Write-up on the decisions taken

Here are the notes taken by John Wilson regarding the decisions and discussions during the meeting.

Improved MetaClass supporting an enhanced MOP

We reviewed the use of the MetaClass in the Groovy run time system. We agreed that the current Meta Object Protocol provided by the MetaClass was inadequate to support the name resolution rules we would like to implement. We discussed my proposal for an enhanced MOP and agreed that this would form the basis for the standard MOP in Groovy 1.0.

Action(tug): I agreed to refine the abstract class which will form the basis of the new MetaClass - this will be done using the experimental directory in CVS HEAD and the work will be available for anybody to experiment and comment on.

Action(tug): I agreed to produce an interface which contained those methods on the new MetaClass which represent the official MOP. This is a documentation activity - there are many methods on the abstract class which are needed for efficient operation but which do not contribute to the functionality exposed by the class, the Interface allows the user to understand what the important methods are. For the avoidance of doubt: people wishing to write their own custom MetaClasses must subclass the abstract class - implementing the interface is not sufficient.

We discussed the use of the MetaClass at runtime. We agreed that the compiler would generate code which directly resolved all local names (a local name is a parameter to a function or closure or a name declared as a local variable in a function or closure body) and all class names (e.g. String, Integer, java.util.Map). All other name resolution is done via the appropriate MetaClass.

We discussed the need for a sort of Groovy refection API which would allow the methods, properties and attributes added via the MetaClass (e.g. via Default Groovy Methods) to be determined. This API would also need to tell the inquirer that there were an infinite number of methods. properties etc. In the case where the MetaClass supported fully dynamic name resolution. We agreed that this API should be on the MetaClass

Action(tug): I agreed to draft an API as part of my work on the new MetaClass.

Name resolution in scripts

We discussed name resolution in scripts. We originally agreed that "def" would create a local variable (i.e. one not in the Binding). However James was concerned that this might cause problems with line by line execution from the interactive console. The issue was not resolved.

Changes on the GroovyObject API

We discussed changes to the GroovyObject API. I proposed that the method signatures and semantics of invokeMethod, and get/set Property should be the same as the equivalent method on the new MetaClass (i.e. the failure to resolve a name is signalled by returning a special value not by throwing an exception). There was a reluctance to agree to this on the basis that it was a breaking change. James proposed that setMetaclass should be removed and replaced with a method which returned a copy of the current object with the new MetaClass. Whilst there was no fundamental objection from the meeting to this change it is also a breaking change and James did not have a concrete proposal for implementation. No decision was taken to change GroovyObject.

Module-like support through a "use" concept

We discussed the current implementation of "use". James felt that it was slightly too general (e.g. the parameters to the use method can be arbitrary expressions evaluation to class values at run time). There was also concern that the changes to behaviour are not just in the lexical scope but apply to the methods or closures called from that scope. It was agreed that this was undesirable behaviour. James proposed replacing the current implementation with a restricted form of use (probably with a different name) which only allowed literal classes to be specified. The effect of the changes to behaviour would only be in the lexical scope of the block to which it was applied. (Note: use is a method which works on a closure, the new scheme will be a language construct which works on a block). The implementation details of this would be that the new construct would generate a MetaClass which would be used instead of the normal MetaClass to resolve the names in the lexical scope. I proposed that there would be mechanisms for applying this mechanism at the method and class level (i.e. to be able to add behaviour to objects for the whole of a method or for the whole of the class. It was agreed that these two extra mechanisms were desirable but there was some doubt as to if they would make it into the 1.0 release. We will need a new way of specifying the behaviour to be added in this manner (i.e. there will need to be an agreed format like DefaultGroovyMethods). The discussion did not go into the detail of this new format. Also we did not agree on the keyword or the precise syntax of the new construct(s).

Action(tug): I did not volunteer at the time but will do so now. I will propose an API on the MetaClass to support this functionality. I will look into a possible format for the classes which add behaviour dynamically.

Name resolution in closures

We discussed name resolution in closures. James was very strongly of the opinion that no name inside a closure body should resolve to anything on the closure object (e.g. hashCode(), clone(), toString() should all be called on the enclosing class not on the closure object). From this it is now clear that "this" in a closure body refers to the enclosing class. With this convention closure bodies can no longer use owner or delegate to access

properties on the closure.

We discussed the treatment of builders in general and markup in particular. Whist it appeared that some of the functionality of the "use" replacement might be helpful in resoving the conflicting requirements of builders with a finite set of names and ones with an infinite set (e.g. SwingBuilder vs MarkupBuilder) there was no conclusion drawn before I had to leave the meeting.