as2lib - eine kleine Einführung

Christoph Atteneder, Martin Heidegger,
Michael Herrmann, Alexander Schliebner und Simon Wacker

24. Mai 2004

Inhaltsverzeichnis

Kapitel 1

as2lib - Enstehungsgeschichte

Abbildung 1.1: www.as2lib.org

Die *as2lib*, eine Open Source ActionScript 2 Library, wurde September 2003 als Projekt zur Schaffung besserer Programmiermöglichkeiten unter ActionScript 2 ins Leben gerufen. Das Projektteam setzt sich mit den Kernproblemen von Flash auseinander und versucht gezielt die täglichen Probleme beim Programmieren mit Flash zu beheben. Als wichtiges Merkmal lässt sich hervorheben, dass die *as2lib* unter der MPL (Mozilla Public License) veröffentlicht wird. Dies bedeutet freie Benutzung in privaten, wie kommerziellen Projekten. Nach der Festlegung der Programmierrichtlinien und dem Grundkonzept, wurde die Arbeit der einzelnen Packages von fünf Projektmitarbeitern in Angriff genommen(siehe Tab. 1.1).

Name	*Aufgabe*	*Website*	*Nationalität*
Atteneder Christoph	Entwickler	www.cubeworx.com	Austria
Heidegger Martin	Projektleitung	www.traumwandler.com	Austria
Herrmann Michael	Entwickler		Austria
Schliebner Alexander	Mitinitiator	www.schliebner.de	Germany
Wacker Simon	Chef Entwickler	www.simonwacker.com	Germany

Tabelle 1.1: Aktive Mitglieder *as2lib*

Kapitel 2

Core Package

Beweggrund: Das Festlegen und zur Verfügung stellen gewisser Funktionalitäten in allen Klassen vereinfacht die Entwicklung und Fehlerbehebung während der Entwicklung.

Lösungsansatz: Alle Klassen, Interfaces und Packages der *as2lib* unterliegen gewissen Vorgaben. Die wichtigsten Kernklassen befinden sich im ***core***-*Package*.

2.1 BasicInterface

Zur besseren Definition von Klassenfunktionalitäten wird in der *as2lib* intensiv von *Interfaces* Gebrauch gemacht. Jedes erstellte Interface der *as2lib* erweitert das *BasicInterface*, um folgende Funktionalität in jeder *as2lib* Klasse sicherzustellen:

- **getClass()**:*ClassInfo* - Diese Methode liefert genauere Informationen zur Klasse, in der diese Funktion aufgerufen wird. Die zurückgegebene Information ist vom Typ *ClassInfo* und beinhaltet zusätzlich deren Klassennamen, Informationen wie Methoden, Eigenschaften, Klassenpfad und Superklasse.
- **toString()**:*String* - Diese Methode gibt einen String der die Klasse repräsentiert zurück.

Die Logik der *getClass* Methode wird in der *BasicClass* Klasse zur Verfügung gestellt (siehe Abb. 2.1, S. 4).

2.2 BasicClass

Die Grundklasse der *as2lib* ist die *BasicClass* Klasse. Alle Klassen der *as2lib* sind direkt oder indirekt von der *BasicClass* Klasse abgeleitet. Sie imple-

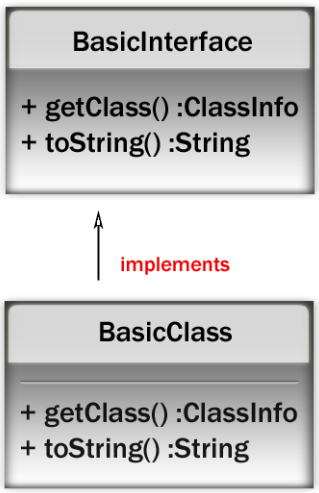

Abbildung 2.1: Hauptbestandteile des *core* Packages

mentiert das *BasicInterface* und stellt über die *ReflectUtil*[1] Klasse und die *ObjectUtil*[2] Klasse die Logik für folgende Methoden zur Verfügung:

- **getClass()**:*ClassInfo* - Erklärung siehe BasicInterface. Das Erstellen der Klasseninformationen wird durch das *as2lib reflection* Package, siehe 6, ermöglicht.
- **toString()**:*String* - Diese Methode liefert eine Darstellung der Klasse, vom Typ String, zurück.

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

Kapitel 3

Output Handling

Beweggrund: Die normale Ausgabe von Applikationen in Flash wird über den internen Befehl

```
trace(ausdruck);
```

durchgeführt. Die trace Ausgabe ist jedoch nur in Entwicklungsumgebungen möglich, die den Befehl auch unterstützen. Bei allen anderen Fällen (z.B.: in einer Webapplikation) ist keine Standardausgabe definiert. Eine Library sollte eine standardisierte Ausgabe sowohl für den User als auch für den Entwickler zu Verfügung stellen und überall möglich sein.

Lösungsansatz: Um in jeder Laufzeitumgebung[1] eine oder mehrere Ausgabemöglichkeiten zu haben, wird die *Out*[2] Klasse verwendet. So kann z.B.: eine Webapplikation Fehlermeldungen auch serverseitig abspeichern, um sicherzustellen das Fehler sich nicht nur beim Kunden bemerkbar machen. Die *Out* Klasse behandelt alle eingehenden Anfragen („Ausgabeversuche“) und leitet sie in Abhängigkeit von der Konfiguration an einen, oder mehrere *OutputHandler*[3] weiter. Die *as2lib* bietet eine standardisierte Ausgabemöglichkeit für beliebig viele Schnittstellen.

Anwendung: Ein einfacher Anwendungsfall des *as2lib* Output Handling wird in Abbildung 3, S. 6 dargestellt. Nachdem eine Instanz der *Out* Klasse erstellt und der zur Verfügung stehende TraceHandler[4] hinzugefügt wurde kann eine Ausgabe erfolgen. Der Ablauf der einzelnen Aktionen entspricht der Reihenfolge ihrer Nummerierung.

[1] Flash kann im Player, in *Macromedia Central*, in einer kompilierten Applikation(*.exe) laufen

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

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

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

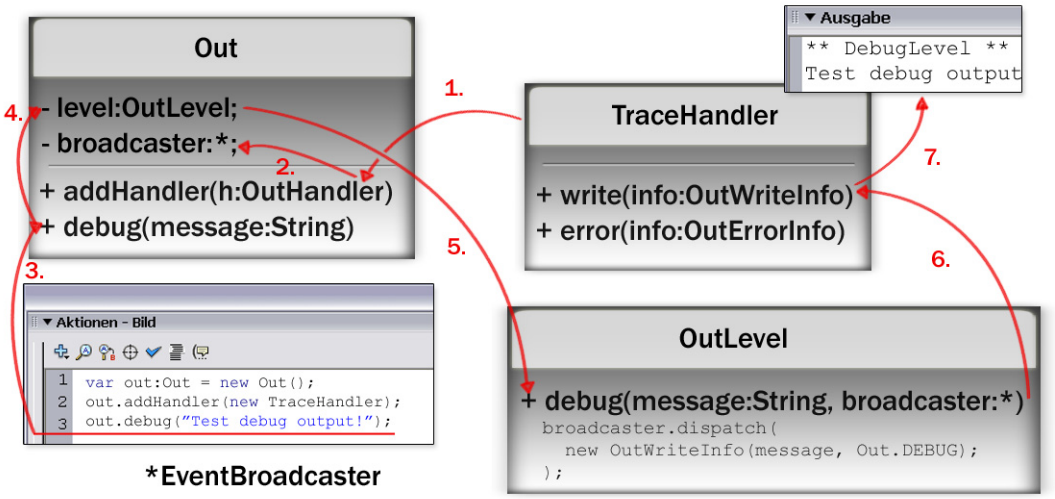

Abbildung 3.1: Anwendungsfall des *as2lib* Output Handlings

Es kann wie in Abbildung 3, S. 6 auf bereits definierte Ausgaben wie den *TraceHandler* oder *ExternalConsoleHandler*[5] zugegriffen oder ein eigener *OutHandler* erstellt (z.B.: eine Ausgabe über die *Macromedia* Alert Komponente) werden[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());
  }
```

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

[6] Die Macromedia Alert Komponente muss sich in der Bibliothek befinden und Sie müssen Flash MX 2004 Professional besitzen, um über die Alert Klasse darauf zugreifen zu können.

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

Durch die Festlegung von Ausgabe Levels(z.B.: *aOut.setLevel(Out.DEBUG)*) kann die Ausgabe bestimmter Informationen verhindert werden und unterschiedliche Ausgabe Levels verwenden. Die möglichen Ausgabe Levels sind:

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

Diese Abstufung ermöglicht einerseits ein übersichtlicheres Debuggen bei der Entwicklung, als auch eine schnelleres Umschreiben der Ausgabe in einer fertigen Applikation. Verwendet man bei der Entwicklung z.B.: *Out.DEBUG* werden bei dieser Konfiguration alle Informationen die sich in einem niedrigeren Level, als das gesetzte(*DEBUG*), befinden ausgegeben:

- *DEBUG*
- *INFO*
- *WARNING*
- *ERROR*
- *FATAL*

Nur die *LOG* Ausgabe wird unterdrückt.

```
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));
```

Möchte man in der fertigen Applikation nur die schwerwiegenden Fehler ausgeben, lässt sich das durch eine einfache Zeile in der Applikation bewerkstelligen.

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

Kapitel 4

Exception Handling

Beweggrund: Nicht abgefangene Fehlermeldungen werden von der Entwicklungsumgebung mit

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

ausgegeben[1]. Neben der Tatsache, dass die ausgegebene Information nur wenig bis gar nicht aufschlussreich ist (Es wird nur „Error“ bzw. der dem Konstruktor übergebene String ausgegeben. Siehe Abb. 4, S. 9) ist eine Anzeige der Fehlermeldungen nur in *Flash MX 2004* möglich.

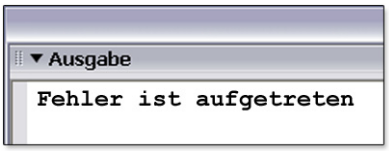

Abbildung 4.1: Ausgabe eines Errors(*throw new Error(„Fehler ist aufgetreten“);*) in Flash MX 2004.

Lösungsansatz: In der *as2lib* stehen, basierend auf der *Macromedia* internen *Error* Klasse unterschiedliche *Exception* Klassen zur Verfügung, die die Methoden des *Throwable*[2] Interfaces implementieren. Die neuen Funktionalitäten des Exception Handlings sind:

- Alle Operationen die aufgerufen werden, bevor die Exception geworfen wurde, werden in einem Stack abgespeichert, um die Fehlersuche zu beschleunigen.

[1] Online Dokumentation unter livedocs.macromedia.com/flash/mx2004/main/12_as217.htm
[2] org.as2lib.env.except.Throwable

- Es ist keine manuelle Eingabe der Fehlerbezeichnung mehr notwendig. Vor der as2lib:

```
class MyError extends Error {
  public var name = "MyError";
}
```

  Mit Hilfe von *Reflections*, siehe Abschnitt 6, wird der Name der Fehlermeldung in der *as2lib* automatisch generiert.

- Exceptions können einfach in andere Exceptions umgewandelt werden.

Folgene Vordefinierte Exception Klassen stehen zur Verfügung:

- *AbstractException* : Alle Exception Klassen werden von der AbstractException Klasse abgeleitet. Sie implementiert die Methoden, die das Throwable Interface definiert (siehe Abb. 4, S. 11).

- *Throwable* : Ist ein Interface, dass die Implementierung folgender Methoden erzwingt:

  - **getStackTrace(Void):***Stack* - Gibt einen Stack aller Operationen zurück, die aufgerufen wurden, bevor diese Exception geworfen wurde.
  - **initCause(cause:Throwable):***Throwable* - Gibt den Grund der Exception an und kann nur einmal gesetzt werden. Die Methode wird normalerweise verwendet wenn eine Exception in eine andere Exception umgewandelt und weitergeworfen wird, um keine Information zu verlieren.
  - **getCause(Void):***Throwable* - Gibt den Grund der Exception zurück.
  - **getMessage(Void):***String* - Gibt die Nachricht, die bei der Erstellung des Exception Objektes im Konstruktor übergeben wurde, zurück.

- *Exception* : Ist eine Standard-Implementierung des Throwable Interfaces und wird von der AbstractException Klasse abgeleitet.

- *FatalException* : Zum Unterschied einer Exception Implementierung, hat eine FatalException eine höhere Priorität bzw. ein höheres Level als eine normale Exception, das abgefragen werden kann.

- In der *as2lib* sind bereits einige Exceptions definiert:

  - IllegalArgumentException
  - IllegalStateException
  - UndefinedPropertyException
  - ...

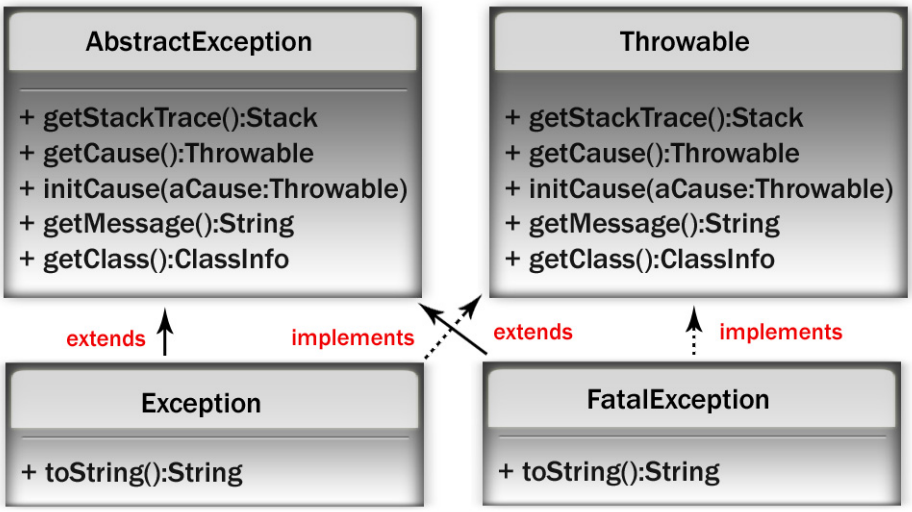

Abbildung 4.2: Grundlegende Hierarchie des *as2lib Exception* Packages

Anwendung: Bei einer fehlerhaften Parameterübergabe kann mit folgendem Code eine IllegalArgumentException geworfen werden:

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

Bei allen Exceptions sind drei Übergabeparameter notwendig:

1. message:String zB.: „Falscher Parameter“- Ein beliebiger Text, der den Grund der Exception deutlicher hervorbringen soll.

2. thrower zB.: „this“ - Referenz auf die Klasse bzw. das Objekt, dass die Exception geworfen hat.

3. arguments - ist eine intrinsische Variable von Flash, die alle Parameter einer Funktion beinhaltet

Eine geworfene IllegalArgumentException kann im ausführenden Code mit einem *try-catch* Block abgefangen wird.

```
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);
}
```

Es können beliebig viele eigene Exception definiert werden. Möchte man zB.: eine OutOfTimeException werfen, muss man folgende Implementierung vornehmen:

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

/**
 * @see org.as2lib.env.except.Exception
 */
class org.as2lib.env.except.OutOfTimeException
  extends Exception {
/**
 * @see org.as2lib.env.except.Exception#Constructor()
*/
public function OutOfTimeException(message:String,
  thrower, args:FunctionArguments) {
  super(message, thrower, args);
}
}
```

Es wird die Exception Klasse abgeleitet und der Konstruktor der Exception Klasse aufgerufen.

Kapitel 5

Event Handling

Beweggrund: Ereignisse(Events) brauchen in *Flash* sehr viel Rechenleistung und sind ein grundlegender Bestandteil bei der Entwicklung von Benutzeroberflächen. Viele Entwickler verwenden den in *Flash* inkludierten *AsBroadcaster*[1] oder die *EventDispatcher*[2] Klasse von *Macromedia* für diese Problematik. Jedoch gibt es weder genaue Spezifikationen für *EventListener* noch für einzelne Events und es sind keine Übergabewerte definiert. Die Festlegung von Schnittstellen über Interfaces ist für eine „saubere“ Programmierung unumgänglich. Die Art und Weise wie *Macromedia* den *EventDispatcher* implementiert hat, kann nur als „lack of Definition“ bezeichnet werden. Es fehlen unter anderem für Listenerentwickler wichtige Informationen bezüglich Events und der Übergabeparameter.

Lösungsansatz: Zu bewältigende Probleme sind:

- Der Objektentwickler muss definieren, welche Events abgewartet werden können.
- Der Listenerentwickler (Listener: Objekt das auf ein Event wartet) muss alle Events definieren.
- Der Objektentwickler muss die Möglichkeit haben, Events auszulösen.
- Der Objektentwickler sollte Eventinformationen einem Event hinzufügen können.

Die *as2lib* unterstützt *Event Handling*, da es ein Kernstück der Applikationsentwicklung ist. Verwendet man direkt die Flash internen *AsBroadcaster* kann es zu einer ineffizienten Implementierung kommen. Der *EventDispatcher* von *Macromedia* ist nicht frei zugänglich, kann also nur verwendet

[1] Nicht dokumentiertes Feature von Flash MX 2004
[2] mx.events.EventDispatcher

werden, wenn man *Macromedia Flash* käuflich erworben hat und stellt nicht alle benötigten Funktionen zur Verfügung.

Das wichtigste Interface des *event* Packages mit der man als Entwickler in Berührung kommt ist der *EventBroadcaster*[3]. Es können mehrere Listener(Zuhörer) zum EventBroadcaster(Ereignissverteiler) hinzugefügt,

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

aber auch wieder entfernt werden.

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

Wie man in den Methodenaufrufen erkennen kann müssen die Listener Objekte das EventListener Interface implementieren. Für eigene Projekte empfiehlt es sich ein eigenes EventListener Interface zu erstellen.

Möchte man den *EventBroadcaster* verwenden kann man seine Funktionalitäten einfach nach folgender Methode erwerben. Die Abbildung 5 auf Seite 15 zeigte eine einfach Anwendung des *as2lib* Event Handlings. Es werden bis auf den SimpleEventListener nur Klassen der *as2lib* verwendet.

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

class edu.diplomarbeit.listener.SimpleEventListener
  extends BasicClass implements EventListener {

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

Der Ablauf wird in Abbildung 5 auf Seite 15 durch die Nummerierung der Ereignisse klarer.

[3] org.as2lib.env.event.EventBroadcaster

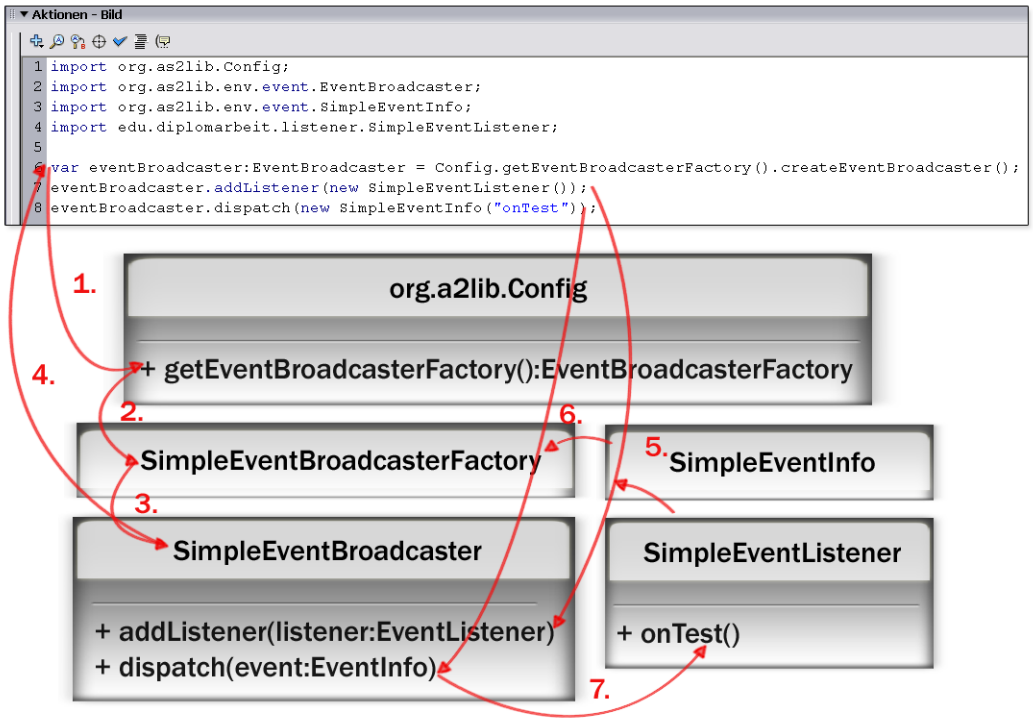

Abbildung 5.1: Ablauf eines einfachen Event Handling Beispiels

Kapitel 6

Reflections

Beweggrund: In Java ist es möglich Informationen über den Namen der Klasse, deren Methoden und Eigenschaften mittels Reflections zu ermitteln. Diese eingebaute Funktionalität von Java ist in *Flash* bzw. *Actionscript 2* nicht integriert und wird deshalb von der *as2lib* unterstützt.

Lösungsansatz: Um diese Funktionalitäten in *Flash* verwenden zu können wurden die Reflections nach folgendem Schema implementiert. Ausgehend vom Namensbereich *_global* wird die Actionscript Klassenstruktur durchsucht. Wird ein Objekt gefunden, wird es als Package erkannt. Ein Unterobjekt vom Typ Function wird als Klasse gekennzeichntet.

Eine der vielen Anwendungen von Reflections in der *as2lib* ist zB.: die BasicClass, siehe Abschnitt 2. Die Methode *getClass*, der BasicClass greift auf die Methode *getClassInfo* der ReflectUtil[1] Klasse zu und gibt ein *ClassInfo* Objekt zurück, dass alle wichtigen Informationen der Klasse zur Verfügung stellt. Das *reflect*[2] Package arbeitet mit unterschiedlichen Algorithmen um an die Klasseninformationen zu gelangen. Die Sammlung aller Algorithmen befindet sich im *algorythm*[3] Package der *as2lib*. Die Funktionaliäten der Reflections können natürlich auch direkt über die *ReflectUtil* Klasse verwendet werden.

```
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);
```

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

Das erzeugte Objekt vom Typ *ClassInfo* beinhaltet Methoden um genauere Informationen der Klasse zu erhalten. Seine Methoden sind:

- **getName**():*String* - Name der Klasse z.B.: „TestClass“.
- **getFullName()**:*String* - Name der Klasse inklusive Packageinformationen z.B.: „edu.test.TestClass“.
- **getRepresentedClass()**:*Function* - Klasse zu der die Klasseninformation erstellt wurde.
- **getConstructor()**: *ConstructorInfo* - Gibt Konstruktor der Klasse als ConstructorInfo zurück.
- **getSuperClass**():*ClassInfo* - Informationen zur Vaterklasse, falls vorhanden.
- **newInstance**() - Neue Instanz der Klasse zu der die Klasseninformation erstellt wurde.
- **getParent()**:*PackageInfo* - Informationen zum Package in dem sich die Klasse befindet.
- **getMethods()**:*Map* - Map, siehe 7, die Informationen zu den einzelnen Methoden der Klasse beinhaltet.
- **getMethod(methodName:String)**:*MethodInfo* - Gibt Information zur Methode dessen Name übergeben wurde als MethodInfo zurück.
- **getMethod(concreteMethod:Function)**:*MethodInfo* - Gibt Information zur Methode die übergeben wurde als MethodInfo zurück.
- **getProperties()**:*HashMap* - HashMap, die Informationen zu den einzelnen Eigenschaften, falls sie eine get set Methode besitzen zurückliefert.
- **getProperty(propertyName:String)**:*PropertyInfo* - Gibt Information zur Eigenschaft dessen Name übergeben wurde als PropertyInfo zurück.
- **getProperty(concreteProperty:Function)**:*PropertyInfo* - Gibt Information zur Eigenschaft die übergeben wurde als PropertyInfo zurück.

Die Verknüpfungen der einzelnen Info Klassen zeigt die Abbildung 6 auf Seite 18.

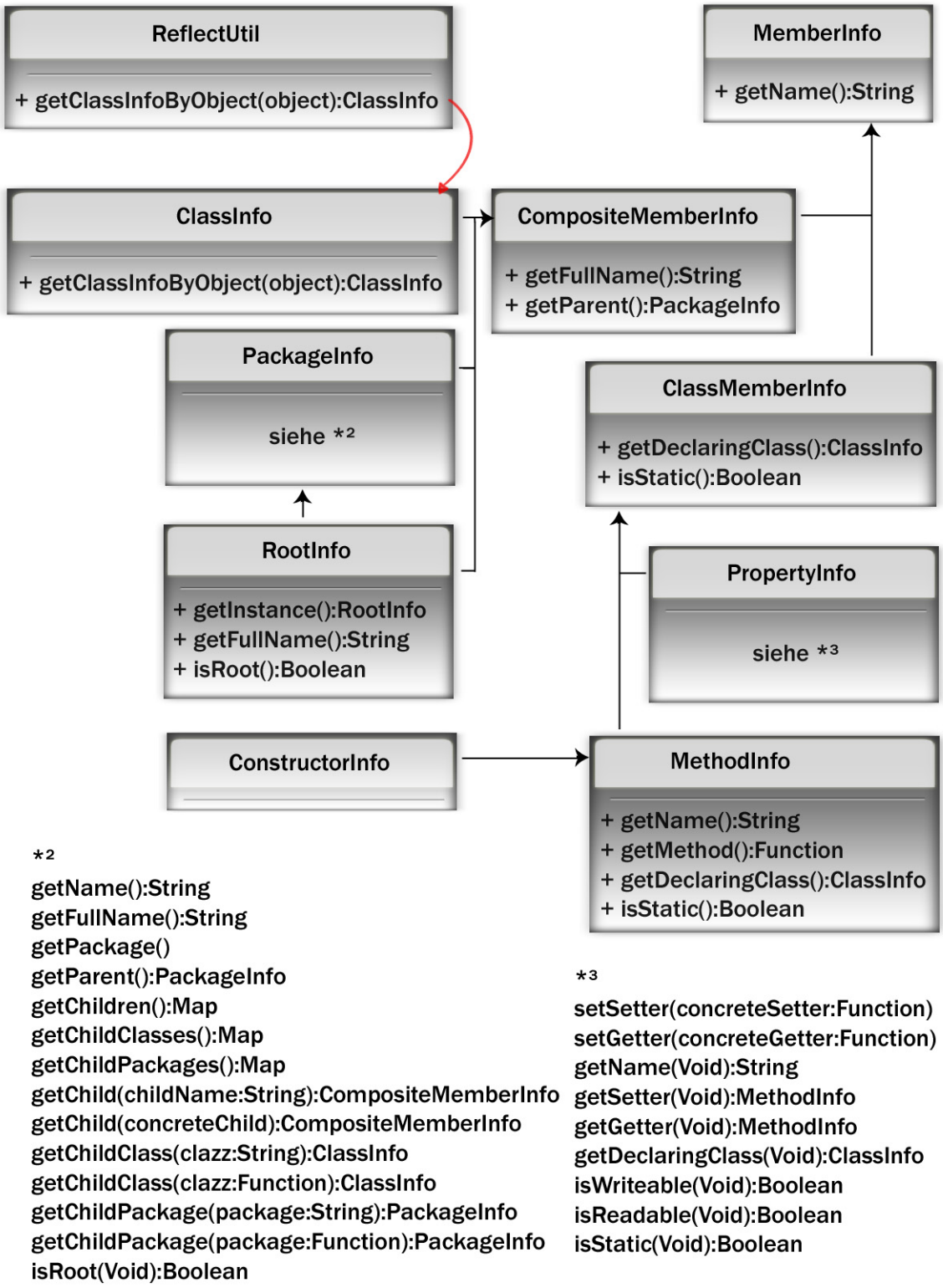

Abbildung 6.1: Hierarchie der Info Klassen im *reflect* Package

Kapitel 7

Data Holding

Beweggrund: Ein typisches Problem beim Gebrauch von *Actionscript 2* ist das einige Datentypen (zB.: Array) unterschiedliche Datentypen (zB.: String und Number) beinhalten können. Diese nicht strikte Notation kann vor allem bei einer Arbeit in Teams leicht zu Problemen führen.

Lösungsansatz: In der *as2lib* gibt es nicht nur eine Array Klasse(TypedArray[1]), die nur bestimmte Datentypen zulässt, sondern auch eine Vielzahl anderer Datentypen zur Datenhaltung.

TypedArray: Die Klasse TypedArray erlaubt eine strikte Datentypisierung eines Arrays.

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

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

In dem Codebeispiel wird ein Array vom Typ Number erstellt. Versucht ein Entwickler einen String zum TypedArray hinzuzufügen(*myA.push(„Hallo“)*) wirft der Compiler einen Fehler. Zusätzlich zum Typ des Arrays kann dem TypedArray Konstruktor ein bereits bestehendes Array als zweiter Übergabeparameter mit übergeben werden. TypedArray besitzt die gleichen Funktionaliäten wie die normale Array Klasse von *Macromedia*.

Weitere *as2lib* Datentypen sind:

- **HashMap** : Ein Datentyp dem ein Key und der dazugehörige Wert übergeben werden kann. Er besitzt alle Methoden einer normalen HashMap(siehe Java).

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

```
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("ripcurlx", aPerson);
nickNames.put("mastaKaneda", bPerson);

trace(nickNames.get("mastaKaneda").toString());
trace(nickNames.get("ripcurlx").toString());
```

- **Stack** : Einem Stack können mit der Methode *push* Werte hinzugefügt bzw. mit *pop* wieder entfernt werden. Es kann immer nur auf das oberste Element zugegriffen werden.

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

var myS:Stack = new Stack();
myS.push("gehts?!");
myS.push("wie");
myS.push("Hallo,");
trace(myS.pop());
trace(myS.pop());
trace(myS.pop());
```

- **Queue** : Einer Queue können mit der Methode *enqueue* Werte hinzugefügt bzw. mit *dequeue* wieder entfernt werden. Es kann immer nur auf das erste Element zugegriffen werden. Mit der Methode *peek* kann zusätzlich auf das erste Element zugegriffen werden, ohne es aus der Reihe zu entfernen

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

var aLQ:LinearQueue = new LinearQueue();
aLQ.enqueue(1);
aLQ.enqueue(2);
aLQ.enqueue(3);
trace(aLQ.peek());
trace(aLQ.dequeue());
trace(aLQ.dequeue());
trace(aLQ.dequeue());
```

Zusätzlich zu den Datentypen wurden auch *Iteratoren*[2] implementiert:

- **ArrayIterator** : Da die zusätzliche *as2lib* Datentypen intern mit Arrays aufgebaut sind, wird bei allen speziellen Iteratoren indirekt der ArrayIterator verwendet.
- **MapIterator** : Wenn die Methode getIterator() oder iterator() in einer HashMap aufgerufen wird, wird automatisch ein MapIterator zurückgeworfen.

Möchte man zB.: alle Elemente einer HashMap ausgeben, kann diese Aufgabe mit einem MapIterator leicht gelöst werden.

[2]Ein Iterator ermöglicht den Zugriff auf die Elemente einer Sammlung ohne Kenntnis der Struktur der Sammlung.

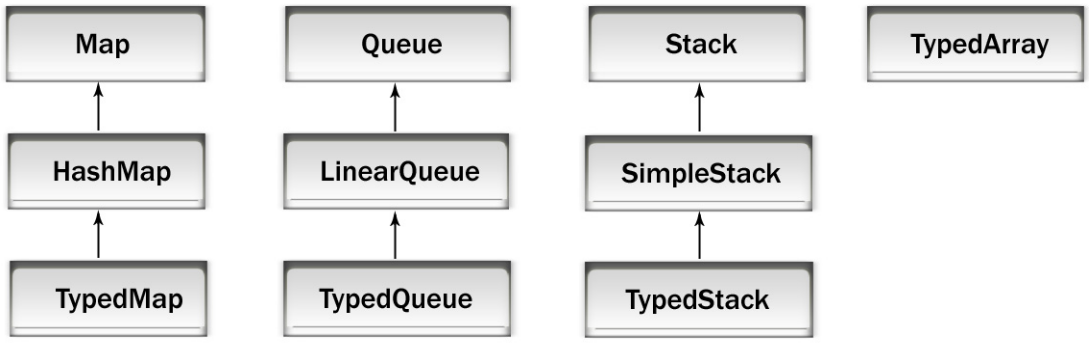

Abbildung 7.1: Datenhalter des *holder* Package

```
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("ripcurlx", aPerson);
nickNames.put("mastaKaneda", bPerson);

var myI:Iterator = myH.getIterator();

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

Die Abbildung 7 auf Seite 22 zeigt eine hierarchische Auflistung der Datenhalter des *holder* Packages.

Kapitel 8

Overloading

Beweggrund: Da Überladen von Methoden in Java unterstützt wird können zB.: in einer Klasse zwei Konstruktoren vorhanden sind, die sich nur durch in ihren Übergabeparameter unterscheiden. Versucht man in *Actionscript 2* mehrere Konstruktoren oder Methoden mit dem gleichen Namen und unterschiedlichen Übergabeparametern zu implementieren wird folgende Fehlermeldung ausgegeben:

> „Eine Klasse kann nur einen Konstruktor haben!“
> bzw.
> „Ein Mitgliedsname darf nur einmal vorkommen!“

Lösungsansatz: Die *as2lib* ermöglicht das Overloading in Actionscript 2 durch das *overload*[1] Package. Benötigt man zB.: drei Konstruktoren in einer Klasse kann dieses Problem mit der *as2lib* einfach gelöst werden.

```
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);
```

[1] org.as2lib.env.overload

```
    overload.forward(arguments);
  }
  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;
  }
}
```

Es wird in der Klasse TryOverload ein Konstruktor erstellt, der keine bestimmten Übergabeparameter definiert. Wird eine Instanz der Klasse TryOverload erstellt, wird eine Overload Objekt zusätzlich erstellt, dem soviele Handler übergeben werden, wie man zusätzliche Konstruktoren haben möchte. In dem speziellen Fall gibt es einen Konstruktor für Number und String, einen Konstruktor für eine Number und einen Konstruktor für einen String. Schlussendlich werden die übergebenen Argumente(arguments) dem Overload Objekt übergeben, der die entsprechende Funktion aufruft. Ein Test der TryOverload Klasse würde wie folgt aussehen:

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

Kapitel 9

Test Cases

Ein *Test Case*(Testfall) ist eine Klasse/Methode, die eine bestimmte Klasse/Methode auf ihre korrekte Funktionsweise überprüft. Jeder Entwickler muss für seine Klassen Testfälle erstellen und wird von einem Test Case System unterstützt, das Funktionalitäten für ein automatisches Testen zur Verfügung stellt.

Beweggrund: Obwohl bereits ein Testsystem für *Actionscript 2* Klassen existiert, as2unit[1], wurde aus folgenden Gründen ein eigenes Testsystem erstellt:

- *as2unit* unterstützt nur wenige Funktionalitäten(as2unit 7 Testmethoden - as2lib 15 Testmethoden).
- *as2unit* ist trotz verkündeter Offenlegung des Quellcodes immer noch nicht Open Source.
- Es gibt immer noch keine offizielle Dokumentation von *as2unit*.
- *as2unit* ist nur als Komponente vorhanden.
- *as2unit* kann immer nur eine Klasse testen.
- *as2unit* erlaubt eine Ausgabe nur über trace.

Lösungsansatz: Um Testfälle durchführen zu können, sollen nicht zwei unterschiedliche Aktionen notwendig sein(Einbinden der Flashkomponente, setzen der Parameter), wie es momentan bei der *as2unit* der Fall ist. Es soll dem Entwickler möglich sein einen Testfall durch einen einfachen Methodenaufruf durchführen zu können. Sowohl ein direkter Aufruf einer Klasse, als auch ein Aufruf eines gesamten Packages, falls mehrere Test Fälle durchzuführen sind, ist möglich.

[1] www.as2lib.org

```
import org.as2lib.test.unit.Test;

// Add here your Tests.
test.org.as2lib.core.TReflections;
test.org.as2lib.util.TReflectUtil;
test.org.as2lib.util.TStringUtil;

Test.run("test.org");
```

Die zu testenden Fälle müssen vor dem Testaufruf angeführt werden, damit sie zur Laufzeit verfügbar sind. Ähnlich wie bei Test Case Systemen zB.: JUnit[2] stehen dem Entwickler in der *as2lib* bereits eine Vielzahl von Methoden zu Verfügung(optionale Parameter sind durch [] gekennzeichnet):

- **assertTrue**([*message:String*], *testVar1:Boolean*) - Ist testVar1 false wird eine Fehlermeldung ausgegeben.
- **assertFalse**([*message:String*], *testVar1:Boolean*) - Ist testVar1 true wird eine Fehlermeldung ausgegeben.
- **assertEquals**([*message:String*], *testVar1*, *testVar2*) - Sind die übergebenen Parameter nicht gleich wird eine Fehlermeldung ausgegeben.
- **assertNotEquals**([*message:String*], *testVar1*, *testVar2*) - Sind die übergebenen Parameter gleich wird eine Fehlermeldung ausgegeben.
- **assertNull**([*message:String*], *testVar1*) - Ist testVar1 nicht null wird eine Fehlermeldung ausgegeben.
- **assertNotNull**([*message:String*], *testVar1*) - Ist testVar1 gleich null wird eine Fehlermeldung ausgegeben.
- **assertUndefined**([*message:String*], *testVar1*) - Ist testVar1 nicht undefined wird eine Fehlermeldung ausgegeben.
- **assertNotUndefined**([*message:String*], *testVar1*) - Ist testVar1 gleich undefined wird eine Fehlermeldung ausgegeben.
- **assertIsEmpty**([*message:String*], *testVar1*) - Ist testVar1 weder undefined noch null wird eine Fehlermeldung ausgegeben.
- **assertIsNotEmpty**([*message:String*], *testVar1*) - Ist testVar1 undefined oder null wird eine Fehlermeldung ausgegeben.
- **assertInifinity**([*message:String*], *testVar1*) - Ist testVar1 nicht unendlich(d.h. != Infinity) wird eine Fehlermeldung ausgegeben.

[2] http://www.junit.org

- **assertNotInfinity**([*message:String*], *testVar1*) - Ist testVar1 unendlich(d.h. == Infinity) wird eine Fehlermeldung ausgegeben.
- **fail**(*message:String*) - Fügt eine selbst definierte Fehlermeldung zu allen Error-Ausgaben hinzu.
- **assertThrows**(*exception:Function, atObject, theFunction:String, parameter:Array*) - Wird beim Ausführen der übergebenen Funktion(theFunction) des Objektes(atObjekt) mit den Übergabeparametern(parameter) keine Exception geworfen wird eine Fehlermeldung ausgegeben.
- **assertNotThrows**(*exception:Function, atObject, theFunction:String, parameter:Array*) - Wird beim Ausführen der übergebenen Funktion(theFunction) des Objektes(atObjekt) mit den Übergabeparametern(parameter) eine Exception geworfen wird eine Fehlermeldung ausgegeben.

Ein Testfall muss von der Test Klasse[3] abgeleitet werden. Jede Methode des Testfalles, die mit „test“ beginnt, wird von dem Testsystem aufgerufen. Um das TestCase System besser verstehen zu können soll dieses Codebeispiel dienen:

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

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

/**
 * Testcase for Reflections.
 * @author Martin Heidegger
 */

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 ("----------------------");
  }
}
```

Kapitel 10

Speed Tests

Beweggrund: Das Testen von Applikationen auf dessen Performanz ist unumgänglich, da diese meistens direkt mit dem Erfolg der Applikation zusammenhängt.

Lösungsansatz: Die Durchführung eines Speed Testes lässt sich nach folgendem Schema durchführen:

- Importieren der SpeedTest Klasse und des Output Handlers.

```
import org.as2lib.env.out.Out;
import org.as2lib.test.speed.Test;
```

- Erstellen einer Instanz der Test Klasse und Übergabe des Output Handlers.

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

- Setzen der Anzahl der durchzuführenden Test Fälle.

```
test.setCalls(2000);
```

- Nachdem die durchzuführenden Testfälle hinzugefügt worden sind kann der Test durchgeführt werden. Der Übergabeparameter *true* in *test.run()* bewirkt eine sofortig Ausgabe des Testergebnisses.

```
test.addTestCase(new TypedArrayTest());
test.addTestCase(new ArrayTest());
test.addTestCase(new ASBroadcasterTest());
test.addTestCase(new EventDispatcherTest());
test.addTestCase(new EventBroadcasterTest());
test.run(true);
```

Ausgabe:

```
** 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)
```

Die Ausgabe des SpeedTests zeigt die Anzahl der Aufrufe, die verstrichene Gesamtzeit und den durchschnittlichen Zeitbedarf. Sie findet den schnellsten Testfall und gibt das prozentuelle Verhalten der anderen Testfälle im Verhältnis zum Schnellsten aus.

Kapitel 11

Ausblick

11.1 Connection Handling

Beweggrund: Verbindungen mit externen Datenquellen kann in *Flash MX 2004* auf verschiedene Art und Weise erstellt werden. Externe Datenquellen können zB.: über *Flash Remoting*, *Web Services*, *XML Socket Connection* angesprochen werden. Aber auch XML- oder Text-Dateien, Anfragen über die URL(zB.: CGI, PHP,...) oder auch Verbindungen zwischen unterschiedlichen SWF-Dateien(LocalConnection) können durchgeführt werden. Die Implementierung dieser unterschiedlichen Datenschnittstellen unterscheidet sich jedoch teilweise massiv und muss für jede Datenquelle speziell durchgeführt werden. Zwar bestehen in *Flash MX 2004 Professional* für bestimmte Datenquellen Verbindungskomponenten(Web Services und XML Dateien), jedoch können sie nur von Flash MX Professional Besitzern verwendet werden und unterscheiden sich in ihren Übergabeparametern. Reine *Actionscript 2* Lösungen bestehen nicht für alle Datenschnittstellen und es kann auch keine standardmäßige Implementierung für alle Datenquellen durchgeführt werden.

Lösungsansatz: Die *as2lib* stellt eine standardisierte Datenschnittstelle zur Verfügung, die alle Datenschnittstellen beinhaltet.

11.2 as2lib Debug Konsole

11.2.1 Anforderungen

Bei der Entwicklung von Flash Applikationen steht dem Entwickler nur die Konsolenausgabe zur Verfügung. Wird jedoch die Applikation auf einem Webserver veröffentlicht und es treten in dieser Phase Probleme auf ist es nur sehr schwer möglich schnell die Fehlerquelle zu ermitteln. Dieser Umstand kostet Zeit und Geld. Es sollte eine zusätzliche Ausgabe von Fehler- und

Statusmeldungen außerhalb der Enwicklungsumgebung möglich sein. Aus diesem Grund stellt die *as2lib* eine externe Konsole zur Verfügung die diese Funktionalitäten bietet. Die Anforderungen an die *as2lib console* sind:

- Ausgabe des *as2lib* Output Handling, siehe Abschnitt 3 auf Seite 5 soll sowohl in der Entwicklungsumgebung als auch in online Flashapplikationen möglich sein.
- Anzeige und Debugging von Verbindungen zu externen Datenquellen einer Flashapplikation.
- Anzeige und Debugging von Events in einer Flashapplikation
- Darstellung aller Objekte einer Flashapplikation in Baumform
- Informationen zu einzelnen MovieClips
- Speicherverbrauch einzelner Elemente

11.2.2 Modellierung einer Applikation mit der *as2lib*

Eine Modellierung mit der *as2lib* führt zu stabileren und besser wartbaren und vor allem schnelleren Ergebnissen. Es können sowohl einzelne Funktionalitäten als auch die gesamte Funktionalität der *as2lib* genutzt werden.

11.2.3 Die as2lib console

Im ersten Prototypen der *as2lib console* wird vor allem das *as2lib* Output Handling, siehe Abschnitt 3 auf Seite 5, und Connection Handling, siehe Abschnitt 11.1 auf Seite 31 verwendet.

Die grundlegende Funktionalität der *as2lib* Debug Konsole wird in Abbildung 11.2.3 auf Seite 33 dargestellt. Im Prototypen werden Debugausgaben(zB.: *Out.debug()*, *Out.info()*,...) mehrerer Flashapplikationen die sich zur Debug Konsole verbinden können in der Konsole ausgegeben. Die Flashapplikationen können sich im Browser im Flash Player oder auch in der Flash Entwicklungsumgebung befinden.

Ausgehend von diesen Anforderungen wurde ein erster grafischer Entwurf erstellt, der die erste geforderte Funktionalität beinhaltet(siehe Abb. 11.2.3, S. 33). Die Konsole besteht aus mehreren Karteireiter(Tabs) in denen jeweils nur die Informationen ausgegeben werden die das jeweilige OutLevel besitzen. Nur im Karteireiter All werden alle Informationen dargestellt.

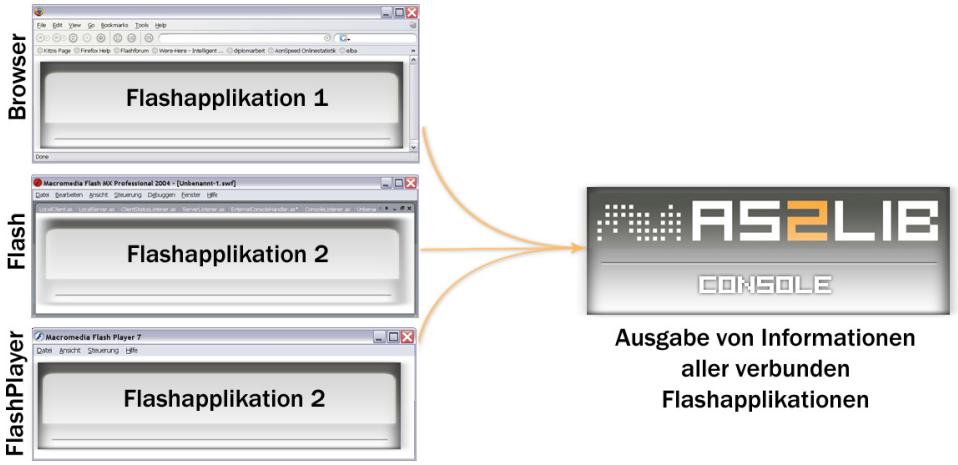

Abbildung 11.1: Verwendung der *as2lib console*

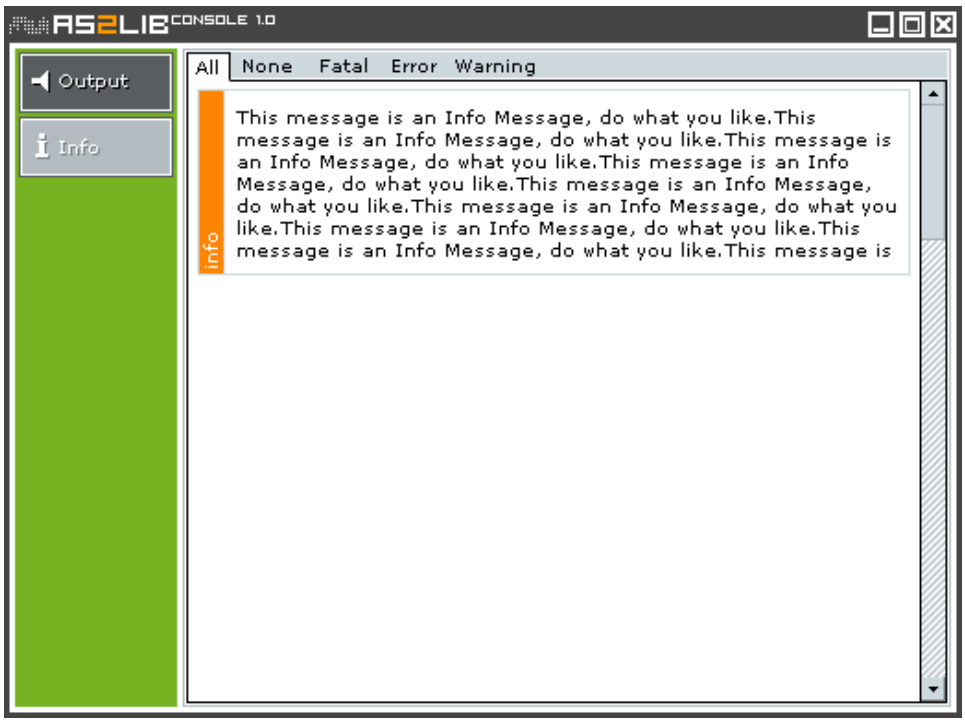

Abbildung 11.2: Grafischer Entwurf der *as2lib* Debug Konsole