Coworker,
An Inter Process Communication framework for AS3 Workers

Hugues Sansen, Shankaa
hugues@shankaa.com

Introduction

As many AS3 developers we missed the multi-threading capabilities for some complex processing. A few years ago, in June 2010, we wrote a never published paper describing how we were using the Flash inter process communication architecture that has been there since almost the beginning : the LocalConnection. We had developed what we called a PseudoThread object that could handle outbound and inbound LocalConnections. In this framework we had to connect to hidden ancillary applications. This technology was mostly used in a mail reader to help the treatment of certain files in a mailer project of our own, Shankaalipi. We sent this paper to Flex's team at Adobe in June 2011. Whether they read it or not, they came back with a solution that was inspired by the same idea but with a better performance: the Worker architecture. This architecture works correctly even if real multi-threading would be better. However, we have to use what we have now, not what doesn't exist. Our current application, Visumag [2], is complex as well as computing intensive. To make the code more straight forward and flexible we have developed an IPC framework on top of the current Worker framework. This framework is slightly inspired from Android IPC but without the IPC scripting language capability. In this paper we discuss the concept in an example : the building of a spreadsheet after loading an xls or xlsx file with the as3xls API.

About Visumag computation needs

Visumag is a tool used to visualize in 3D the performances of retail stores. It must be able to be used either on line or off line. When off line all, computations cannot be done on a server. In order not to freeze the display when doing computation we need to delegate those computation to ancillary processes. Intensive computation intervene at different levels of the code. We also need to receive processing events from remote processes as well as the final results. To make things more straight forward we need to send messages to local objects that are connected to remote objects. We use a Stub/proxy architecture.

About ActionScript Worker

The AS worker architecture is a way to describe how to run an independent Flash code in its own process and how to communicate with it. Most of the time the Worker is described in the project. The mechanism generates a “swf” file for this worker that will run in its own process.

The DataWorker

The DataWorker is our generic implementation of a worker. Since we have to register all the class aliases that will be used in our application, we subclass the DataWorker class to include the registerClassAliases. In our example this subClass is ExcelDataWorker. “ ExcelDataWorker” will be the class set as ActionScript Worker. This worker is limited to the use of Excel files as described in the as3xls API.

```
package org.shankaa.coworkers
{
	import flash.net.registerClassAlias;

	import org.as3.ms.xls.Cell;
	import org.as3.ms.xls.ExcelFile;
	import org.as3.ms.xls.Sheet;

	public class ExcelDataWorker extends DataWorker
	{
		public function ExcelDataWorker()
		{
			super();
		}

		protected override function initialize():void
		{
			// register all the classes you need to pass from the main worker to
			// this worker
			registerClassAlias("org.as3.ms.xls.ExcelFile", ExcelFile);
			registerClassAlias("org.as3.ms.xls.Sheet", Sheet);
			registerClassAlias("org.as3.ms.xls.Cell", Cell);
		}
	}
}
```

The DataWorker stays generic and will run with all the classes that can be marshaled and unmarshaled.
For classes that cannot be marshaled by the standard AS3 mechanism mostly classes for which a builder requires arguments we have added our own mechanism that is often application dependent as shown in the following code.

```
package org.shankaa.coworkers.stubs
{
	import mx.collections.ArrayCollection;
	import mx.collections.ISort;
	import mx.collections.XMLListCollection;

	/**
	 *
	 * @author Hugues Sansen, Shankaa
	 * The MainToWorkerUtils class is part of the coworkers framework.
	 *
	 * we centralize the marshalling and unmarshalling of classes that do not marshall
automatically

	 *
	 * Coworker is provided under the European Union Public Licence (EUPL)
	 */
	public class MainToWorkerUtils
	{

		public static function xmlListToArray(xList:XMLList):Array{
			if(xList == null){
				return null;
			}
			var arxl     : Array = new Array();
			var data     : Array = new Array();
			var header   : Array = [];
			arxl[0] = header;
			arxl[1] = data;
			for each(var x:XML in xList){
				data.push(x);
			}
			return arxl;
		}

		public static function
xMLListCollectionToArray(coll:XMLListCollection):Array{
			if(coll == null){
				return null;
			}
			var array          : Array = [];
			var data           : Array = [];
			var sort           : ISort = coll.sort ;
			array[0] = sort;
			array[1] = data;
			for each(var x:XML in coll){
				data.push(x);
			}
			return array;
		}

		public static function arrayCollectionToArray(coll:ArrayCollection):Array{
			if(coll == null){
				return null;
			}
			var array          : Array = [];
			var data           : Array = [];
			var sort           : ISort = coll.sort ;
```

```
                array[0] = sort;
                for each (var o:* in coll){
                        data.push(o);
                }
                array[1] = data;

                return array;
        }



        public static function arrayToXMLList(arxl:Array):XMLList{
                if(arxl == null){
                        return null;
                }
                var xList    : XMLList = new XMLList();
                for each(var x:XML in arxl[1]){
                        xList += x;
                }
                return xList;
        }

        public static function
arrayToXMLListCollection(arxl:Array):XMLListCollection{
                if(arxl == null){
                        return null;
                }
                var xList          : XMLList = new XMLList();
                var sort           : ISort = null;//arxl[0];
                for each(var x:XML in arxl[1]){
                        xList += x;
                }
                var collection     : XMLListCollection       = new
XMLListCollection(xList);
                if(sort){
                        collection.sort = sort;
                        collection.refresh();
                }
                return collection;
        }

        public static function arrayToArrayCollection(array:Array):ArrayCollection{
                if(array == null){
                        return null;
                }
                var xList          : XMLList = new XMLList();
                var sort           : ISort = null;//array[0];
                var collection     : ArrayCollection   = new ArrayCollection(array[1]);
                if(sort){
                        collection.sort = sort;
                        collection.refresh();
                }
                return collection;
        }

        public static function marshallComplexObject(value:*):*{
                var passedValue : * = value;
                if(value is XMLList){
```

```
                        passedValue = MainToWorkerUtils.xmlListToArray(value as
XMLList);
                  } else if (value is XMLListCollection){
                        passedValue = MainToWorkerUtils.xMLListCollectionToArray(value
as XMLListCollection);
                  } else if(value is ArrayCollection){
                        passedValue = MainToWorkerUtils.arrayCollectionToArray(value as
ArrayCollection);
                  }
                  return passedValue;
            }

            public static function unmarshallComplexObject(value:*,targetClass:Class):*{
                  var passedValue : * = value;
                  if(value is Array){
                        if(targetClass == XMLList){
                              passedValue = arrayToXMLList(value);
                        } else if(targetClass == XMLListCollection){
                              passedValue = arrayToXMLListCollection(value);
                        } else if(targetClass == ArrayCollection){
                              passedValue = arrayToArrayCollection(value);
                        }
                  }
                  return passedValue;
            }
      }
}
```

By default, the DataWorker registers only one main[1] process to remote[2] process channel.
When created, a DataWorker object has the possibility to create new objects upon request from the main process. Those objects are stored in a dictionary. Each entry is a pointer that corresponds to a UUID created inside the main process by a Stub object.

Once created, the objects are addressed through messages.

In general methods that define messages that are inter-process include the following arguments :

```
replyChannel:MessageChannel
eventChannel:MessageChannel
args:Array=null
```

where:
the `replyChannel` is the channel that will return a result
the `eventChannel` is a channel exclusively used to send events created in the remote process back to the main process.

Then instead of having to write especial code inside the a Worker, except the class aliases registration (that we do in ExcelDataWorker in this example) , the DataWorker doesn't need to know what it will be used for. All the logics is written in the main process.

1 We use the terms “main process” for the process that requests the service of a DataWorker. In some cases both can be workers.
2 We use the terms “remote process” for the ancillary Worker. In some cases both can be workers.

```as3
package org.shankaa.coworkers
{
	import flash.display.Sprite;
	import flash.events.Event;
	import flash.net.getClassByAlias;
	import flash.system.MessageChannel;
	import flash.system.Worker;
	import flash.utils.Dictionary;

	import mx.utils.UIDUtil;

	import org.shankaa.coworkers.stubs.InWorker_Proxy;
	import org.shankaa.coworkers.stubs.MainToWorkerUtils;

	/**
	 *
	 * @author Hugues Sansen, Shankaa
	 *
	 * Coworker is provided under the European Union Public Licence (EUPL)
	 *
	 */
	public class DataWorker extends Sprite
	{

		private var localObjects                    : Dictionary = new Dictionary();
		private var requestChannel                          : MessageChannel;

		private var _name                                   : String =
			UIDUtil.createUID();

		private var _requestChannelID               : String = "requestChannel";

		protected function set requestChannelID(value:String):void{
			_requestChannelID = value;
			initialize();
		}

		protected function get requestChannelID():String{
			return _requestChannelID;
		}


		public function DataWorker()
		{
			initialize();

		}

		/**
		 * should be overridden to register class aliases
		 *
		 * creates all the channels
		 *
		 */
		protected function initialize():void
		{

			try{
```

```
                        requestChannel      =
                  Worker.current.getSharedProperty("requestChannel") as MessageChannel;
                        requestChannel.addEventListener(Event.CHANNEL_MESSAGE,
                        onInputHandler);
                  } catch (error:Error){
                        trace(error.getStackTrace());
                  }

            }

            /**
             * received serialized objects must have a constructor without arguments.
             * only attributes or setters that are public can be passed by the  AMF3
unmarshalling mechanism
             *
             *
             * @param event
             *
             */

            private function onInputHandler(event:Event):void{
                  var request               : Array            =
requestChannel.receive();
                  var requestType           : String     = (request as Array).shift();
                  request[2]                                    =
Worker.current.getSharedProperty(request[2] as String) as MessageChannel;
                  request[3]                                    =
Worker.current.getSharedProperty(request[3] as String) as MessageChannel;
                  var i                     : int        = 4;
                  var local                 : *    ;
                  while(i<request.length){
                        var o : * = request[i];
                        try{
                              if( o is InWorker_Proxy){
                                    var uid : String = o.handle;
                                    request[i] = indexOnLocal(uid,o.path);
                              }
                        } catch (error:Error){

                        }
                        ++i;
                  }

                  switch(requestType){
                        case "workerCallback":
                              onCallback(request);
                              break;
                        case "getObjectByUuid":
                              onGetObjectByUuid(request);
                              break;
                        case "applyOnRemoteObject":
                              onApplyOnLocalObject(request);
                              break;
                        case "getRemoteObjectValue":
                              onGetRemoteObjectValue(request);
                              break;
```

```
                        case "setRemoteObjectValue":
                              onSetRemoteObjectValue(request);
                              break;
                  }
            }


            protected function
setName(replyChannel:MessageChannel,eventChannel:MessageChannel,name:String):void{
                  _name = name;
            }


            protected function
getName(replyChannel:MessageChannel,eventChannel:MessageChannel):void{
                  replyChannel.send(_name);
            }


            protected function
build_Object(className:String,replyChannel:MessageChannel,eventChannel:MessageChannel,args
:Array=null):void{
                  var uuid                  : String                  =
UIDUtil.createUID();
                  var objectClass           : Class                         =
getClassByAlias(className);

                  var funArgs               : Array                         = args ==
null?[]:args;
                  switch(funArgs.length){
                        case 0:
                              localObjects[uuid] = new objectClass();
                              break;
                        case 1:
                              localObjects[uuid] = new objectClass(funArgs[0]);
                              break;
                        case 2:
                              localObjects[uuid] = new
                                    objectClass(funArgs[0],funArgs[1]);
                              break;
                        case 3:
                              localObjects[uuid] = new
                                    objectClass(funArgs[0],funArgs[1],funArgs[2]);
                              break;
                        case 4:
                              localObjects[uuid] = new
                              objectClass(funArgs[0],funArgs[1],funArgs[2],funArgs[3]);
                              break;
                        case 5:
                              localObjects[uuid] = new
                                    objectClass(funArgs[0],funArgs[1],funArgs[2],
                                          funArgs[3],funArgs[4]);
                              break;
                        case 6:
                              localObjects[uuid] = new
                                    objectClass(funArgs[0],funArgs[1],funArgs[2],
                                          funArgs[3],funArgs[4],funArgs[5]);
                              break;
```

```
                        case 7:
                              localObjects[uuid] = new objectClass(funArgs[0],
                                    funArgs[1],funArgs[2],funArgs[3],funArgs[4],
                                    funArgs[5],funArgs[6]);
                              break;
                        case 8:
                              localObjects[uuid] = new objectClass(funArgs[0],
                                    funArgs[1],funArgs[2],funArgs[3],funArgs[4],
                                    funArgs[5],funArgs[6],funArgs[7]);
                              break;
                        case 9:
                              localObjects[uuid] = new objectClass(funArgs[0],
                                    funArgs[1],funArgs[2],funArgs[3],funArgs[4],
                                    funArgs[5],funArgs[6],funArgs[7],funArgs[8]);
                              break;
                        case 10:
                              localObjects[uuid] = new objectClass(funArgs[0],
                                    funArgs[1],funArgs[2],funArgs[3],funArgs[4],
                                    funArgs[5],funArgs[6],funArgs[7],funArgs[8],
                                    funArgs[9]);
                              break;
                  }

                  replyChannel.send(uuid);
            }

            /**
             * args : functionName, replyChannel, eventChannel, the rest of args to apply
to the function
             *
             * a generic way to call a method
             * @param args
             *
             */
            private function onCallback(args:Array):void{
                  var data     : Array      = args.concat();
                  var replyFunctionName     : String     = data.shift();
                  (this[replyFunctionName] as Function).apply(this,data);
            }

            private function onApplyOnLocalObject(args:Array):void{
                  var data     : Array =    args.concat();
                  var replyFunctionName     : String = data.shift();
                  var uuid     : String = data.shift() as String;
                  var object   : *;
                  if(uuid == "this"){
                        object = this;
                  } else {
                        object = localObjects[uuid];
                  }
                  (object[replyFunctionName] as Function).apply(object,data);
            }

            private function onGetObjectByUuid(args:Array):void{
                  var data     : Array      = args.concat();
                  var uuid     : String     = data.shift() as String;
                  var object   : *          = localObjects[uuid];
                  var replyChannelUid : String    = data.shift() as String;
```

```
		var replyChannel :  MessageChannel     =
			Worker.current.getSharedProperty(replyChannelUid);
		replyChannel.send(object);
	}

	private function indexOnLocal(knownIndex:String,accessToLocal:Array):*{
		var object   : *    = localObjects[knownIndex];
		var result   : *;
		switch(accessToLocal.length){
			case 0:
				result = object;
				break;
			case 1:
				result = object[accessToLocal[0][0]][accessToLocal[0][1]];
				break;
			case 2:
				result = object[accessToLocal[0][0]][accessToLocal[0][1]]
					[accessToLocal[0][2]];
				break;
		}

		return result;
	}

	private function onSetWorkerFieldValue(args:Array):void{
		var data : Array = args.concat();
		var functionName : String = data.shift() as String;
		// we shift the channels that are supposed to be null
		data.shift();
		data.shift();
		var value : * = data.shift() ;
		this[functionName] = value;
	}

	private function onGetWorkerFieldValueValue(args:Array):void{
		var data : Array = args.concat();
		var functionName : String  = data.shift() as String;
		var replyChannel : MessageChannel = data.shift() as MessageChannel;
		var result : * =
			MainToWorkerUtils.marshallComplexObject(this[functionName]);
		replyChannel.send(result);
	}

	private function onGetRemoteObjectValue(args:Array):void{
		var data : Array  = args.concat();
		var functionName : String = data.shift() as String;
		var uuid : String = data.shift() as String;
		var object : * ;
		if(uuid == "this"){
			object = this;
		} else {
			object = localObjects[uuid];
		}
		var replyChannel : MessageChannel = data.shift() as MessageChannel;
		var result : * =
			MainToWorkerUtils.marshallComplexObject(object[functionName]);
		replyChannel.send(result);
	}
```

```
		private function onSetRemoteObjectValue(args:Array):void{
			var data : Array  = args.concat();
			var functionName : String = data.shift() as String;
			var uuid : String = data.shift() as String;
			var object : * ;
			if(uuid == "this"){
				object = this;
			} else {
				object = localObjects[uuid];
			}
			// we shift the channels that are supposed to be null
			data.shift();
			data.shift();
			var value : * = data.shift() ;
			object[functionName] = value;
		}
	}
}
```

The Stub and its real object counterpart in the Worker

The Stub is an object created in the main process that communicates with a corresponding object, the real object, that lives in the remote process.
When a Stub is created, it requests the creation of its fully blown counterpart on the worker. And then communicates with it through blocking or non blocking methods.

Each real object that has to communicate with its stub must implement new methods with "InWorker" suffix, a reply channel and an event channel :

When a stub is defined by the following Interface describing the functions we want to access in its remote object :

```
package org.shankaa.coworkers.example
{
	public interface ICoworkingObjectExample
	{
		function set field0(value:String):void;

		function get field0():String;

		function doSomethingBlockUntilReturn(arg0:String):String;

		function doSomething(arg0:String):void

		function doSomeThingDoNotBlock(callbackFunction:Function,arg0:String):void;

	}
}
```

the remote object will extends the previous functions with the “InWorker” function that add the reply channel and the event channel :

```
package org.shankaa.coworkers.example
{
	import flash.system.MessageChannel;

	public interface InWorker_CoworkingObjectExample extends ICoworkingObjectExample
	{
		function doSomethingBlockUntilReturnInWorker(replyChannel:MessageChannel,
			eventChannel:MessageChannel,arg0:String):String;

		function doSomethingInWorker(replyChannel:MessageChannel,
			eventChannel:MessageChannel,arg0:String):void

		function doSomeThingDoNotBlockInWorker(replyChannel:MessageChannel,
			eventChannel:MessageChannel,
			callbackFunction:Function,arg0:String):void;
	}
}
```

where the reply channel is used to send the result and the event channel, will send event objects (a serialized form of Events that we have to unmarshall ourself.

The code in the main process could be something like

```
package org.shankaa.coworkers.example
{
	import flash.events.Event;
	import flash.system.MessageChannel;
	import flash.system.Worker;

	import org.shankaa.coworkers.WorkerManager;

	public class Example
	{

		private var remoteWorker		: Worker;
		private var requestChannel		: MessageChannel;
		private var coworkingObjectExample	: Stub_CoworkingObjectExample

		public function Example()
		{
			preinitialize();
			initialize();
		}

		protected function preinitialize():void
		{
			var xlsWorkerId : int = 1;
			remoteWorker = WorkerManager.initializeExcelDataWorker(xlsWorkerId,
				Worker.current);
			requestChannel = Worker.current.createMessageChannel(remoteWorker);
			requestChannel.addEventListener(Event.CHANNEL_STATE,
				onRequestChannelState);
			remoteWorker.setSharedProperty("requestChannel",requestChannel);
			remoteWorker.start();
		}

		protected function initialize():void{
			coworkingObjectExample = new Stub_CoworkingObjectExample(remoteWorker,
				"initialValue");
		}

		protected function onRequestChannelState(event:Event):void{
			trace("requestChannel state",(event.currentTarget as
				MessageChannel).state)
		}

		protected function doSomethingOnRemote(arg0:String):void{
			coworkingObjectExample.doSomething(arg0);
		}

		protected function doSomethingBlockUntilReturnOnRemote(arg0:String):void{
			var returnedValue : String =
				coworkingObjectExample.doSomethingBlockUntilReturn(arg0);
			trace("Example","doSomethingBlockUntilReturnOnRemote","value:",
				returnedValue);
		}

		protected function doSomeThingDoNotBlockOnRemote(arg0:String):void{
```

```
                var onCallback : Function = function(event:Event):void{
                        var returnedValue : String= (event.currentTarget as
                                MessageChannel).receive();
                        trace("Example","doSomeThingDoNotBlockOnRemote","value:",
                                returnedValue);
                }
                coworkingObjectExample.doSomeThingDoNotBlock(onCallback,arg0);
        }
    }
}
```

Stub and proxies

In some case, when applying a function on a remote object we must pass an argument that corresponds to another remote object.

When we have the stub of this remote object, it is possible to get a reference to it by passing an InWorker_Proxy as argument. This object holds the reference to the corresponding object in the remote worker.

```
package org.shankaa.coworkers.stubs
{
    /**
     *
     * @author Hugues Sansen, Shankaa
     *
     * Inworker_Proxy gives a handle on an object running on a remote DataWorker that
can be used in a stub as a reference to this remote object
     * an Inworker_Proxy is usually used from the main worker but is can also be used
between two remote workers...
     *
     * var excelFileProxy     : InWorker_Proxy;
     * excelFileProxy = new InWorker_Proxy();
     * excelFileProxy.setInWorker_Proxy(myExcelFile_Stub,["sheets",0]);
     * anotherStub.doSomething(excelFileProxy)
     *
     * Coworker is provided under the European Union Public Licence (EUPL)
     */
    public class InWorker_Proxy
    {
        public var handle                       : String;
        public var path                             : Array;

        public function InWorker_Proxy(){
        }

        public function setInWorker_Proxy(stub:Stub,...args):void{
            this.handle = stub.remoteHandle;
            path = args;
        }

        public function _setInWorker_Proxy(stub:Stub,args:Array):void{
            this.handle = stub.remoteHandle;
            path = args;
        }


    }
}
```

The following code show how to pass the first sheet of the ExcelFile as an argument of a function:

```
var excelFileProxy        : InWorker_Proxy = new InWorker_Proxy();
excelFileProxy.setInWorker_Proxy(myExcelFile_Stub,["sheets",0]);
anotherStub.doSomething(excelFileProxy)
```

Triangle process computation

It is not recommended to have too many Workers running but it may be useful to do so.

The Coworker framework allows the creation of 2 workers, both controlled by the main process and allows those worker to communicate together. InWorker_Proxies can be used to pass arguments between those workers when the main process has access to them.

How to apply the coworker framework to the Excel As3 API

For our purpose we have modified the Excel As3 API in order to :

- make the ExcelFile object serializable with the standard AS3 serialization mechanism,
- make it accept big FAT files, the original API was not able to use extended fFAT files,
- make it accept xlsx files that use the OOXML format,
- use it in a worker.

In Visumag, the size of Excel files can be very large when we deal with a big number of products (lines) and a big number of data per product (columns).
The usage of Workers was a real benefit. Besides we have to compute data in real time when viewing a store in 3D and interrogating the shelves, the line of shelves or when we have to draw diagrams.
We also need to pass events from the remote process to the main process in order to tell the user what the application is currently doing.
Besides we are doing many of hyper-cubic computations that are not described here.

In order to call the function from the main, ExcelFile that implements:

```
public function loadFromByteArray(onCallback:Function, onEvent:Function ,xls:ByteArray, fileExtension:String):void{
. . .
}
```

must also implement:

```
public function loadFromByteArrayInWorker(replyChannel:MessageChannel, eventChannel:MessageChannel, xls:ByteArray, fileExtension:String):void{
. . .
}
```

On its side, the Stub_ExcelFile will implement :

```
public function loadFromByteArray(onCallback:Function, onEvent:Function, xls:ByteArray, fileExtension:String):void
{
applyOnRemote("loadFromByteArray"+"InWorker",onCallback,onEvent,xls,fileExtension);
}
```

The function that builds the ExcelFile object from the raw data is

```
/**
             * loads transforms .xls and .xlsx files into an ExcelFile object.
             * used either in a worker or directly inside the main process.
             *
             *
             *
             * @param xls
             * @param fileExtension used to differentiate .xls and .xlsx files
             * @return
```

```
         *
         */
        public function loadFromByteArray(xls:ByteArray,
                                          fileExtension:String):ExcelFile{
              var result : ExcelFile;
              if(fileExtension == "xls"){
                    result = loadFromXlsByteArray(xls);
              } else if(fileExtension == "xlsx"){
                    result = loadFromXlsxByteArray(xls);
              }
              return result;
        }
```

The corresponding “inWorker” function is :

```
        /**
         * used in worker multi threading
         *
         * @param replyChannel
         * @param eventChannel
         * @param xls
         * @param fileExtension xls or xlsx
         */
        public function loadFromByteArrayInWorker(replyChannel:MessageChannel,
                    eventChannel:MessageChannel,xls:ByteArray,
                    fileExtension:String):void{
              var onSizingSpreadsheetEvent : Function =
                          function(event:XLSEvent):void{
              eventChannel.send({type:event.type,numRows:event.
                    numRows,numColumns:event.numColumns});
              }
              var onProgressEvent       : Function = function(event:Event):void{
                    if(event is ProgressEvent){
                          var pe : ProgressEvent = event as ProgressEvent;
                          eventChannel.send({type:event.type,
                                bytesLoaded:pe.bytesLoaded,
                                bytesTotal:pe.bytesTotal});
                    } else {
                          eventChannel.send({type:event.type});
                    }
              }
              this.addEventListener(XLSEvent.SIZING_SPREADSHEET,
                    onSizingSpreadsheetEvent);
              this.addEventListener(XLSEvent.SIZING_SPREADSHEET_OVER,
                    onSizingSpreadsheetEvent);
              this.addEventListener("excelFormatError",onProgressEvent);
              this.addEventListener("excelProgess",onProgressEvent);
              this.addEventListener("excelComplete",onProgressEvent);
              this.addEventListener("unpackingOOXMLFile",onProgressEvent);
              this.addEventListener("unpackingOOXMLFileOver",onProgressEvent);
              this.addEventListener("unpackingOOXMLFileFailed",onProgressEvent);

              var result : ExcelFile = loadFromByteArray(xls,fileExtension);
              replyChannel.send(result);

    this.removeEventListener(XLSEvent.SIZING_SPREADSHEET,onSizingSpreadsheetEvent);
```

```
        this.removeEventListener(XLSEvent.SIZING_SPREADSHEET_OVER,onSizingSpreadsheetEvent);
                    this.removeEventListener("excelFormatError",onProgressEvent);
                    this.removeEventListener("excelProgess",onProgressEvent);
                    this.removeEventListener("excelComplete",onProgressEvent);
                    this.removeEventListener("unpackingOOXMLFile",onProgressEvent);
                    this.removeEventListener("unpackingOOXMLFileOver",onProgressEvent);
                    this.removeEventListener("unpackingOOXMLFileFailed",onProgressEvent);
                }
```

Note that the “InWorker” function subscribes to events, marshals them into event objects and sends them to the main process.

The Excel file display in a DataGrid application example

The WorkerManager

The worker manager manages the creation of the remote worker. We have decided to isolate the creation of the worker since sometimes the created swf file grows at each compilation.

When it is the case we unset the worker and compile again. A centralized definition of worker(s) is more convenient.

```
package org.shankaa.coworkers
{
        import flash.system.MessageChannel;
        import flash.system.Worker;
        import flash.system.WorkerDomain;
        import flash.utils.Dictionary;


        /**
         *
         * @author Hugues Sansen, Shankaa
         *
         * The WorkerManager is part of the coworkers frameworker.
         * We centralize the creation of workers here.
         * Why? Because most of the time the worker swf tend to grow at each clean and
compilation.
         * If you experience the issue,
         * - remove all the workers in the application properties window.
         * - comment the line
         *                  dataWorker                =
WorkerDomain.current.createWorker(Workers.org_shankaa_coworkers_DataWorker,true);
         * - clean your code and compile,
         * - declare you worker classes as workers,
         * - clean and compile,
         * - uncomment the worker declaration lines,
         * - clean and compile again...
         *
         *
         *  Coworker is provided under the European Union Public Licence (EUPL)
         *
```

```
	 */
	public final class WorkerManager
	{

		public static const WORKER_IDS		: Dictionary = new Dictionary();


		/**
		 * creates a worker running a DataWorker instance
		 *
		 * @param workerNumber
		 * @param requestWorker
		 * @return
		 *
		 */
		public static function initializeExcelDataWorker(workerNumber : int,
requestWorker:Worker):Worker{
			var dataWorker			: Worker;
			var requestChannel	: MessageChannel;

			dataWorker				=
WorkerDomain.current.createWorker(Workers.org_shankaa_coworkers_ExcelDataWorker,true);
			requestChannel = requestWorker.createMessageChannel(dataWorker);
			dataWorker.setSharedProperty("requestChannel",requestChannel);
			WORKER_IDS[dataWorker] = requestChannel;

			return dataWorker;
		}

		public static function getRequestChannel(remoteWorker:Worker):MessageChannel{
			var requestChannel		: MessageChannel	=
WORKER_IDS[remoteWorker];
			return requestChannel;
		}
	}
}
```

The ExcelDataWorker class

```
package org.shankaa.coworkers
{
	import flash.net.registerClassAlias;

	import org.as3.ms.xls.Cell;
	import org.as3.ms.xls.ExcelFile;
	import org.as3.ms.xls.Sheet;

	public class ExcelDataWorker extends DataWorker
	{
		public function ExcelDataWorker()
		{
			super();
		}

		protected override function initialize():void
		{
```

```
                    // register all the classes you need to pass from the main worker to
this worker
                    registerClassAlias("org.as3.ms.xls.ExcelFile", ExcelFile);
                    registerClassAlias("org.as3.ms.xls.Sheet", Sheet);
                    registerClassAlias("org.as3.ms.xls.Cell", Cell);
              }
        }
}
```

The application code

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                                    xmlns:s="library://ns.adobe.com/flex/spark"
                                    xmlns:mx="library://ns.adobe.com/flex/mx"
                                    preinitialize="preinitializeHandler(event)"
                                    close="closeHandler(event)">
        <s:Group width="100%" height="100%">
              <s:layout>
                    <s:VerticalLayout />
              </s:layout>

        <s:Button id="loadExcelButton" label="load Excel file"
click="Loadbutton_clickHandler(event)"/>
        <mx:DataGrid id="excelSpreadSheet" width="100%" height="100%"
                              dataProvider="{xlsSheetValues}" draggableColumns="false"
                              horizontalScrollPolicy="{ScrollPolicy.ON}" lockedRowCount="1"
                              verticalScrollPolicy="{ScrollPolicy.ON}"
                              contentBackgroundAlpha="0.0"/>

        </s:Group>
        <!--comments
        Author : Huges Sansen, SHANKAA
              This example shows how to use the ExcelFile api and CoWorker (worker to
worker communication framework proposed by Shankaa)
              This example opens a Datagrid with the first spreadsheet of a workbook.
              The analysis of the spreadsheet is done inside a worker.

              deals with .xls and .xlsx files
        -->
        <fx:Script>
              <![CDATA[
                    import flash.net.registerClassAlias;

                    import mx.events.FlexEvent;

                    import flashx.textLayout.container.ScrollPolicy;

                    import org.as3.ms.events.XLSEvent;
                    import org.as3.ms.xls.Cell;
                    import org.as3.ms.xls.ExcelFile;
                    import org.as3.ms.xls.Sheet;
                    import org.shankaa.coworkers.WorkerManager;
```

```
		import org.shankaa.coworkers.stubs.Stub_ExcelFile;

		private var excelFile 			: ExcelFile;
		private var _excelFileName 		: String;
		private var _excelFileExtension 	: String;
		private var _excelFileByteArray : ByteArray;


		private var remoteWorker 	: Worker;
		private var requestChannel 		: MessageChannel;

		[Bindable]
		public var xlsSheetValues 	: Array;


		protected function preinitializeHandler(event:FlexEvent):void
		{
			var xlsWorkerId : int = 1;
			remoteWorker =
WorkerManager.initializeExcelDataWorker(xlsWorkerId,Worker.current);
			requestChannel =
Worker.current.createMessageChannel(remoteWorker);

	requestChannel.addEventListener(Event.CHANNEL_STATE,onRequestChannelState);
			remoteWorker.setSharedProperty("requestChannel",requestChannel);
			remoteWorker.start();
		}

		protected function onRequestChannelState(event:Event):void{
			trace("requestChannel state",(event.currentTarget as
MessageChannel).state)
		}



		protected function closeHandler(event:Event):void
		{
			remoteWorker.terminate();
		}

		protected function
loadXLSFromByteArray(xls:ByteArray,onLoaded:Function,useThreading : Boolean = true):void {
			if(useThreading && WorkerDomain.isSupported){
				xlsSheetValues = null;
				registerClassAlias("org.as3.ms.xls.ExcelFile", ExcelFile);
				registerClassAlias("org.as3.ms.xls.Sheet", Sheet);
				registerClassAlias("org.as3.ms.xls.Cell", Cell);

				var remoteExcel 		: Stub_ExcelFile	= new
Stub_ExcelFile(remoteWorker);

				var onCallback 		: Function 		=
function(event:Event):void{
					var excelFile 	: ExcelFile 		=
(event.currentTarget as MessageChannel).receive();
					onLoaded(excelFile);
				}
```

```
                                    var onEvent                         : Function
        = function(event:Event):void{
                                          var eventObject               : *
        = (event.currentTarget as MessageChannel).receive();
                                          var progressEvent   : ProgressEvent;
                                          var xlsEvent        : XLSEvent;
                                          switch(eventObject.type){
                                                case "excelProgess":
                                                case "excelComplete":
                                                      progressEvent = new
ProgressEvent("excelProgess",false,true,eventObject.bytesLoaded,eventObject.bytesTotal);
                                                      onExcelAnalysisProgress(progressEvent);
                                                      break;
                                                case "excelFormatError":
                                                      progressEvent = new
ProgressEvent("excelFormatError",false,true,eventObject.bytesLoaded,eventObject.bytesTotal
);
                                                      onExcelFormatError(progressEvent);
                                                      break;
                                                case "sizingSpreadsheet":
                                                case "sizingSpreadsheetOver":
                                                      xlsEvent = new
XLSEvent("sizingSpreadsheetOver",eventObject.numRows,eventObject.numCol);
                                                      onSizingSpreadsheet(xlsEvent);
                                                      break;
                                          }
                                    }


        remoteExcel.loadFromByteArray(onCallback,onEvent,xls,_excelFileExtension);


                              } else {
                                    excelFile = new ExcelFile();

        excelFile.addEventListener("excelProgess",onExcelAnalysisProgress);

        excelFile.addEventListener("excelFormatError",onExcelFormatError);

        excelFile.addEventListener("sizingSpreadsheet",onSizingSpreadsheet);

        excelFile.addEventListener("sizingSpreadsheetOver",onSizingSpreadsheet);

                                    var onXLSLoaded : Function = function():void{
                                          onLoaded(excelFile);
                                    }


        excelFile.loadFromByteArraySimulatedThreading(xls,onXLSLoaded,2000);
                              }
                        }

                        /**
                         * handler for the ExcelFile progressEvent
                         *
                         * must be overriden
                         */
                        protected function onExcelAnalysisProgress(event:ProgressEvent):void{
```

```
                        trace("excel progress",event.bytesLoaded,"on",event.bytesTotal,
Math.round(100*event.bytesLoaded/event.bytesTotal),"%");
                }

                /**
                 * must be overriden
                 *
                 * @param event
                 *
                 */
                protected function onExcelFormatError(event:ProgressEvent):void{

                }

                protected function onSizingSpreadsheet(event:XLSEvent):void{

                }

                protected function onXLSLoaded(xlsFile:ExcelFile):void{
                        var sheet : Sheet = xlsFile.sheets[0];
                        xlsSheetValues = sheet.values;
                }

                protected function Loadbutton_clickHandler(event:MouseEvent):void
                {
                        var fileRef:FileReference = new FileReference();
                        fileRef.addEventListener(Event.CANCEL, cancelHandler);
                        fileRef.addEventListener(Event.SELECT, selectDataHandler);
                        var xlsFileFilter:FileFilter = new FileFilter("Data Excel file
(*.xls,*.xlsx)", "*.xls;*.xlsx");
                        var formats:Array = new Array(xlsFileFilter);
                        fileRef.browse(formats);

                }

                private function cancelHandler(event:Event):void{

                }

                private function selectDataHandler(event:Event):void{
                        var file:FileReference = FileReference(event.target);
                        var fileName:String = file.name;
                        file.addEventListener(Event.OPEN,openHandler);
                        file.addEventListener(ProgressEvent.PROGRESS,progressHandler);
                        file.addEventListener(Event.COMPLETE, dataCompleteHandler);
                        file.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
                        file.addEventListener(Event.OPEN, openHandler);

                        file.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
securityErrorHandler);

                        // it seems that it is better to delay the load activation
                        var loadFunction:Function = function():void{
                                file.load();
                        }
                        loadFunction();
                }
```

```
                        private function openHandler(event:Event):void{
                              trace("openHandler",event);

                        }

                        private function progressHandler(event:ProgressEvent):void{
                              trace("progress",event.bytesLoaded,event.bytesTotal);
                        }

                        private function dataCompleteHandler(event:Event):void{
                              var file:FileReference = FileReference(event.target);
                              _excelFileName = file.name;
                              _excelFileExtension = file.extension;

                              excelFileByteArray= file.data;
                        }

                        private function ioErrorHandler(event:IOErrorEvent):void{
                              trace("ioErrorHandler: " + event.errorID);
                        }

                        private function securityErrorHandler(event:SecurityErrorEvent):void {
                              trace("securityErrorHandler: " + event.errorID);
                        }

                        private function set excelFileByteArray(value:ByteArray):void{
                              try{
                                    _excelFileByteArray = value;
                                    _excelFileByteArray.position = 0;

      loadXLSFromByteArray(_excelFileByteArray,onXLSLoaded,true);

                              } catch (error:Error){

                              }
                        }

                  ]]>
            </fx:Script>

</s:WindowedApplication>
```

Conclusion

Once again, multithreading would have been better and inter process communication is not as fast. However multi-processing has its advantages, especially with multicore machines.
We have been using the Coworker API almost since the Worker API was introduced in AS3. It is pretty robust and fits perfectly our need when a 3D image of a shop is running on the main process while we do computation in the background.
The Coworker API has also allowed us to minimize the modification of the initial mono process application that was we started with more than 3 years ago.
Some features are not well documented in the Worker API : the way to set the timeout of a MessageChannel as well as how a MessageChannel is garbaged. However even if we create many message channels in our application (2 per call to a remote object) we haven't seen any problem.
The other issue we have with using workers on Eclipse is the fact the the worker swf grows. When its size is too big, we have to delete it and recompile it.
All the code is available on GitHub under Shankaa_coworker_framework [4].

References

[1] Multiprocessing with Flex, Hugues Sansen, Shankaa, June 4th, 2010, unpublished
[2] Visumag, un logiciel de présentation des résultats commerciaux des supermarchés ou la “tammetisation” des chiffres, Hugues Sansen, Shankaa,, Damien Morlier, Assortiment-Conseil, Gérard Chollet, Institut Télécom, , Hervé Crespel, x4it, Conference: International Workshop IHM, At Sousse, Tunisia, June 6th, 2012 https://www.researchgate.net/publication/265123514_Visumag_un_logiciel_de_prsentation_des_rsultats_commerciaux_des_supermarchs_ou_la_tammetisation_des_chiffres
[3] https://code.google.com/p/as3xls/
[4] Shankaa_coworker_framework on GitHub
https://github.com/Shankaa/Shankaa_coworker_framework/