.as2lib.test.unit.Test;
// Add your Tests here.
test.org.as2lib.core.TReflections;
Test.run("test.org");
```

Les unités à tester doivent être invoqué avant le début du test pour être disponible durant l'exécution. Similaire aux APIs de test unitaire comme : *JUnit*[2], une variétés de méthodes sont disponibles aux développeurs. (les paramètres optionnels sont marqué avec []) :

- **assertTrue**([*message :String*], *testVar1 :Boolean*) - Une erreur sera reporté si testVar1 est false.
- **assertFalse**([*message :String*], *testVar1 :Boolean*) - Une erreur sera reporté si testVar1 est true.
- **assertEquals**([*message :String*], *testVar1*, *testVar2*) - Une erreur sera reporté si les deux paramètre ne sont pas identique.
- **assertNotEquals**([*message :String*], *testVar1*, *testVar2*) - Une erreur sera reporté si les deux paramètre sont identiques.
- **assertNull**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 n'est pas null.
- **assertNotNull**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 est null.
- **assertUndefined**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 n'est pas undefined.
- **assertNotUndefined**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 est undefined.
- **assertIsEmpty**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 n'est pas undefined ou null.
- **assertIsNotEmpty**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 est undefined ou null.
- **assertInfinity**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 n'est pas équivalent à l'infini.
- **assertNotInfinity**([*message :String*], *testVar1*) - Une erreur sera reporté si testVar1 est équivalent à l'infini.
- **fail**(*message :String*) - Ajoute un message d'erreur personnalisé pour toutes les erreurs reportés.
- **assertThrows**(*exception :Function, atObject, theFunction :String, parameter :Array*) - S'il n'y pas d'exception de lancé durant l'exécution de la fonction envoyé (theFunction) sur l'objet (atObject) avec les paramètres(parameter), un erreur sera reporté.
- **assertNotThrows**(*exception :Function, atObject, theFunction :String, parameter :Array*) - Si une exception est lancé durant l'exécution de la fonction envoyé (theFunction) sur l'objet (atObject) avec les para-

[2] www.junit.org

mètres(parameter), un erreur sera reporté.

Un test unitaire doit hériter de la classe *Test*[3]. Toutes les méthodes du test dont le nom débute par "test" seront exécuté par l'API de test. Voir le code suivant pour mieux comprendre l'API de test unitaire :

```
import org.as2lib.test.unit.Test;
import org.as2lib.core.BasicClass;
import org.as2lib.env.reflect.ClassInfo;

class test.org.as2lib.core.TReflections extends Test {
  private var clazz:BasicClass;

  public function TReflections(Void) {
      clazz = new BasicClass();
  }

  public function testGetClass(Void):Void {
    trace ("::␣testGetClass");
    var info:ClassInfo = clazz.getClass();
    assertEquals(
    "The␣name␣of␣the␣basic␣class␣changed",
    info.getName(),
    "BasicClass");
    assertEquals(
    "Problems␣evaluating␣the␣full␣name",
    info.getFullName(),
    "org.as2lib.core.BasicClass");
    trace ("————————————————————");
  }
}
```

[3]org.as2lib.test.unit.Test

Chapitre 10

Speed Tests

Motivation : Tester les applications pour leur performance est inévitable parce que le succès d'une application dépend largement sur la performance.

Solution : On peut procéder à un test de vitesse en suivant le schéma suivant :

- Creating of single test cases. Example for MyAddSpeedTest :

```
import org.as2lib.test.speed.TestCase;

class MyAddSpeedTest implements TestCase {
  private var a:Number = 0;
  public function run (Void):Void {
    a++;
  }
}
```

- Importé les classes SpeedTest et OutputHandler.

```
import org.as2lib.test.speed.Test;
import org.as2lib.Config;
```

- Créer une instance de la classe Test et lui envoyé le OutputHandler.

```
var test:Test = new Test();
test.setOut(Config.getOut());
```

- Assigner le nombre de test à être exécuté.

```
test.setCalls(1000);
```

- Le test peut débuter après avoir ajouté les tests unitaires qui doivent être testés. La paramètre *true* dans *test.run()* déclanche la sortie immédiate des résultats du test.

```
test.addTestCase(new MyAddSpeedTest());
test.addTestCase(new MyMinusSpeedTest());
test.run(true);
```

Output :

```
** InfoLevel **
-- Testresult [2000 calls] --
187% TypedArrayTest: total time:457ms;
  average time:0.2285ms; (+0.106ms)
111% ArrayTest: total time:272ms;
  average time:0.136ms; (+0.014ms)
[fastest] 100% ASBroadcasterTest: total time:245ms;
  average time:0.1225ms;
175% EventDispatcherTest: total time:428ms;
  average time:0.214ms; (+0.092ms)
191% EventBroadcasterTest: total time:469ms;
  average time:0.2345ms; (+0.112ms)
```

La sortie du test de vitesse affiche le nombre d'appels, le temps en écoulé au total et le temps écoulé en moyenne. Il trouve le test le plus rapide et affiche sous forme de pourcentage l'efficacité de chaque test par rapport au plus rapide.

Chapitre 11

Preview

11.1 Connection Handling

Motivation : Les connexions avec des sources de données externes peuvent être fait de plusieurs façons avec *Flash MX 2004*. Elles peuvent être accédé via Flash Remoting, Web Services, XMLSocket. Le chargement du XML ou de fichier texte, une requête vers un URL (ex : CGI, PHP,:..) ou une connexion entre 2 SWF différent (LocalConnection) peuvent aussi être fait. L'implémentation de ces interfaces peut différer grandement, ce qui résulte en une implémentation individuelle de chaque interface. Même si les data components de *Flash MX Professional 2004* sont déjà fournis pour certaines interfaces (Web Services et fichier XML), les paramètre qui leur sont envoyés sont différents. Près du fait que vous devez posséder la version professionnelle, il n'existe pas de solution purement en ActionScript 2 qui supporte tous les interfaces, et que leurs utilisations soient identique.

Solution : *As2lib* fournit une interface standard pour chaque type de connexion. Toutes les connexions sont basées sur un *Proxy*, en contraste avec le proxy régulier de *Flash Remoting*, à un typage fort et permet de valider à la compilation.

11.2 as2lib console

11.2.1 Requirements

Durant le développement d'un application Flash, on peut utilisé la console et le débuggeur. Mais lorsque l'application est déployée sur le serveur web et qu'il survient une erreur, il est très difficile d'identifier la source de l'erreur. Cette circonstance coûte du temps et de l'argent. Il devrait être possible d'afficher l'erreur et le statut indépendamment de l'environnement de développement. A cause de cela, as2lib fournit une console externe qui offre cette

fonctionnalité.

Voici les fonctionnalités de la console d'*as2lib* :

- Affichage de la gestion de sortie, voir chapitre 3 page 5, devrait être possible à l'intérieur de l'environnement de développement ainsi que pour les application en ligne.
- Affichage et débugage des connexions vers les sources de données externes.
- Affichage et débugage des événements.
- Affichage de tous les objets de l'application sous forme d'arbre.
- Informations détaillé de chaque MovieClip.
- Utilisation de la RAM pour chaque élément.

11.2.2 The as2lib console

Dans le premier prototype de la console *as2lib* nous utilisons la gestion de sortie d'*as2lib*, voir chapitre 3 page 5, et la gestion des connexions, voir section 11.1 à la page 30. La première fonctionnalité de base qui est supporté dans la *console as2lib* est montrée dans la figure 11.1 à la page 31. Le prototype supporte le sortie (ex : *Out.debug()*, *Out.info()*,:..) de tous les applications Flash qui lui y sont connecté à la console. L'application flash peut se trouvé dans le navigateur web autant que dans un IDE.

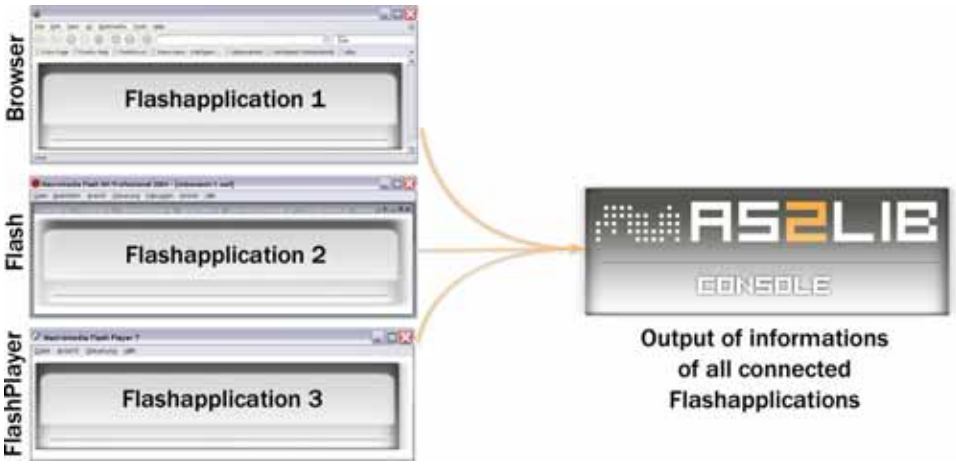

FIG. 11.1 – Usage of the *as2lib console*

En partant de ces conditions, une ébauche expérimentale a été fait et fournit les fonctionnalités souhaitées (voir figure 11.2 à la page 32). La console consiste de différents onglets pour afficher la sortie de chaque *OutLevel*. L'onglet *All* affiche toutes les sorties.

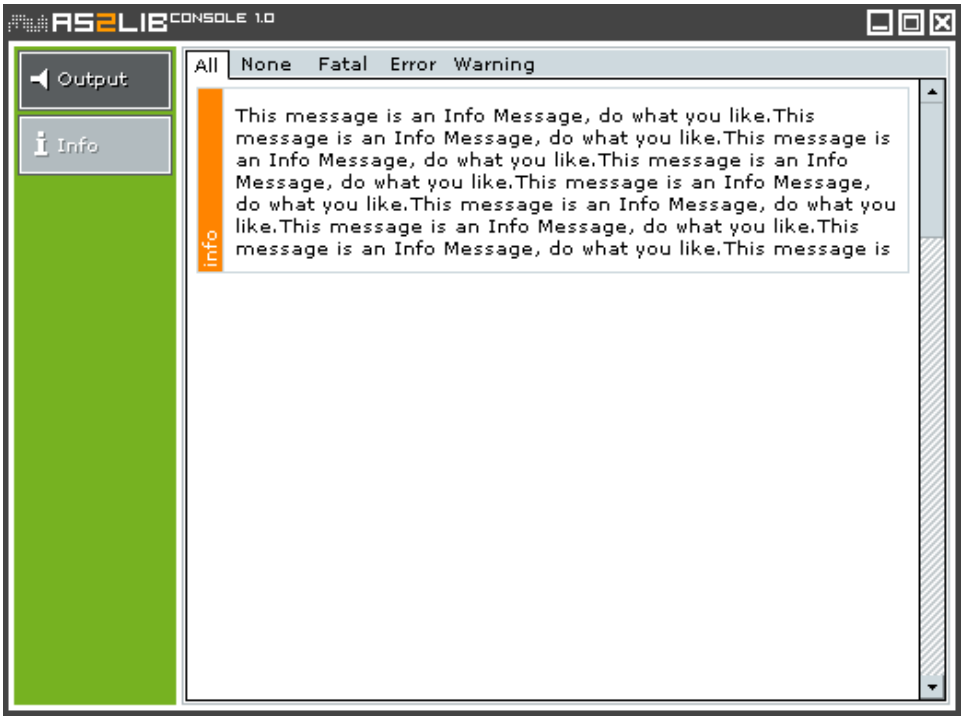

FIG. 11.2 – Utilisation de la console *as2lib*

11.3 Roadmap

see www.as2lib.org/roadmap.php