Chapter 6

Implementation

6.1 Introduction

The previous chapter it discussed about the analysis and design phase of the project. Also it presented the set of high level function requirements of the system. This chapter is based on the implementation process that was carried out during the project. This chapter basically discusses the major technologies used during the implementation of this tool. Detailed descriptions of the solutions provided for major issues occurred during the implementation process is also contained in this chapter. This chapter first discusses about the implementation of the XML Processing section of the tool. Then it moves on to the Agent Implementation where it discusses each message that is being used in the tool and then discuss about the logics of each agent that has been implemented in the system. Then it discusses about the Genetic Algorithm implementation and finally about the MADKit kernel and the ontology.

6.2 Major technologies used.

It has been used Java as the programming language to implement this prototype. The Eclipse programming IDE has been used to do the developments.

There are three major parts associated with the work carried out during the implementation of the proposed SOAP optimization system. According to the previous researches, the best way to process the XML SOAP document is by using the XML characteristics rather than using text characteristics. Therefore some effort has been done for identify the XML characteristics of the message being processed. For that xmlbeans[2] and the JAVA StAX[15] API has been used. Then Agent technology and Genetic Algorithms has been used for the pattern recognition of the identified characteristics and then template generation process. Finally some effort has been put on plugin this into web-services. For that JAX-WS standards has been used.

6.3 XML Processing

On a particular XML document, it can identify some common templates (a same data structure is being repeated with several data). XML schema document is the one that define the data structures correspond to a particular XML document. If we identify the XML schema or set of schemas of a particular XML then it is possible to identify the particular XML pattern which corresponding to those schemas. For example, when we generate a schema document for the XML document, then for each complex type of the schema there will be one to many instances of the data on the XML document. But all those instances follow the same pattern. Therefore identifying the schema document plays a major part of the optimization.

It has been used the xmlbeans[2] Inst2Xsd utility to generate the schema document from the XML SOAP message. But this schema is totally depends on the values on the given instance of the XML document. For example on a particular tag it contains numbers on this instance but it is possible to have alphabetical characters also on that field. On such incident Ins2Xsd will identify this field as a decimal field and generate schema. But such a schema will not be valid for the next XML document of same type. To overcome this issue it has been converted all simple types to be in string type by using SIMPLE_CONTENT_TYPES_STRING option in Inst2XsdOptions. Then the generated schema will be valid for any type of data being carried on those tags for other instances of the same type. The xmlbean's Inst2Xsd utility also tries to identify xml enumerations by examining the values for a particular tag on the XML document. This can also cause the schema document to be not to valid for another XML document of the same type. It is because the second document can have different values on those tags that were not in the original document and therefore not contained in the schema. As a fix for this enumeration issue it has been used the ENUMERATION_NEVER option on Inst2XsdOptions. There are three xml schema designs possible with xmlbeans. The schema design controls the way the xml schema document is formatted. The possible three designs are DESIGN_RUSSIAN_DOLL, DESIGN_SALAMI_SLICE and DESIGN_VENETIAN_BLIND. Since it is required to identify each XML complex type separately it has been used the DESIGN_VENETIAN_BLIND option for schema generation.

6.4 Agents Implementation

After the comparison of JADE and MadKit[18] it has decided to use MadKit as the agent development framework for this implementation mainly because of its support for broadcast messages within the local kernel as well as among the distributed kernel on the network.

There are seven agents implemented in this tool. Some are single instance agents which do a particular task and some are multi instance agents that do the pattern optimization work through Multi Agent Negotiation.

Following are the list of agents that implemented on this tool :

- AbstractSOAPAgent
- SOAPMessageAgent
- TemplateAgent
- TemplateManagerAgent
- TemplateBuilderAgent
- NodeAdministratorAgent
- TemplateOptimizerAgent
- ClusterAgent
- TextAgent

Following are the list of messages used on those agents:

- TemplateRequest
- TemplateResponse
- OptimizeRequest
- OptimizeResponse
- NewTemplateMessage
- AllTemplateRequest

Following are the list of messages specifically used for clustering:

- TemplateOptimizeRequest
- MembershipRequest
- MembershipResponse

- MembershipConfirmRequest
- MembershipConfirmResponse
- PerformNewClusterRequest
- PerformNewClusterResponse
- ClusteringEndMessage
- NewClusterPatternMessage

6.4.1 TemplateRequest

Template request message contains a set of key words extracted from a particular XML document. Those key words are generated using the XML tag names and the level of that tag in the document. If the same xml tag is being used in two levels of the document then it will be considered as two key words.

The level of the tag is being calculated as the child count to the root tag. This message is used to request for applicable Templates for a particular XML document.

6.4.2 TemplateResponse

Template response is the response message to template request. When a particular Template agent is capable of handling the Template request then it will send the TemplateResponse.

6.4.3 OptimizeRequest

Optimize Request is the message sent asking a particular agent to do the optimization for the current XML document. The complete XML document will be contained on this message as a string field. Also this message is addressed to a particular Template Agent.

6.4.4 OptimizeResponse

This message is the response for optimize request message. This message contains the optimized version of the original XML document as a string field on the message. This message is also addressed to a particular single agent. This is addressed to the sender of the original optimize request message.

6.4.5 NewTemplateMessage

After recognizing a new template from the system this New Template Message will be sent to all nodes of the cluster. This message is used to notify all the components of the system about the recognition of new template. This message contains the name of the template as a string field. The name will be the root tag name of the XML document. For SOAP message it will be the complex method name. This message also contains list of sub templates. Template identification process will generate a sub template for each complex type on the schema document of the XML. The list of those sub templates will be contained in this message.

6.4.6 AllTemplateRequest

This message is used to request all the templates cached on a particular node of the system. For example when a new web-service client is connected to the system, there may be some new templates that have been generated recently on the server which may have not transferred to this new client. But server will not aware that the new client does not have the knowledge of those templates and server will optimize the message using those templates also. It is the new client's responsibility to gather the knowledge he missed. Therefore every client will send this message to server and request for all templates cached in there.

6.4.7 TemplateOptimizeRequest

When the system continues on processing XMLs it also gathers some statistics of frequently processed data and keep them to further optimization of the templates. The Template Optimize Request is used to start the template optimization process. When template agent receives this message it will use currently stored statistics and start the data clustering process.

6.4.8 MembershipRequest

This message is used in data clustering process. This message contains the text that is needed to get the membership of the cluster. This message will be used to request the membership of a particular cluster.

6.4.9 MembershipResponse

This message is the response for the membership request. This will contains a score calculated by a particular cluster for the given text on the membership request. This score will be used to make the decision whether to join with that cluster or not.

6.4.10 MembershipConfirmRequest

When it is decided to join for a particular cluster base on the scores received on Membership Responses, text agent will send this message again. This message also contains the particular text on a string field. This message is addressed to the particular cluster agent.

6.4.11 MembershipConfirmResponse

Membership Confirm Response contains a Boolean value will contain in this message. This will carry the information that whether the Membership Confirm Request is accepter or not. For a text to be a member of the cluster, the cluster must agree to the Membership Confirm Request and send the Membership Confirm Response with a true value.

6.4.12 PerformNewClusterRequest

This message is used by text agents to inform other text agents that it will be going to perform a new cluster. It is required perform a new cluster when a text agent is unable to find a membership in the current existing clusters. But when there are multiple agents who unable to find clusters and going to perform new cluster then those new clusters will be redundant. Therefore it will be only allowed one text agent to build new cluster at a given time and for that all other text agents who were unable to find clusters must agree. The Perform New Cluster Request is used to inform other text agents that it is going to perform a new cluster.

6.4.13 PerformNewClusterResponse

This message is the response message to Perform New Cluster Request. This message will contain a Boolean value specifying whether cluster creation is allowed or not.

6.4.14 ClusteringEndMessage

Clustering End Message is used to notify all clusters that all text agents have finished their lifetime and therefore it is not required for wait for messages.

6.4.15 NewCluaterPatternMessage

This message is used to inform the Templates the findings of the clusters. This message will contain a key to identify the correct sub template and the field on which the clustering has been done. And also this message contains the new static content pattern which is the result of clustering.

6.4.16 AbstractSOAPAgent

In MADKit environment agents are created as Java Classes. AbstractSOAPAgent is the base class that is used to generate every other agent in the system. This class is inherited by the Agent class of the MADKit. This class carries a protected method that can be used to register other agents to their corresponding community, group and role. Because MADKit work on the community/group/role model, it is essential to register to desired group at the activation of every agent. This will enable the agents to fetch messages from the message space that is corresponding to the community/group/role on which they are working on.

6.4.17 SOAPMessageAgent

When a new SOAP message is generated from the web-service and it is required to send it to the client side, it will generate a SOAPMessageAgent. The duty of this agent is to get the optimization work done.

First of all the SOAPMessageAgent will generate the keyword list for the given XML document. For that it will use the XML Utility class. This will traverse the XML document by using the StAX API and generate the keyword list using the tag names on the XML document and their levels. The level of the tag is calculated using its child depth from the root tag of the XML document. The root tag is considered as the level 0.

Then this agent will broadcast a TemplateRequest message to all Template Agents using the template community/group/role. This TemplateRequest will contain the extracted keyword list. After that the SOAP Message Agent will wait for Template Responses. It is not guaranteed that always it will find a possible Template to optimize the XML message. Therefore this agent will wait for only 100 milliseconds. If it does not get a response within that period of time agent will decide there are no possible templates on the system to optimize the current XML document and therefore it will return the XML as it is without optimization. Then the raw XML will be transferred through the network. Meantime the SOAP Message Agent will launch a new Template Builder Agent with the original XML document and the local tag name of the web service method as the parameters. This is to generate the possible template for this new XML document.

If the SOAP Message Agent receives a response to the Template Request within 100 milliseconds then it will accept that response and will send an OptimizeRequest. It will include the complete XML document on that optimize request and it will be directed to the template agent who generated the template response.

SOAPMessageAgent will receive an OptimizeResponse as the reply for the OptimizeRequest and this response will contain the optimized SOAP message content. Then SOAP Message Agent will return this optimized message as the output.

This tool has to be work as an additional plugin to the web-service environment. Therefore the agent based implementation has to be work inside the non-agent supported environment. The SOAP Message Agent is the interface that the processing of message will enter into the Multi Agent based environment and the return output from the Multi Agent Based environment to web-service system. Therefore the SOAP Message Agent will use the waitNextMethod to deal with agents and also Java wait/notify system to deal with non-agent base code.

6.4.18 TemplateAgent

Template Agent can be considered as the resource agent for the template finding process of SOAP message optimization lifecycle. Template Agents will be created for each template on the ontology at the initialization of the system. Then it will load all corresponding data of its own from the ontology. Template Agents will be also created when system identifies new templates or when system receives new templates from other system connected to it.

Template Agents will listen for the incoming Template Requests. When a message appears on the message space corresponding to its community/group/role it will fetch that and process. Template Agent will use the waitNextMessage method of MADKit framework for this purpose. The wait time is unbounded and the agent exists on the system forever until the system ends.

When agent starts, it will generate the keywords list that it supports and will be kept for TemplateRequest processing. When agent fetches a TemplateRequest from the message space it will compare the keyword list contained on that with the agent's supported keywords. If all the keywords on the request message is contained on the supported word list then it will send a TemplateResponse. If there are keywords on the request that not supported by the template then it will not send any replies to the message.

Based on the Template Responses sent, agent will receive the Optimize Requests. When the Template Agent receives the optimize request it will process the XML message on that request and use the XML Utility class to generate the optimized message. The XML Utility class will use the StAX interface to traverse through the XML document and for each tag on the XML document it will first identify whether it is a sub template within the template or it is a variable within the current processing sub template. Then it will either call the recursive method for process the next sub template or iterate through tags to extract the dynamic content corresponding to this sub template.

After Optimized message is extracted the Template Agent will create an Optimized Response message and send with the extracted optimized SOAP message content on it.

Template Agent also collects statistics of the dynamic content being processed. This statistics will be used to further optimization of the template by identifying common patterns of those dynamic text and classify them also in to static category. By this it will keep on improving the static content templates.

The Template Agent also listens for Template Optimize Request. Upon receiving of this message Template Agent will start the template optimization process. It will use the statistics it has used up to now and start the clustering process. First it will start cluster agents for each fields for which statistics has been collected. Those agents will be registered on separate groups such that on clustering process it will receive messages corresponding for its own field data. For that it will use a key generated combining the complex name of the sub document and variable name as the group of the cluster agent. All text agents of the same variable also will be registered against the same group and therefore when a text agent broadcast a message it will receive for only the cluster agent corresponding to the same field. By doing this it will reduce the message processing overhead because only relevant agents will receive the message.

After starting the clustering process the Template Agent will clear its statistic store and start to store fresh collection of data.

6.4.19 TemplateManagerAgent

Template Manager Agent is a single instance agent on the system. It will register on the template manager distributed group and wait for New Template Messages.

At the initialization of the system Template Manager Agent examine the ontology and launch Templates agents for the number of templates contained on the ontology. Then it waits for messages from the other system. Because the template manager group is a distributed group this agent can receive messages from the agents on the kernel it resides as well as other kernels that running on other computers but linked to this kernel by MADKit communicator utility.

Template Manager Agent will listen for New Template Messages and it will add the receiving new templates for its cache. It will also launch new Template Agents with correspond to those new templates. Also this agent will update the ontology with the new knowledge received.

Template Manager Agent will also serve for AllTemplateRequest message. It will send NewTemplateMessages as the response to the AllTemplateRequest. Those messages will be directed to the sender of the request (the agent who requested the templates).

6.4.20 TemplateBuiledrAgent

Template Builder Agent is launched by the SOAP Message Agent when it is unable to find a Template to optimize its XML message. Responsibility of the Template Builder Agent is to generate the initial template using the XML document. For this it will first generate the schema document of the xml by using xmlbeans. Then it will generate the sub templates for each complex type on the schema. Finally it will again scan through the XML document to generate the static content details and calculate the tag levels.

6.4.21 TemplateOptimizerAgent

Template Optimizer Agent is also a single instance agent. Its responsibility is to notify Template agents to start the clustering process. This will be done by broadcasting the Template Optimize Request for the template agent group on every 30 seconds.

6.4.22 ClusterAgent

Cluster Agent is created as the resource agent on the data content data clustering process. This clustering is done for identify the common static content of data which are previously identified as dynamic. Therefore output of the clustering process is some set of regular expressions which represents the common text of the members of the cluster. Cluster Agents are created against a particular group. It has to deal with the text agents who are also registered for the same group only.

At the start of the agent lifecycle, there are no data associated with the cluster and any Text agent is welcome. The cluster agent will accept the MembershipRequest message and collect them into a Hash Map. When it received more than 10 such requests or wait idle for 500 milliseconds and there are no more member requests, then it starts to process the current membership requests collection. Such approach is taken for produce a better generalized cluster by considering more samples in the first stage but also not to keep text agents waiting for longer time.

The cluster agent will create a distinct list of the texts that were received on the membership requests. Then if the list size more than 2 it will call Genetic Algorithms utility function for get the optimum text list to generate the cluster pattern. Otherwise it will use all available text data and generate the template pattern.

There are two utility classes written for this purpose. One is the GAUtils class which handles the Genetic Algorithms part. The Second it the StringUtils class which process two given strings and generate the regular expression which is common to both strings. And also it calculates the membership values of two given strings.

GAUtil class handles the Genetic Algorithm functionality. The static method provided on this class will take a list of string as the input and will produce the optimal string list to create the cluster as the output of the method. The JGAP [27] package has been used to create the genetic algorithm functionality. For the given list of String it will find the subset of strings that is more suitable for generate the cluster pattern. Those strings will have more common substrings on them than when compared to other strings of the list. The implementation of the Genetic Algorithms will be discussed on a separate subsection of this document.

StringUtils class handles the string processing functionality. There are three basic static functions defined in this class. Those are “getCommonRegx”, “getMembership” and “getPatternMembership”. It has been done some studies on several string comparison techniques such as Levenshtein Distance [25] and Needleman-Wunsch distance [25] and implemented the getCommonRegx function. In this function it compares all the letters of the first string with all the letters of the second string as it is in those two string distance calculation methods. But instead of returning the distance of those strings, this function is producing the most restrictive regular expression that applicable for both of the given strings. The generated regular expression is in Java accepted regular expression format and therefore can be easily used with Java.

After getting the most optimal list to generate the cluster pattern the Cluster Agent will create the common pattern correspond to the list by using the getCommonRegx method of the StringUtill class. Then it will calculate the membership value of the each text of the membership request message list and then respond to the senders of those messages with a Membership Response message. The membership response message contains the membership value as the score on the message. The decision to be a member of that particular cluster is taken depending on that score. When it is decided to be a member of that cluster, it will be requested again by a Membership Confirm Request. This message is directed to a particular cluster agent only.

On receiving of Membership Confirm Messages also, if it has not yet built any cluster pattern, the cluster agent collects the first ten messages in to a list. Then it starts to process this list as it done before to the other list. If there are more than two distinct texts then it uses the Genetic Algorithms to get the appropriate list for create the cluster pattern. Once the pattern is created the Cluster Agent is not going to change it unless the energy of the cluster will be increased by doing so.

6.4.23 TextAgent

Text Agents will be created as the request agents of the data clustering process. The objective of the text agent is to find a suitable cluster. When a text agent is created first it broadcast a Membership Request Message to the cluster agent group. Then wait for the Membership Responses. It collects the set of Membership Responses that receives within two seconds and then evaluate the response collection.

Each response contains a score given by cluster agent to the text contained in the text agent. The response with highest score will be chosen as the best cluster to join with. Then text agent sends a Membership Confirm Request. This message is directed to the particular cluster agent who has offered a higher score. If the text agent receives the Membership Confirm Response which is accepted by the cluster agent, then text agent will end its lifecycle because its job is completed.

If the text agent receives Membership Rejected Response, then it has to find another cluster. First it will analyze the rest of the messages in the Membership Response list. Then if such cluster cannot be selected then the text agent will decide to create a new cluster. For that first of all it has to launch a cluster agent. But if there are multiple text agents trying to create clusters at the same time that attempt may be redundant because some of those texts can be put in to same cluster. To avoid such redundancy, the text agent that decides to create a cluster agent will wait a randomly selected time for Perform New Cluster Request message. If it receives such a message that means another text agent is also tries to create a cluster and it must allow the other agent to create it. Therefore text agent sends the allowed response for that message and will not create new cluster this time. But if it does not receive such a message on the waiting time it will broadcast the Perform New Cluster Request message and wait. If it receives not allowed responses then it will decide not to create the new cluster. But if it is received at least one response with allowed flag on, then that text agent will create a new cluster agent.

Agent collaboration diagrams are shown on the Figure 6.1, Figure 6.2 and Figure 6.3.

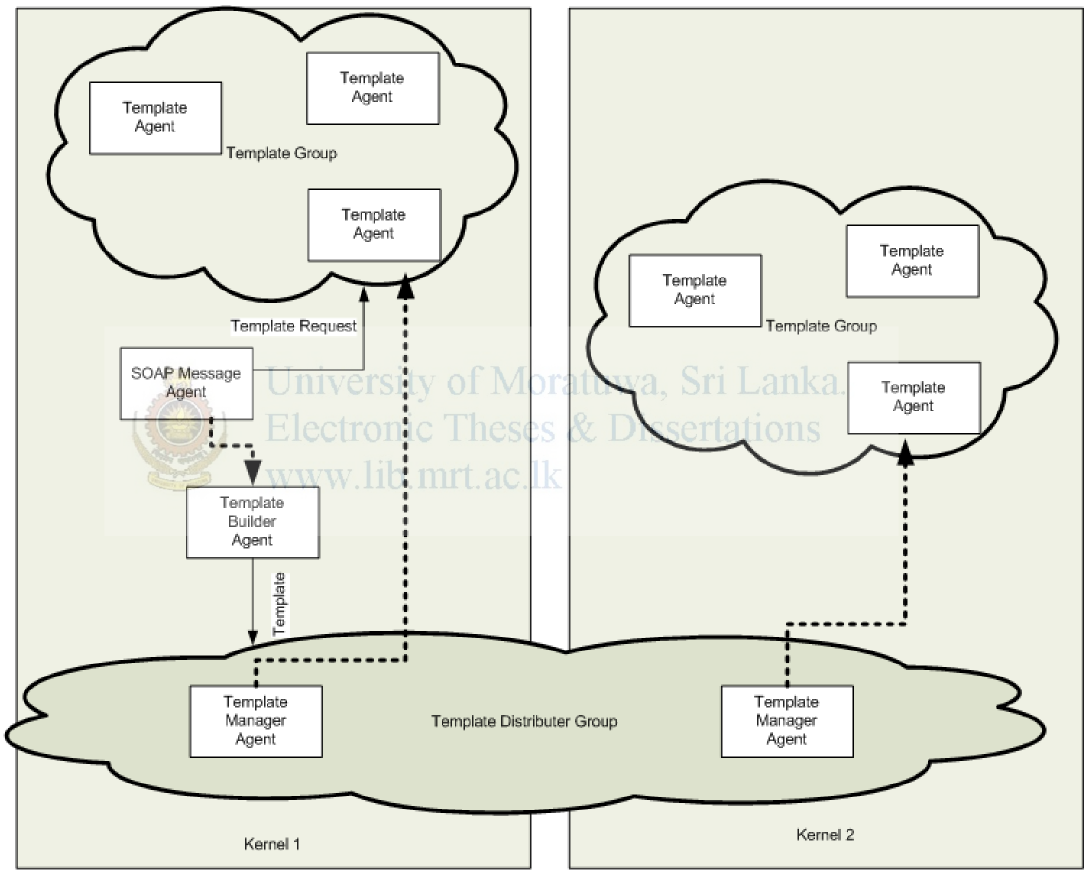

Figure 6.1 : Agent collaboration diagram when new type of message arrives

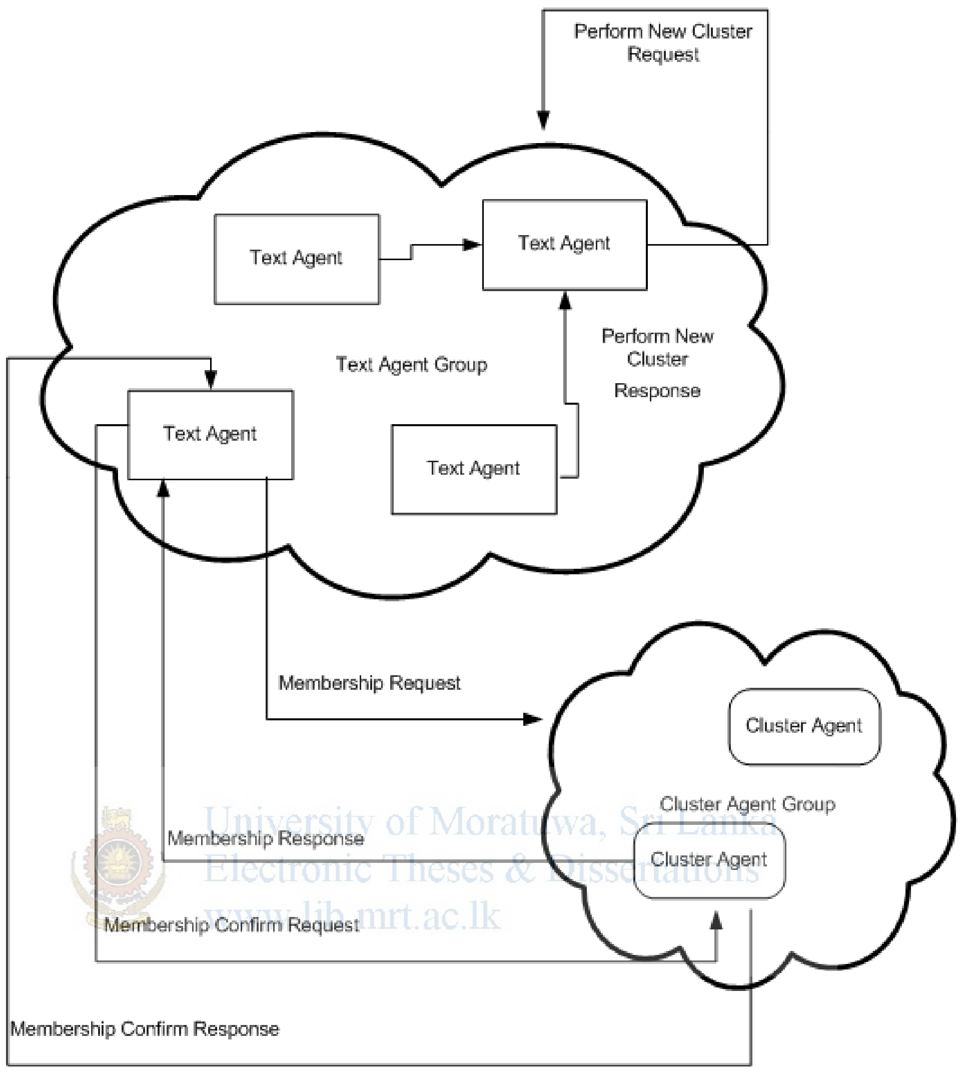

Figure 6.2 : Agent collaboration diagram during data clustering

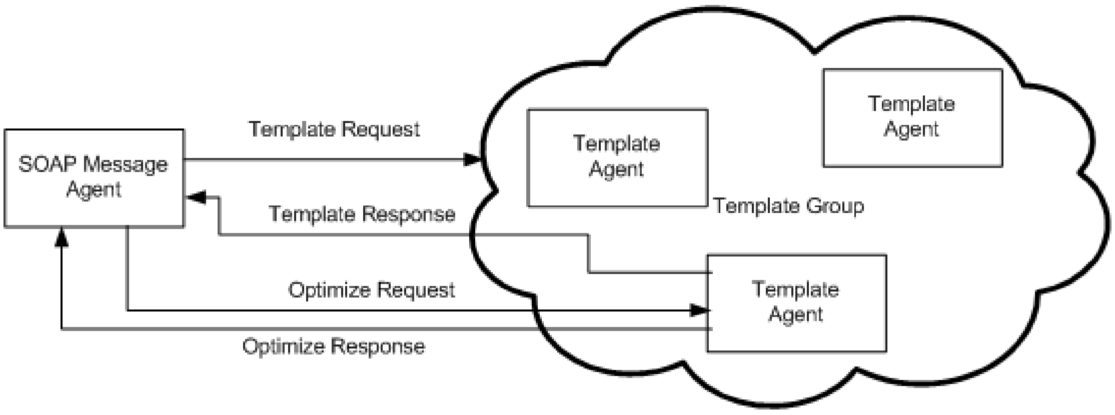

Figure 6.3 : Agent collaboration diagram when pre defined template exists

6.5 Genetic Algorithms

The JGAP[17] Java library has been used for implement the Genetic Algorithm processing on this SOAP Optimizer tool. JGAP is an open source project which is distributed under GNU Lesser Public License.

In JGAP it is required to implement the fitness function as a separate class which is has to be inherited by the FitnessFunction class of the JGAP. Then it is required to create a configuration object to which it has to set an instance of this fitness function. Then it has to create a sample array of gens and using that it is required to generate a chromosome. For our purpose it decides the length of the chromosome by considering the number of strings on the list. If the number of strings on the list is more than 7 then it will use 7 as the maximum chromosome length. Otherwise it will use the number of strings in the list as the chromosome length. The values for this calculation have been decided as a result for several test carried out using this JGAP library. These vales will give the optimal result in less time.

There are several pre-built gene types available on JGAP. For example some of the pre-built gene types are, BooleanGene, CompositeGene, DoubleGene, IntegerGene, NumberGene and StringGene. For this implementation it has been use the IntegerGene type. For the given list of string a gene will represent the index of a particular string on that list. Then the chromosome will represent a set of strings by using their indexes. Even though the chromosome length is fixed the optimal set of strings on the list can be less than the chromosome length. Therefore all the genes on the chromosome do not need to represent a string in the list. When creating the IntegerGene it is possible to specify the minimum value and the maximum value that the gene can have. To allow the possibility of a particular gene not representing a string on the list, it has been set the maximum value of the gene to a larger value than the length of the string list. By doing that, it will allow the gene to have a larger value which is greater than the size of the string list. When a gene contains a greater value than the size of the string list it is considered that such a gene will not represent a string on the list and therefore chromosome is representing the less number of strings than its length.

The population size is also calculated based on the size of the string list. The population size is taken as the square of the size of the string list. When the string list is larger, then the population size of the genetic algorithm is also become larger. By doing that it will generate a more distributed population and it will always converge to a solution.

The number of generations to be computed is set as a fixed value. For finding the set of optimal strings it has been chosen to compute up to 25 generations.

The fitness function of the Genetic Algorithm is implemented as a separate class named as StringPatternFF. It is inherited from the FitnessFunction class on JGAP distribution. It has been implemented the evaluate function on this class which take a chromosome as the input and return a double value as the fitness of the chromosome. For each chromosome given as the input to this function, it first identifies the list of distinct strings represented by the chromosome. The genes that does not represent a string (has a higher value than the size of the string list) will be eliminated on this stage. Then if there are more than one string on the list it will first generate a regular expression which is common to those string and then calculate the membership value of each string with that common regular expression. Then the total will be given as the fitness of the chromosome.

6.6 MADkit kernel

Usually when an Agent framework is used to implement agents, when run the application it should start with boot class provided on the agent development framework. Otherwise it will not be able to launch the Agents. But this tool has implemented as a plugin to web-service applications and therefore it cannot be use the boot class provided by agent framework to start the application. But in MADKit environment it has provided the facility to create our own booter class by extending the AbstractMadkitBooter. This will start a madkit kernel on its constructor. That kernel can be used to launch agents. By this method it is possible to use the MADKit framework without altering the main class of the application.

6.7 Ontology

For this tool we have used simple text files as the storage mechanism for ontology. There is a predefined parent directory for the ontology on each client modules and service modules of the system. Each template agent will create an own subdirectory on that and store their templates on that sub directory. Then when the system is restarted the already identified templates will be loaded from ontology by creating Template agent per each subdirectory in the base ontology directory. When a new template has been generated from the system then a new Template agent will be activated for that new template and also a new subdirectory will be created on the ontology for store the new template data. Sample format of a template will be in Appendix C.

6.8 Clustering Viewer

The clustering viewer module will provide the facility to view the agent negotiation happens on data clustering process. It will provide a graphical view as well as a text based view. The graphical view will show the current status of the clustering by using some set of shapes to represent each agent. It will also display the communication happening between agents using arrows. When the agent ends its live the symbol that represents the agent will be removed from the panel. By this it will be possible to see the current agents of the system. Please refer the Appendix B for more details and the screens of Clustering Viewer.

6.9 Summary

This chapter has discussed about the implementation process of the SOAP optimization system. This chapter presented the main technologies used to implement this system, the agents implemented in the system and messages used by them. It also discussed how genetic algorithms applied to improve the clustering process. It has used some agent collaboration diagrams to illustrate the agent activities and agent message passing in detail. It has been also presented how the system is handled the MADKit kernel such that system will work as plug-in on existing web-service application servers. Finally it was discussed about the ontology and how the ontology is implemented on this system. Next chapter it will be discussed in detail as to how this tool can be plugged into existing web-services.