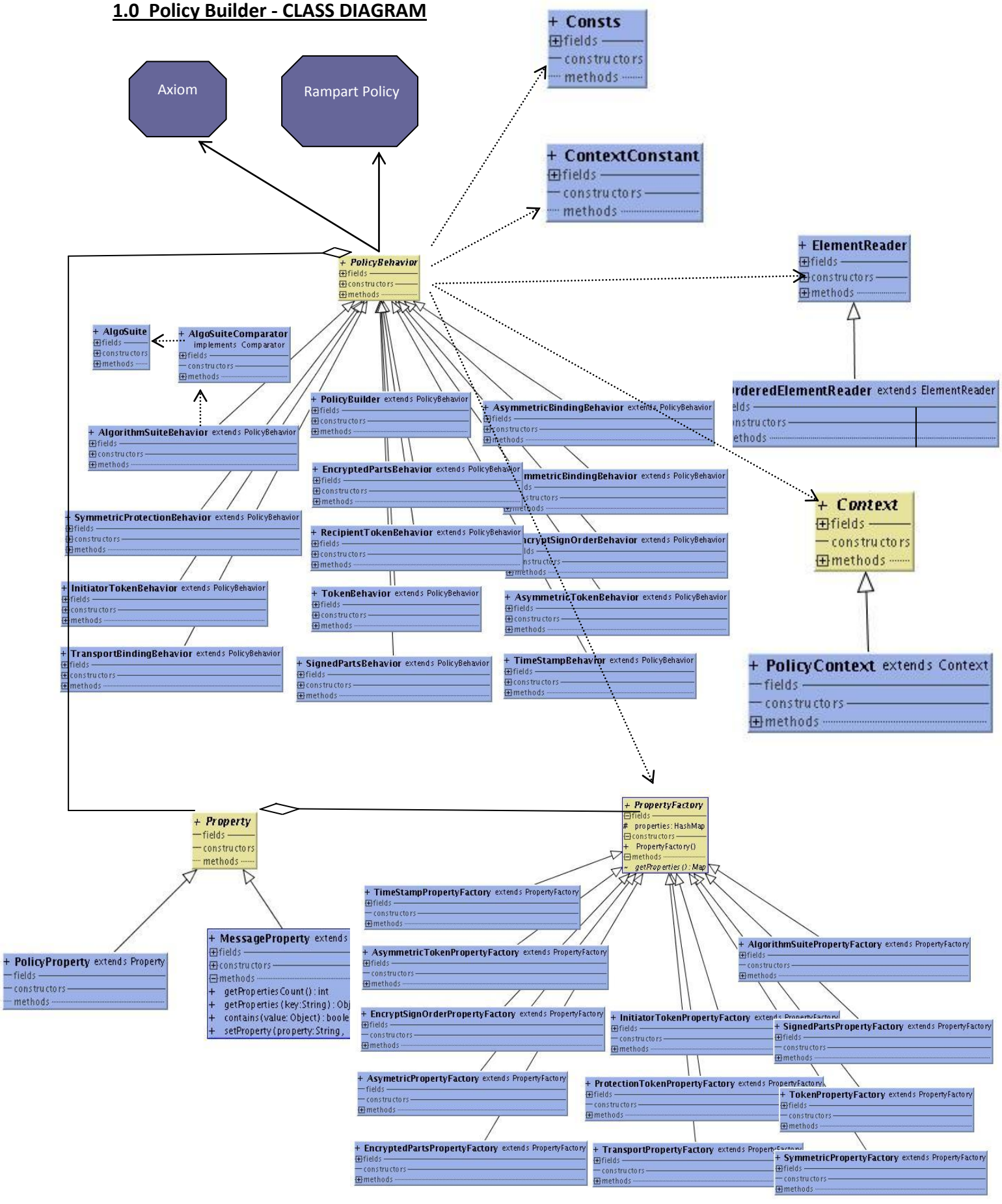
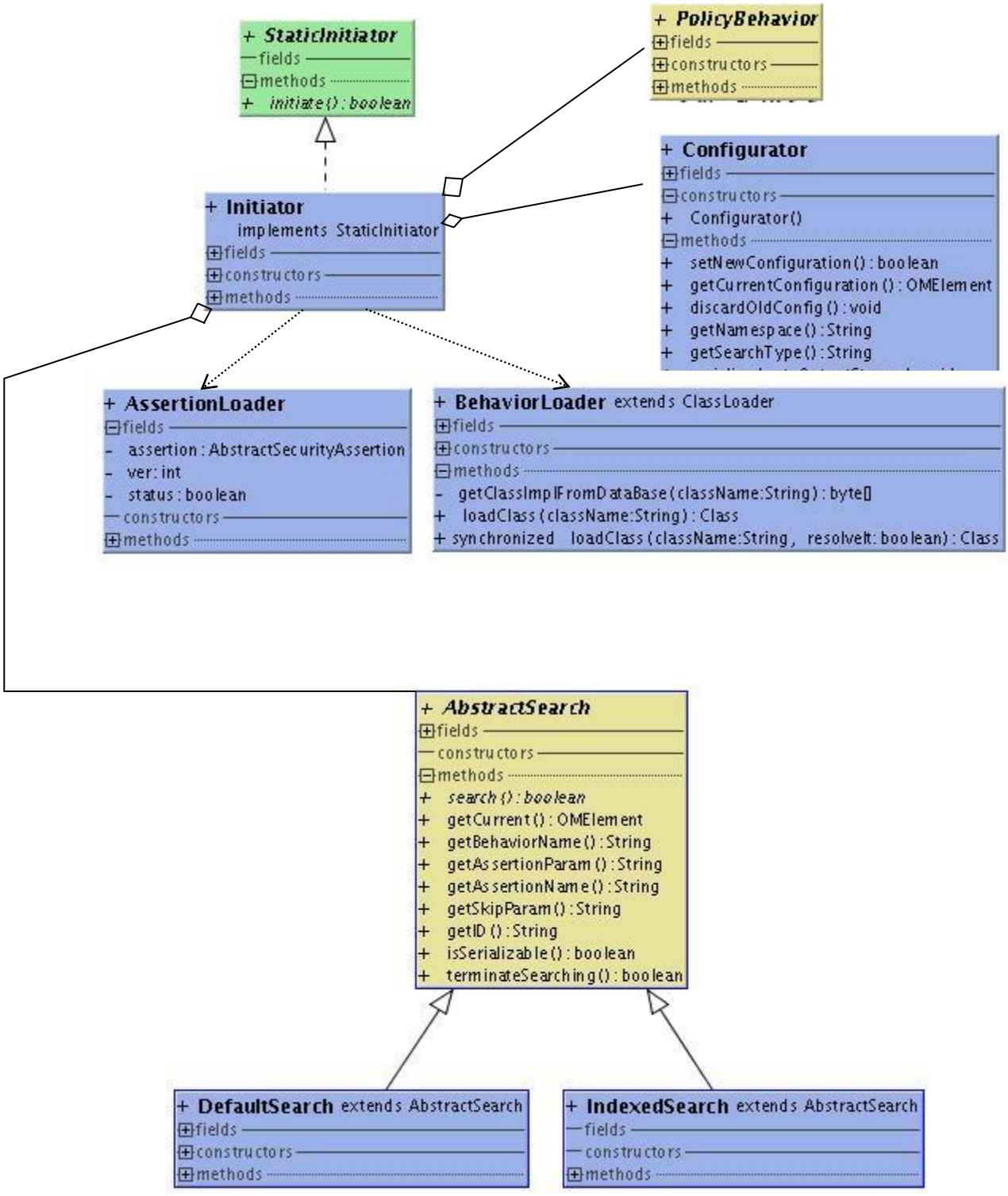
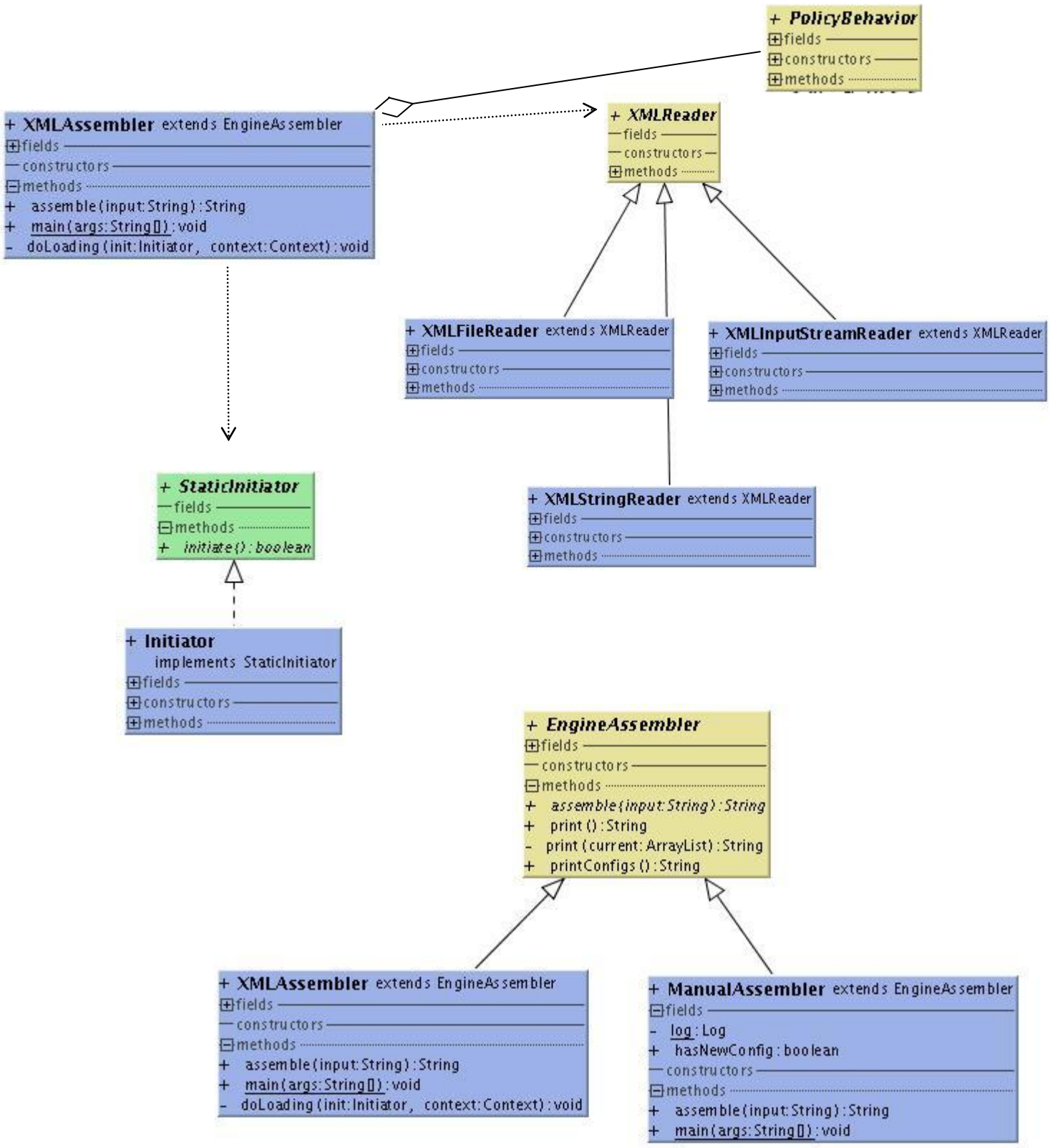
1.0 Policy Builder - CLASS DIAGRAM

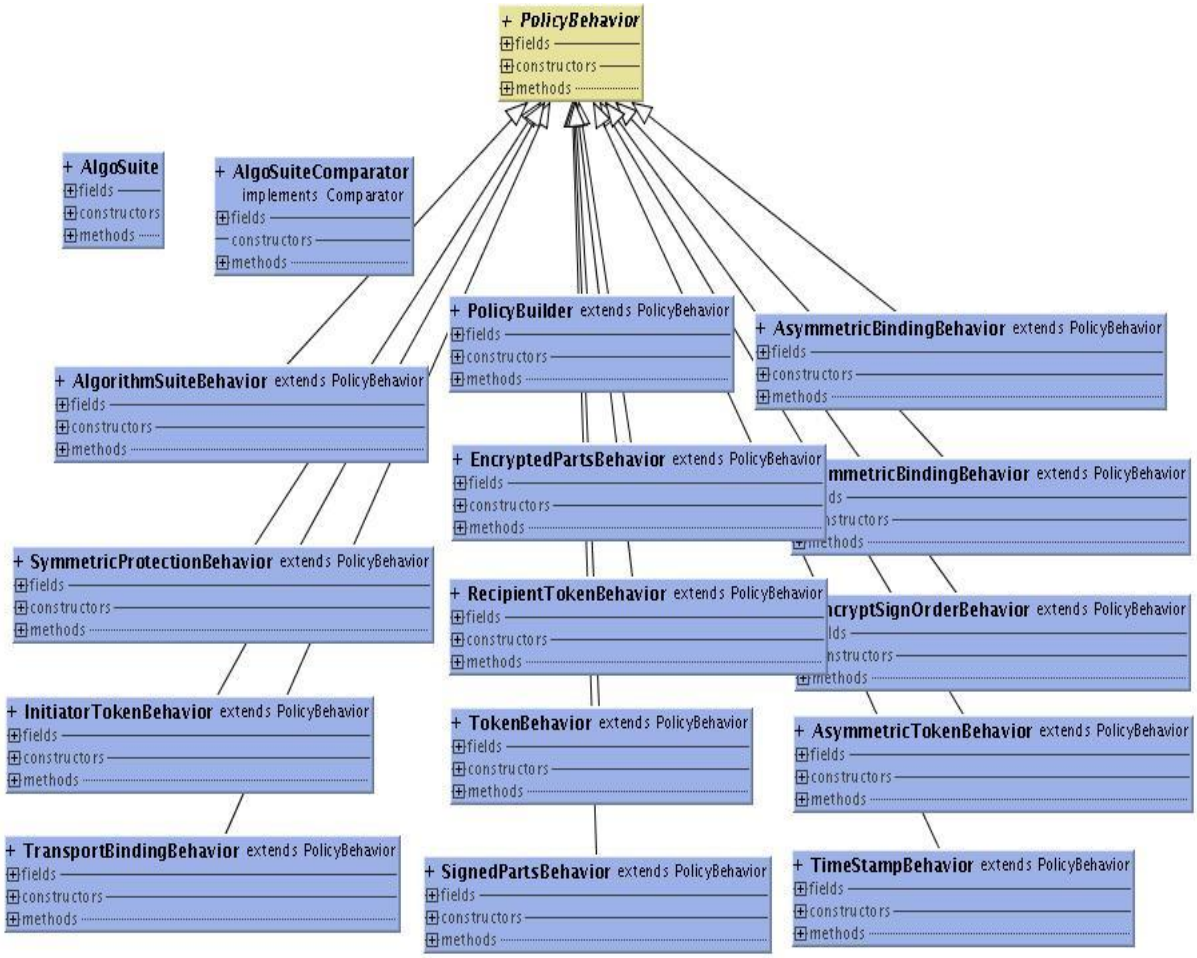



+ StaticInitiator
fields
methods
+ initiate() : boolean
+ PolicyBehavior
fields
constructors
methods
+ Configurator
fields
constructors
+ Configurator()
methods
+ setNewConfiguration() : boolean
+ getCurrentConfiguration() : OMElement
+ discardOldConfig() : void
+ getNamespace() : String
+ getSearchType() : String
+ Initiator
implements StaticInitiator
fields
constructors
methods
+ AssertionLoader
fields
- assertion : AbstractSecurityAssertion
- ver : int
- status : boolean
constructors
methods
+ BehaviorLoader extends ClassLoader
fields
constructors
methods
- getClassImplFromDataBase(className:String) : byte[]
+ loadClass(className:String) : Class
+ synchronized loadClass(className:String, resolveIt:boolean) : Class
+ AbstractSearch
fields
constructors
methods
+ search() : boolean
+ getCurrent() : OMElement
+ getBehaviorName() : String
+ getAssertionParam() : String
+ getAssertionName() : String
+ getSkipParam() : String
+ getID() : String
+ isSerializable() : boolean
+ terminateSearching() : boolean
+ DefaultSearch extends AbstractSearch
fields
constructors
methods
+ IndexedSearch extends AbstractSearch
fields
constructors
methods


+ PolicyBehavior
fields
constructors
methods
+ XMLAssembler extends EngineAssembler
fields
constructors
methods
+ assemble(input:String):String
+ main(args:String[]):void
- doLoading(init:Initiator, context:Context):void
+ XMLReader
fields
constructors
methods
+ XMLFileReader extends XMLReader
fields
constructors
methods
+ XMLInputStreamReader extends XMLReader
fields
constructors
methods
+ XMLStringReader extends XMLReader
fields
constructors
methods
+ StaticInitiator
fields
methods
+ initiate():boolean
+ Initiator
implements StaticInitiator
fields
constructors
methods
+ EngineAssembler
fields
constructors
methods
+ assemble(input:String):String
+ print():String
- print(current:ArrayList):String
+ printConfigs():String
+ XMLAssembler extends EngineAssembler
fields
constructors
methods
+ assemble(input:String):String
+ main(args:String[]):void
- doLoading(init:Initiator, context:Context):void
+ ManualAssembler extends EngineAssembler
fields
- log:Log
+ hasNewConfig:boolean
constructors
methods
+ assemble(input:String):String
+ main(args:String[]):void

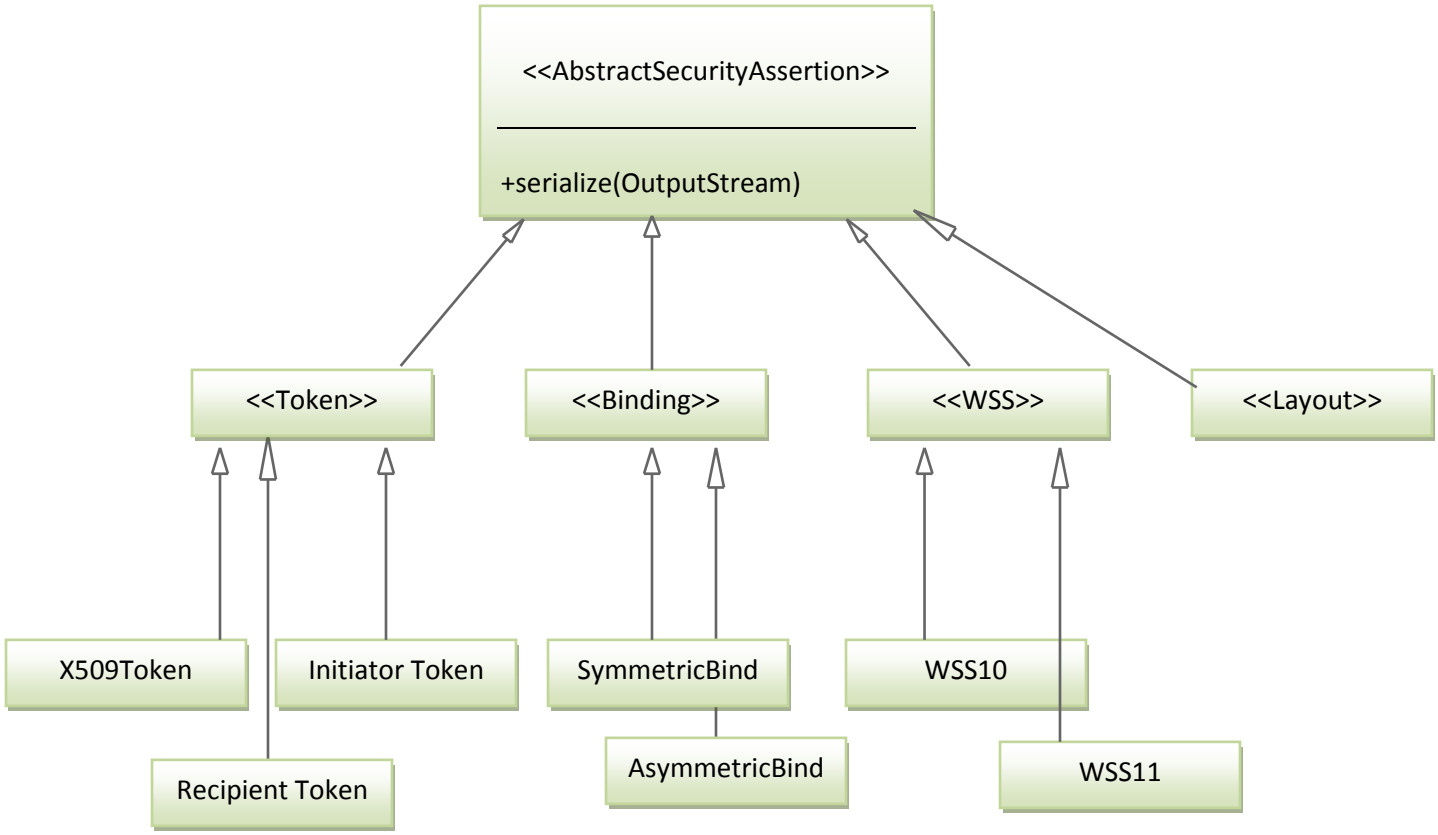
2.0 Policy Builder - DATA Model

2.1 Policy Behavior Model



Basic unit of policy checking mechanism is a Policy Behavior**.** One can create a policy checking handler by simply extending to 'PolicyBehavior' Abstract Class**.** Each Behavior is responsible for extracting policies of a particular wsp : policy scenario. Some set of behaviors can be inherently focused on checking a particular wsp policy instance and related types (ie:-Symmetric Binding and related assertion types) and is grouped into configs/phases. Each concrete Policy Behavior is provided with init(),evaluate(),handleSuccessor(),skip() functionalities by it's abstract interface in-order to work on each policy scenario. For Example 'init()' can be used to initiate the respective behavior with the required properties(ie:-gathered from a PropertyFactory) it depends on and 'handleSuccessor()' can be used to go to the next behavior(as specified in the descriptor) when policy evaluation is completed. Also there is no restriction to use functionalities of other behaviors to fulfill/check a particular property and hence behaviors may contain dependencies on other behaviors as well. Another important fact is policy derivation system's dependency on Rampart Security Policy model (org.apache.ws.secpolicy.model) and therefore output will be a Serialization dependent on this model itself.

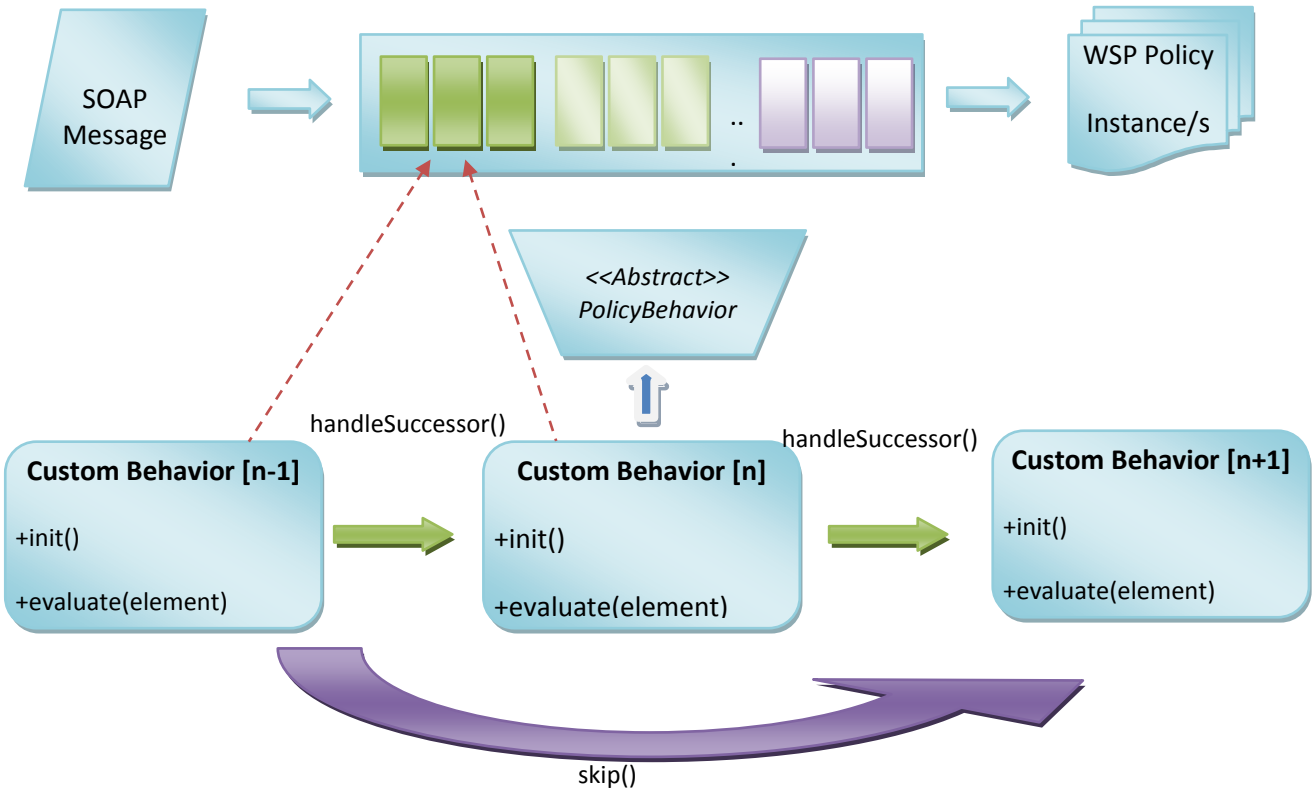
2.2 Rampart Security Policy Model



Rampart Security Policy Model is really important to Policy Builder System since it provides the essential wsp compliant policy model to build the hierarchy based on the underline information in the SOAP message itself . When the required information is gathered in the process of a handler (or 'behavior') the aforementioned information is properly fed into the respective Policy Model Constructs . Therefore after a successful 'phase' completion the necessary ws policy assertion hierarchy will be built and Eventually will be serialized into a consumable format.

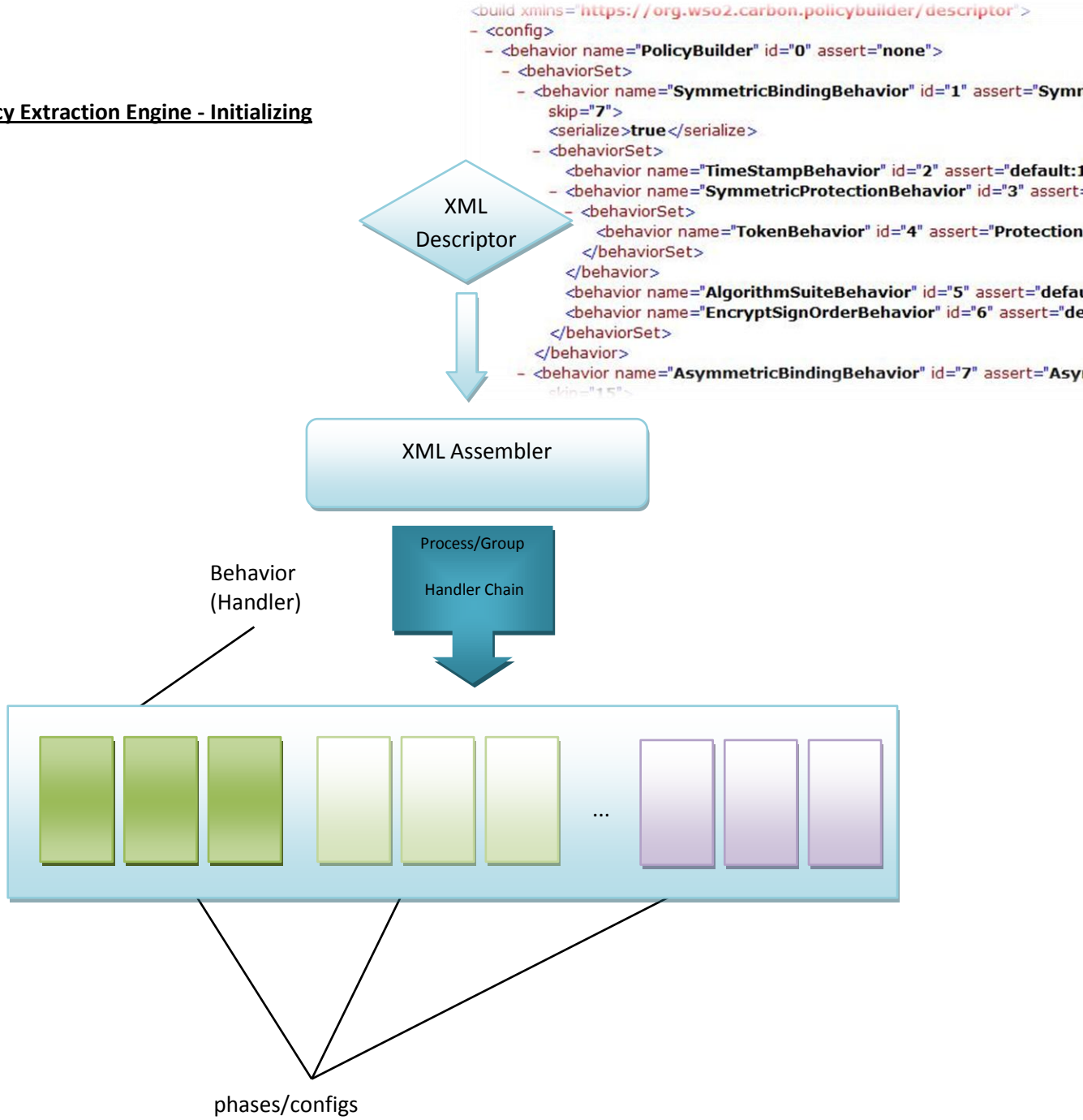
3.0 Policy Builder – Internal Operation

3.1 Policy Extraction Process



Policy Engine extracts a wsp:Security Policy compliant policy assertions using the underline SOAP message.Basically this Procedure is started with the provided SOAP Message entering the handler/behavior chain which was contructed at the startup of the policybuilder system. Each individual handler checks for underline policy mappings in the SOAP message inside their 'evaluate' methods. Developers are provided with two abstract methods 'handleSuccessor(OMelement)' and 'skip(OMElement)' inorder to continue with the current phase or skip to the next phase(ie:-incase SOAP doesn't relate to current policy checking phase) respectively.

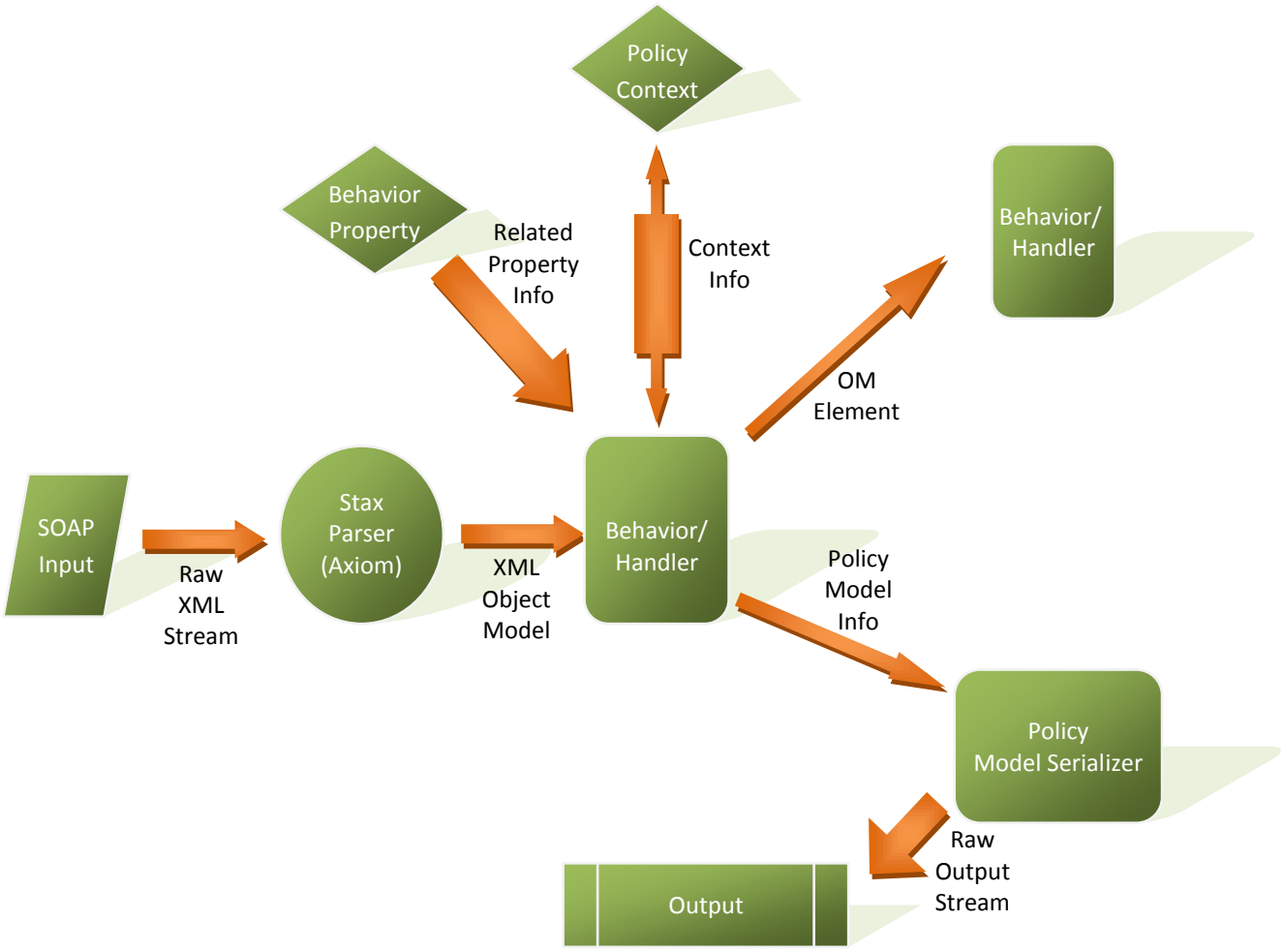
3.2 Policy Extraction Engine - Initializing



An External Descriptor file(builder.xml) is used to build the handler chain. This Descriptor defines the order of the handler chain , different phases(ie:-'config' s), assertion types associated with each behavior, etc. This is kind of similar and based on Chain of Responsibility (ie:-Chain of Responsibility Design Pattern)configuration for a handler set (/set of Behaviors) . However behavior set is grouped into configurations(ie:-configs) or phases where each seperate set can be evaluated separately.Policy extraction will be done for each phase and a security policy assertions will be derived for each individual phase if the phase is successfully completed. Following shows a descriptor configured to have two phases checking may be two different policy scenarios.

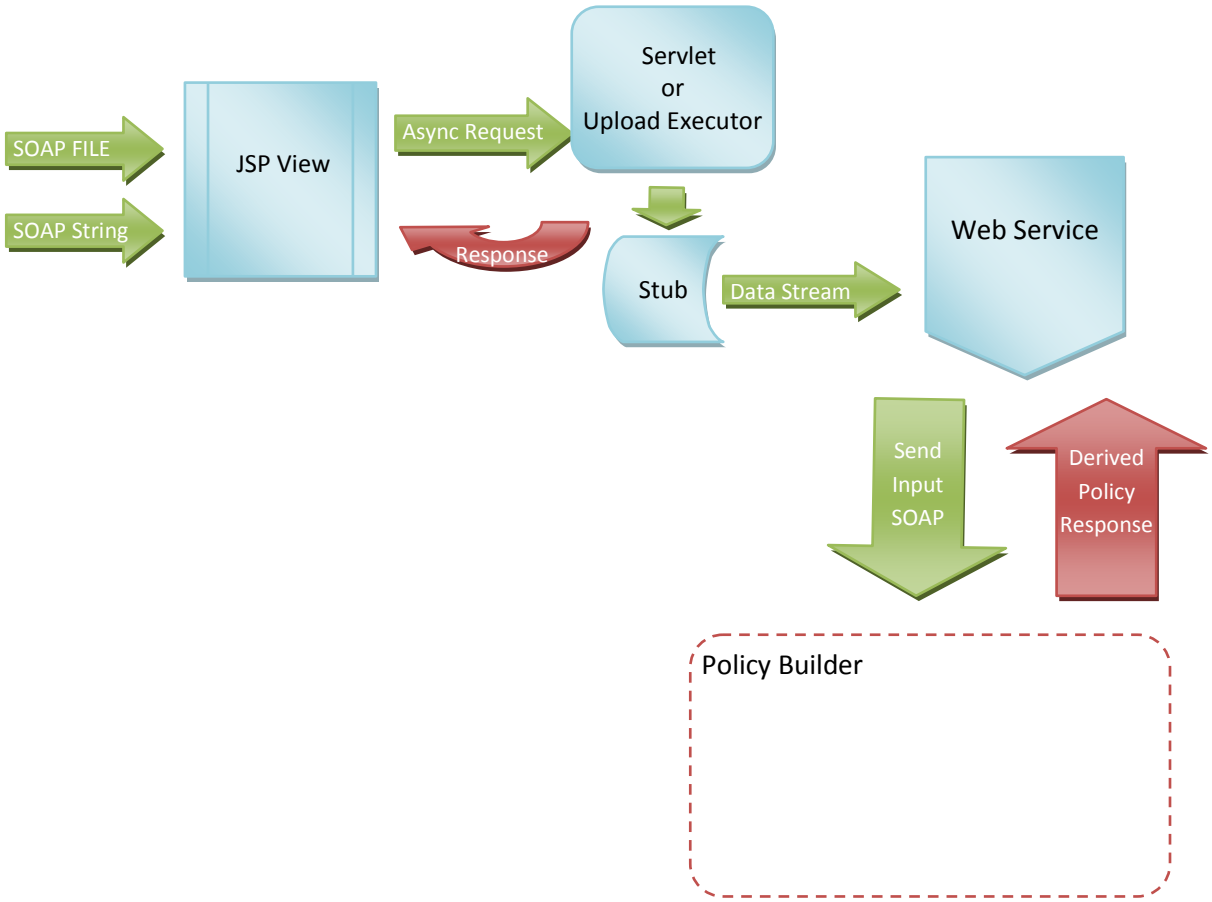
```
- <build xmlns="https://org.wso2.carbon
  + <config>
  + <config>
  </build>
```

3.3 Handler Data Flow



Axiom Object model and Rampart Security Policy Model is extensively used in the Policy Builder System.Policy Builder uses StaxParser (A Pull parser used in Apache Axiom) to generate XML Object Model and provide it as input to the handlers. Using the property information and other context information , underline security policy information is derived and inturn provided for Policy Model which is then made responsible for policy serialization. The OMElement (Axiom XML Model Object) will be propagated through the handler chain ('behavior Set') since the XML infoset information of the SOAP message will be needed to extract the underline policy throughout this process.

4.0 Policy Builder UI



In the user interface of the Policy builder ,jsp dispatches the input stream of SOAP data asynchronously into Servlet Dispatcher (Upload Executor) which inturn uses a stub generated (using the Service WSDL) to invoke the web Service. Web Service interacts with the Policy Builder System and provides the derived output to the servlet. Sevlet will inform the jsp of the response through a the Ajax CallBack. Ajax call-backs are done through Yahoo yui wrapper scripts while WSDL2Java is used as the Client stub-generator for the Web service .An Upload Executor(ie:-Variant of a java Servlet) has been used intermediately to avoid cross-domain browser restriction errors.

Author : U.S.Wickramasinghe

gmail : mastershield2007@gmail.com