Abstract

Law Governed Interaction context allows interaction within a community based on an explicit policy which is public and known by each agent, called the law of the community. A community is distributed in nature, no global information is maintained in any central place. Agents may operate independently, without any concern about other agents. Furthermore, there is no central mechanism for one agent to find out of the existence of the other agents which belong to the same community. The agents, although born in equal conditions, may want to interact on asymmetric basis. Not all the agents are equal. The agents act differently and they have different powers and privileges. There is no direct way to achieve such properties in a community governed by a low $L$. The secretary agent is a mechanism that attempts to solve the above problems. The secretary agent acts as a central server that manages the content of a group of agents and, in some way, its logical state. Thus the group of agents is no longer a simple community. Instead it is a subgroup composed of agents of some community that have knowledge of the membership of this group and its logical functionality.

1 Introduction

Law Governed Interaction context allows a group of a distributed heterogeneous number of agents, called a community, to interact with each other by an explicit policy. This policy which is public and known by each agent is called the law of the community. Each agent obeys the rules of the law, thus the law introduces some regularities over the communication, and furthermore, over the behavior of each agent.
A community is distributed in nature so no global information is maintained in any central place and the agents may operate independently, without any concern about other agents. Furthermore, there is no central mechanism for one agent to find out about the existence of the other agents which belong to the same community. The Moses system [1] has the notion of a trusted controller that holds the control state of the agent and enforces the communication of the agents according to the law L of the group. There is a large number of controllers. Each agent has to connect to a controller under a chosen law and, as a result, it will belong to a community.

As a subsequent result of all the agents operating under the same law, all the agents become symmetric. Each of them has equal initial conditions and equal abilities to operate. It is the law of the group that assigns the same control state to the agents and enables them to act in the same manner and with the same privileges. The above behavior may not be satisfactory for a broad range of application. The agents, although born in equal conditions, may want to interact on asymmetric basis. Not all the agents are equal. For example, in an enterprise there is one manager, several deputies, and regular employees. They act differently and they have different powers and privileges. There is no direct way to achieve such properties in a community governed by a low L, because all agents receive the same initial control state and they obey the same set of rules equally.

The secretary agent mechanism attempts to solve the above problems. The secretary agent is an agent which operates under a different law, the law of the secretary (or secretary law). It acts as a central server that manages the content of a group of agents and in some way its logical state. Thus the group of agents is no more a simple community, instead it is an explicit group of agents that is called a group. In such a group one agent may find the exact membership of the group and, at some point, the state of the whole group, depending upon the law.

There are several main features the secretary attempts to solve: a centralized mechanism by which one agent may find out about all the agents which operate in the same group and implicitly their names. Thus the secretary may act as a name server for the group. Another issue the secretary attempts to solve refers to the persistence of the control state of one agent. In the condition of mobility of the agents and discontinuous operation, as long as an agent disconnects itself with its host controller, it may lose its name and its control state. The secretary acts as a control state repository, providing the agents with the opportunity of retrieving their saved control state
at some later moment. Another feature of the secretary is that of initial control state assigner: each agent that joins the group receives an initial control state according to a control state configuration file and some particular role requested by the agent. Several restrictions may apply to this assignment: the request of a particular role may be conditioned by the presence of some terms in the previous control state of the agent and only a maximum number of agents may ask for a particular role and, therefor, for a specific control state. The above functionality, though is not a general mechanism - tries to enforce a particular behavior for a particular group in a non centralized environment.

The remainder of the paper is organized as follows. In Section 2 we describe the notion of group and the main functions of the secretary. In Section 3 we describe additional features to control the membership of a group and to create a role based structures of the agents. In Section 4 we give a brief discussion of how an agent can make its Control State public, and further, in Section 5 we discuss how to make the Control State persistent for a later retrieval. We conclude the paper in Section 6, where we present further research and open issues. In Appendix A we will present the law frame the agents and the secretary operate under, and in Appendix B we present the message interface between the agents and the secretary along with a description of each particular message.

2 Basic Secretary Functions

As we have defined before, a community is a group of agents that operate under the same law. A group is formed by the agents which belong to some community that have made their name public to the same name-server: the secretary of the group. The secretary does not belong to the group. The secretary operates under a secretary law which is different than the law of the group. The secretary only provides a common place where one agent may find useful information about the agents operating under its law. Thus, the secretary main function is to form and to start a group.

An agent declares its intention to belong to a group through the join operation which sometimes is called the join of the group or the join of the secretary. As a result of a join an agent can be either accepted to the group or rejected. When joining the group, the agent should provide the secretary with a chosen nickname. We use the concept of nickname for several reasons.
A name in Moses is composed of a local name, which is chosen by the agent before the connection time, and a domain name which is the name of the controller an agent connects with. Because of this naming scheme, once an agent disconnects from its controller, its name ceases to exist or it may be taken away by other agent, despite the fact that the disconnected agent may continue to operate off-line. Moreover, if it changes the controller the agent receives a different name that makes the communication with other agents impossible. Thus, the basic Moses mechanism does not provide any temporal persistence in regard to the names of agents. In order to solve the problem, the secretary mechanism introduces a new concept: the nickname of an agent. The nickname of an agent is different from its name, and the secretary holds the correspondence of the nicknames and the names of the agents which form the group. The nickname is a name chosen by the agent when it decides to belong to a group. The nickname has the property that it is unique over the group. Therefore when the agent changes its controller, its name is still known by the other members of the group by the mean of the nickname and by issuing a request to the secretary.

The opposite operation to the join of a group is the quit operation. When an agent decides to leave the group it simply issues the quit and the secretary will release all the information associated with its nickname. We will talk more about the information stored along with the nickname of an agent in Section 4. Another way an agent can lose the membership of the group is when another agent issues the remove command. The remove command has as argument the nickname of the agent to be removed, while quit doesn’t have any argument. Both commands have the same effect upon the subject agent, and the secretary doesn’t make any assumption regarding the legitimacy to issue the remove command. Instead, the law of the group is responsible to limit the issuing of remove by ordinary agents. The law should be constructed in such a way that it should allow only some privileged or responsible agents like a manager or a director to issue a remove message. In a well defined law, a remove message may leave the controller of some agent only after a checkup of the Control State. As long as the secretary receives the remove command, it processes it immediately without any verification.

As one goal of present work, the secretary provides a mechanism by which the agents may find out about each other. This facility can be achieved through the use of several commands. In the context one important command is the who command. When the secretary receives the command it simply replies with the list of all active agents. As defined in the current work, an
agent becomes active after the secretary accepts a join command that was issued by that agent. The agent remains active until it issues a quit command or another agent issues a remove command having as subject the first agent.

Another useful command is *where* command. An agent uses this command to find out the name of some particular agent when it only knows its nickname. As we were talking before, the names in Moses are only lasting during the connection session with the host controller. Many times, this policy it is not convenient. It is very likely that one agent may log in with the Moses system multiple times, not always with the same controller. Since the nickname of one agent is not assigned by Moses most of the time and it is not bounded by the host controller of the agent, the nickname it is more likely to be persistent public and known by the other agents. The secretary provides the way for agents to find out the name of a particular agent when its nickname is known.

The *howMany* command completes the set of commands that provides information about the membership of the group. It simply returns the number of agents that are active when the secretary of the group receives the command. It should be pointed out that this command and the *who* and *where* as well may be issued by any agent that operates under the law of the group, not necessarily by the members of the group. It may be of particular interest for one agent to ask the secretary how many agents are active at some moment of time, before deciding to join the group.

A particularly powerful command provided by the secretary in the basic mechanism is the *broadcast* command. When an agent issues this command it passes the message to be broadcasted to the secretary, and the secretary subsequently exports the message and the name of the sender to all the active agents in the group. In order to protect the group by unwanted messages, only the active members of the group may issue this command.

In the current chapter we discussed issues related to the membership of a group. As we have seen, an agent voluntarily joins a secretary and one agent may become a member of many groups in the same time. But how can we individualize a particular group? The Moses mechanism provides the mechanism of *fingerprinting*. The *fingerprints* of one agent is one random number assigned to the agent upon request. As long as a group is defined by a secretary, we can say that the fingerprints of the secretary may individualize the group. The law of the secretary currently exploits this Moses provided facility. Along with the acceptance message from a join operation, the agent receives the secretary fingerprints that are saved in its Control State. Those
fingerprints are marks having the significance that one agent belongs to a particular group.

Preamble:

Portal(secretary,URL(http://host:port/secretary.law))
initialCS([initial CS])
PolicyName(member)

R1. imported(S,secretary,reply(join(F),accepted(N)),X)
    :- do(deliver(S,reply(join,accepted(N)),X)),
       do(+secretaryfp(F)).
    when the acceptance message arrives, the secretary fingerprints are added to the Control State

R2. sent(X,groupCommand(quit),S) :- do(-secretaryfp(F)),
    do(export(X,groupCommand(quit), S, secretary)).

R3. sent(X,groupCommand(remove(N)),S) :- do(-secretaryfp(F)),
    do(export(X,groupCommand(remove(N)), S, secretary)).
    when the agent quits the group, it has the fingerprints removed

R4. sent(X,M,Y) :- secretaryfp(F)@CS, do(forward(X,fm(M,F),Y)).

R5. arrived(X,fm(M,F),Y)
    :- secretaryfp(E)@CS, E = F, do(deliver(X,M,Y)).
    only the messages that contains the fingerprints of the same secretary may pass through the law

Figure 1: Example Law that restricts the interactions only inside the group. The law should be completed with the rules in Appendix A that regulate the interaction with the secretary.

The example in Fig. 1 shows how a group may have a particular behaviour based on the fingerprints of the secretary. Under the law of the group, when the agent receives the acceptance for the join, it saves the fingerprints of the secretary. The fingerprints subsequently prohibit any communication with nonmembers of the group (but still members of the same community).
3 Membership and Structure Control

In the previous section we have seen the basic functions the secretary provides the agents with. Sometimes the above functionality turn out to be not flexible enough. Moreover, sometimes it may lead to the overload of the secretary. This may be the case when arbitrarily large number of agents decides to join the same group and when no agent quits or gets removed from the group. The set of commands described below control the membership of the group by limiting its size. Since the secretary is a general mechanism, it does not have any information regarding the appropriate size of the group and should not make any assumption about when to accept and when to reject some particular joining agent. The law of the group should decide what is the limit size of the group, the secretary is only the central place where it is enforced.

One important membership control command is close. As a result of receiving this command, the secretary close the group immediately, not allowing any new agent joins. The opposite command is open that opens the previously closed group. Both commands acts immediately when received by the secretary. Though powerful, the above commands can not provide a way to set the maximum size of the group to a a priori desired value. Let’s suppose that the manager of the group is the agent that is responsible for opening and closing the group. In order to set the size of the group to some particular value, the manager should issue the howMany command, and, when it reaches the desired value, the manager should issue the close command. Nobody may insure that another new agents joined the group meanwhile, thus the size of the group can not be preset by the above mechanism. It may be crucial for some applications to have the number of agent for example even.

In order to prevent the above behavior, secretary provides the group with the setGroupSize command. This command has two parameter, the minimum and the maximum watermark of the group. When the secretary receives the setGroupSize(min, max), the group is closed automatically when the number of active agents reaches the max watermark, and it will stay closed until the number of active agents will drop under min watermark. Then, the group opens automatically. If an agent attempts to join the group while the group is closed it will be rejected with the message GroupClosed.

All the commands above control the membership of the group by restricting it to a particular size. But most of the time, the requirements of a par-
ticular group refer to the structure of that group. The secretary introduces a new concept: the role of an agent. The role of the agent is the position an agent has inside its group. One agent may play different roles based on different Control States. For example, in an enterprise, some people has the role of the manager, some the role of accountants and some people just a simple employee role. The secretary provides a command that assigns specific roles to requesting agents. This command is joinAs. As we mentioned before, based on the law, all the agents in some community receive the same initial Control State. The secretary may assign different Control States to agents who decide to join the group. The difference is based upon the role requested by each agent, the secretary assigns different Control States corresponding to different roles.

How to prevent all the agents to attempt to get the same role? There are several possible ways of doing that. One way is by the mean of the law of the group. In order to request some role, an agent should provide some term role(role) in its Control State (either by the mean of certificates or by other mechanism) when it attempts to join the group. Another way is to allow only a limited number of agents to ask for a special role. This limited permission is enforced by the secretary. Thus, the secretary should provide a correspondence between some roles and initial Control State of the agents to join the group. An example of this correspondence can be seen in Figure 2. Another way to prevent an agent to get a role without permission is by using the password mechanism. When issuing a joinAs command, beside the

```
role(anybody)
cardinality(-1)
password(none)
CS([policyName(member),portalTable([secretary]),
   authorityTable([]), role(anybody) ..another Control State terms..])
role(manager)
cardinality(1)
password(managerpswd)
CS([policyName(member),portalTable([secretary]),
   authorityTable([]), role(manager) ..another Control State terms..])
```

Figure 2: Example of role file.
requested role the agent should send the corresponding password to that role. The secretary has a role file that holds the correspondence for the supported roles, their cardinality and password and the assigned Control State. An example of such a file can be seen in Fig. 2. In that example, it can be seen two roles. One role is a generic anybody role, it has the cardinality (-1) which means that an unlimited number of agents may ask for that particular role, and an a priori known password of “none”. The other role is a manager role with cardinality 1 and password “managerpswd”.

4 Globalization of Control State

Another important issue the secretary deals with is that of the Globalization of the Control State. The information the secretary provides the agents with only refers to the naming scheme and control information. In order to understand the complex functionality of a particular group, one should have knowledge of the Control State of the members. The Control State of the members ultimately enforces the behaviour of each agent and, subsequently, of the entire group. The secretary provides a way for the agents to make their Control State public.

An agent may make its Control State public by issuing the command savePCS - which stands for save-public-Control State. When the agent issues this command, the Control State of the agent is sent to the secretary along with the command, and the secretary holds it in a storage corresponding to the nickname of the agent. Once this Control State becomes public, all the members of the group may issue queries to retrieve it. The command that implements this queries is whoIs command, which takes as argument a Control State term. The secretary replies with a list of agents that have the particular queried term in their Control State. Obviously, the search is restricted among the agents that have made their Control State public. It can be seen that this command doesn’t simply disclose the entire Control State of a particular agent, it just confirms the existence of some particular term.

The agents make their Control State public on a volunteer basis. Unless otherwise specified, no one enforces an agent to disclose its Control State at all or at some particular moment. This may be a disadvantage as well, since the Control States held by the secretary may become obsolete in a very short time, and they can mislead the other agents. More than this, they simply
Preamble:

Portalsecretary,URL(http://host:port/secretary.law))

initialCS([initial CS.])

PolicyName(member)

R1. imported(S,secretary,reply(join(F),accepted(N)),X)
    :- do(imposeObligation(timeout,0,10)), do(deliver(S,
        reply(join,accepted(N)),X)), do(+secretaryfp(F)).

    when an agent becomes member of some group, it should make its Control
    State public it 10 seconds

R2. sent(X,some message,Y)
    :- secretaryfp(F)@CS, do(+term(some term)), do(forward),
        do(export(X,groupCommand(savePCS(newCS)), secretary
        address, secretary)).

    whenever the Control State changes because a new term is added, the agent
    should make its new Control State public to the secretary

R3. arrived(X,some message,Y)
    :- secretaryfp(F)@CS, do(-term(some term)), do(forward),
        do(export(X,groupCommand(savePCS(newCS)), secretary
        address, secretary)).

    the same as above whenever an old term is deleted

R4. obligationDue(timeout)
    :- do(export(X,groupCommand(savePCS(newCS)), secretary
        address, secretary)).

    when the obligation due occurs, the Control State is sent to secretary

Figure 3: Example Law that enforces the members of the group to sent their
Control State within a small period of time after joining the group.
put a futile load on the secretary, which remove the Control State only when the agents quit the group (or are being removed from the group).

In fact, it is not the secretary that should decide when the agent are supposed to save their Control State. Once again, the secretary plays a passive role only. It is the law of the group that should enforce the moment when the agents have to make their Control State public. A possible example of that kind of policy can be followed in Fig. 3. Whenever an agent is accepted into a group, it receives an obligation to make its Control State public within a small amount of time. After the amount of time expires, its Control State is sent automatically to the secretary. The Control State is sent to the secretary as long as an “important” term is changed (the law is the one to decide which term is important).

5 Persistence

In Moses system the Control State of the agents is held on per controller basis, as long as an agent leaves its controller, its Control State becomes lost. Thus, the Control State is held in a so named per session basis. Obviously the above functionality may not be satisfactory for a broad range of applications.

The secretary provides the agents with a mean to save their Control State in a permanent way. The saved Control State becomes secret and the CS time-persistence is gained even when the agent happens to quit the group or moreover, the community. Agents may need such services because of reliability as long as mobility reasons. In order to provide a stable storage for the agents of the group, the secretary enables them to save their CS independently of their name or nickname. This stable storage enables a particular agent to cease the interaction with the group or even with its controller for arbitrary long periods of time. An agent saves its Control State based on a realname scheme. The saving scheme obeys the law of the group. Only when the law allows, an agent may save its control state. For a proper functionality of the group, when an agent decides to retrieve its control state, it should prove that it is indeed the same agent that has saved the control state previously. This decision is based upon the mean of the realname. The realname expresses the certification that an agent is indeed the one who pretends to be. The certification is done by the mean of certificates. Whenever an agent brings up a valid certificate that states the identity of the owner, the law adds the term realname(realname) in the Control State of the
agent who has the certificate. It is the law of the community (and, of course, of the group) who enforces the existence of such a term in the Control State of the agent and allows it to save its CS. Since the law is the one that verifies the existence of this term, the secretary passively save the Control State without any verification. The operations to save and to later retrieve the Control State are \texttt{saveSCS(realname)} and \texttt{restoreSCS(realname)}. The last command replaces the Control State that has been saved by the secretary on behalf of \texttt{realname}. The acronyms in the name of the commands states for save (or restore) Stable Control State.

Since the saved Control States may last for long time in the secretary storage, there should be a mean through the obsolete Control States are removed. This command is \texttt{removeSCS}. It removes the entry that bind the \texttt{realname} with the corresponding Control State and frees up the storage space. Some variations to the basic commands described above are \texttt{rejoin} and \texttt{save-AndQuit} commands. The first one joins a group and retrieves its previously saved Control State, and it is completely equivalent to a join followed by a \texttt{restoreSCS}. The last command is equivalent to a \texttt{saveSCS} followed by a quit operation.

As we have seen in the paragraph above, the facilities for saving the Control State in a persistent manner brings up a new level of reliability and a freedom degree in the case of the mobility of the agents.

Yet, another level of reliability is added when the secretary saves the stable Control State of the agents along with the corresponding \texttt{realname} in a more stable storage - in a file on the disk. The secretary, an agent itself, is as reliable as ordinary agents. Thus if the secretary ceases to operate, it is possible to lose all the Control States already saved. It is a central point of failure also.

In order to avoid it, the secretary saves all the necessary information about the stable Control State in an initialization file every time a change in some CS is received. The file described above may be used when starting a new secretary to replace an old one, or when the secretary has to change its name (by changing the controller).

The manager of the secretary, the (possible) human that starts up and stops the secretary, is responsible for this file manipulation. This facility is not subject to the law and it is transparent to the agents of the group.
6 Conclusion

The secretary agent is an agent that runs under a special law, called the secretary law. Its main function is to form a group of member within a community. There are several features the secretary implements: a centralized mechanism by which one agent may find out about all the agents operating under the same group and, implicitly, their names. Thus the secretary may act as a name server for the group. Another issue that the secretary attempts to solve refers to the persistence of the Control State of one agent. In the condition of mobility of the agents and discontinuous operation, as long as the agent disconnects from its host controller, it may lose its name and its Control State. The secretary acts as a control state repository, providing the agents with the possibility to retrieve their saved control state at some later arbitrary moment of time. Another feature of the secretary is that of initial control state assigner: each agent that joins the group receives an initial control state, according to the control state configuration file and some particular requested role.

Though the secretary implements the above mechanism of the initial Control State assigner based on some initial configuration role file, it is still an awkward mechanism. This mechanism does not offer too many alternatives on how the control states to be assigned to the agent. In the future work a new mechanism will be proposed, such a way that the secretary may assign the initial CS based on some algorithm that can describe the functionality of the group better or which can fit and make easier the writing of the law of the group. As a future work we think of a way the agents may find what law a particular secretary serves. We have implemented already a command, getLaw that returns the symbolic name of the group law. That may not be satisfactory for many agents, and further it should be implemented a more complete mechanism.

By the mechanism of secretary, agents do not belong any more to a simple community, instead, agents form a well defined group that provides a complete mechanism of interaction. Each member has access to information regarding the membership and Control State of the group and the secretary provides persistence of the Control State and coherence of the initial Control States of agents.
7 Appendix A : Secretary law and Member law

Preamble:

InitialCS([secretary,name(secretary),obligation(timeout,0,0)])
PolicyName(secretary).

R1. sent(X,command(Y),Z) :- do(command(Y)).

R2. imported(X,L,groupCommand(Z),Y)
   :- secretary@CS, do(addPortal(L)), isCommand(Z),
     do(deliver(X,[L,groupCommand(Z)],Y)).

R3. isCommand(Z) :- (Z = join(N); Z = joinAs(N,R,P); Z =
     rejoin(N,R); Z = broadcast(M); Z = quit; Z = remove(M); Z =
     removeSCS(N); Z = saveAndQuit(M,C); Z = where(M); Z =
     who; Z = whoIs(T); Z = getLaw; Z = savePCS(C); Z =
     saveSCS(R,C); Z = restoreSCS(R); Z = open; Z = close; Z =
     setGroupSize(M,N); Z = howMany).

R4. sent(X,reply(C,R,T),Y)
    :- secretary@CS, do(export(X,reply(C,R),Y,T)).

R5. sent(X,broadcast(M,I,T),Y)
    :- secretary@CS, do(export(X,broadcast(M,I),Y,T)).

R6. obligationDue(timeout) :- do(createFingerPrint).

Figure 4: Secretary law
Preamble: Portal(secretary,URL(http://host:port/secretary.law))

InitialCS([..initial CS..])

PolicyName(member).

R1.
sent(X,command(Y),Z) :- do(command(Y)).

R2. sent(X,groupCommand(C),S) :- isCommand(C),
    do(export(X,groupCommand(C), S, secretary)).

R3. isCommand(C) :-
    (C = join(N); C = joinAs(N,R,P); C =
    rejoin(N,R); C = broadcast(M); C = quit; C = remove(M); C
    = removeSCS(N); C = where(M); C = who; C = whoIs(T); C =
    getLaw; C = restoreSCS(R); C = open; C = close; C =
    setGroupSize(M,N); C = howMany).

R4. sent(X,groupCommand(saveAndQuit(M)),S) :- do(export(X,
    groupCommand(saveAndQuit(M,NewCS)), S, secretary)).

R5. sent(X,groupCommand(savePCS),S) :- do(export(X,
    groupCommand(savePCS(NewCS)), S, secretary)).

R6. sent(X,groupCommand(saveSCS(R)),S) :- do(export(X,
    groupCommand(saveSCS(R,NewCS)), S, secretary)).

R7. imported(S,secretary,reply(restoreSCS(N),accepted(C)),X)
    :- do(deliver(S,reply(restoreSCS(N),accepted(C)),X)),
    do(replaceCS(C)).

R8. imported(S,secretary,reply(joinAs,accepted(N,C)),X)
    :- do(deliver(S,reply(joinAs,accepted(N,C)),X)),
    do(replaceCS(C)).

R9. imported(S,secretary,reply(rejoin,accepted(A,C)),X)
    :- do(deliver(S,reply(rejoin,accepted(A,C)),X)),
    do(replaceCS(C)).

R10. imported(S,secretary,Z,X) :- (Z = broadcast(C,I); Z =
    reply(C,R)), do(deliver(S,Z,X)).

Figure 5: Member law
8 Appendix B : Secretary API

Any agent that operates under the secretary group law may communicate with the secretary using the group command interface. This interface allows the agent to join a group and to request information about the group according to the policies described above. Unless otherwise specified, all the commands are issued by the agents that belong to the group. The group commands format is: "groupCommand(C)" where C is a generic command described below.

Basic commands

• **join(nickname)** - An agent sends this command when it wants to become a member of the group served by the secretary. The nickname represents the nickname the agent chooses to have. The assigned nickname may differ by the proposed nickname by a numerical suffix.

The possible answers to join are:

– reply(join, accepted(anickname)). The secretary sends back the acceptance as member of the group to the agent along with the nickname assigned to it.

– reply(join, rejected(alreadyActive)) is sent back when the agent is already active in the group. The secretary already holds a correspondence entry with the same nickname and the same name.

– reply(join, rejected(differentLaw)) is sent back when the agent operates under a different law than the law of the already functioning group.

– reply(join, rejected(groupFull)) is sent back when the group was closed by either close() group command or by setGroupSize().

This message may be issued by any agent.

• **quit** - An agent sends this command to cease its membership of the group. As a result, it is not recognized further as a member of the group.

There is no acknowledge answer. Any active member may issue this command.
• **remove**(nickname) - By issuing this command an agent removes the membership of the agent nickname to the group. It has the same result as the previous command, except that the subject of this command is not the sender.

There is no acknowledge answer. Any active member may issue this command.

Remark: the law of the group should restrict the command use.

• **who** - This command returns the list of all active members of the group. The list is composed of the names of the active agents and their nicknames, separated by comma.

The possible answers are:

- reply(who,[list of names and nicknames]) if there is at least one agent active in the group.
- reply(who,none) if the group is empty.

This message may be issued by any agent.

• **where**(nickname) - This command returns the name of the member that has the nickname nickname.

The possible answers are:

- reply(where(nickname), name) - occurs when there is one agent registered under the "nickname".
- reply(where(nickname), none) - occurs when no agent happens to be registered with the requested "nickname".

This message may be issued by any agent.

• **howMany** This command returns the number of agents that already have joined the group (the number of active members).

The possible answers are:

- reply(howMany, Number) - occurs when there is at least one active agent.
- reply(howMany, none) - occurs when there is no active agent.
This message may be issued by any agent.

• **broadcast**(Msg) This group command will ask the secretary to send the Msg message to all the active members of the group except the initiator of the broadcast.

Answers to this command:

- No reply to the sender of this message.
- broadcast(message,issuer) is the format of the message exported by secretary to all other active agents.

**Membership and Structure Control commands**

• **close** This command closes the group formed by the receiving secretary. As a result of this command all the joining attempts (join, joinAs and rejoin) will be denied by a rejected(groupFull) reply.

There is no acknowledge answer. Only the members of the group may issue this command.

Remark: The law of the group should restrict the use of this command only to privileged agents.

• **open** This command is used in conjunction with close command. As a result, the secretary will allow the any new member to join the group.

There is no acknowledge answer. Only the members of the group may issue this command.

Remark: The law of the group should restrict the use of this command only to privileged agents.

• **setGroupSize**(min,max) It instructs the secretary to close the group automatically when its size exceeds the watermark max value and to reopen the group when its size drops under watermark min value.

There is no acknowledge answer. Only the members of the group may issue this command.

Remark: The law of the group should restrict the use of this command only to privileged agents.
• **joinAs(nickname, role, password)** - An agent sends this command when it wants to become a member of the group served by the secretary and when it wants to play a particular role. The nickname represents the nickname the agent chooses to have, the role is the role it wants to play and the password is the password necessary for the particular role, according to the role configuration file of the secretary. The assigned nickname may differ by the proposed nickname by a numerical suffix. A none-password should be supplied as an implicit public password. The same none string should be present in the configuration file.

The possible answers to joinAs are:

- reply(joinAs, accepted(anickname)). The secretary sends back the acceptance as member of the group to the agent along with the nickname assigned to it.
- reply(joinAs, rejected(alreadyActive)) is sent back when the agent is already active in the group. The secretary already holds a correspondence entry with the same nickname and the same name.
- reply(joinAs, rejected(differentLaw)) is sent back when the agent operates under a different law than the law of the already functioning group.
- reply(joinAs, rejected(groupFull)) is sent back when the group was closed by either close() group command or by setGroupSize().
- reply(joinAs, rejected(roleFull)) is sent back when there are already enough active members to exceed the cardinality of the specific role.
- reply(joinAs, rejected(invalidPassword)) is sent back when the supplied password doesn’t match the one in the specific configuration role file.

This message may be issued by any agent.
Globalization of the Control State commands

- **savePCS** When an agent sends this command to the secretary, the law of the group appends the agent Control State to the message. When the message arrives, the secretary saves the Control State in a public storage on per nickname basis, making it available to the whole group for later consult.

  There is no acknowledge answer. Only the members of the group may issue this command.

- **whoIs(Term)** The secretary will reply to the agent the list of all members of the group that have the term Term in their Control State. The accurate answer of this command depends whether the Control States of the agent are saved appropriately or they are obsolete.

  The possible answers are:

  - reply(whoIs(Term),[list of names and nicknames]) when there is at least one agent who has saved its Control State in the public storage and the term Term is present in that Control State.
  - reply(whoIs(Term),none) when no agent has saved its Control State in the temporary storage or when the term Term is not present.

This message may be issued by any agent.
Persistence commands

- **saveSCS(real-name)** When the command is sent, the Control state of the agent is appended to the message. When the secretary receives the message, it saves the correspondence between the real-name and Control State of the agent in a stable storage.

  There is no acknowledge answer. Only the members of the group may issue this command.

  Remark: The group low should verify the realname passed in to this command. The verification should be done when the agent issues the saveSCS. It should be based on the existence of the term realname(realname) in the Control State of the agent.

- **restoreSCS(real-name)** This command replaces the Control State of the agent with the one that have been saved under the name real-name. It should be used after saveSCS command.

  Possible answers:

  - reply(restoreSCS(real-name), accepted(CS)) means that the restoration succeeded and the saved CS was successfully exchanged with the current CS of the agent.

  - reply(restoreSCS(real-name), rejected(notsaved)) - occurs when the secretary does not hold any Control State corresponding to the real-name presented. This situation may be possible either when no Control State has ever been stored under realname, either when the Control State held by the secretary was already removed by another agent.

  - reply(restoreSCS(real-name), rejected(rnameActive)) - occurs when another active agent has already restored the same Control State under the same realname.

  Only the members of the group may issue this command.

  Remark: The group law should verify the realname passed in to this command. The verification may be based on the existence of some term in the Control State of the agent.
• **removeSCS(realname)**  Removes the stable Control State of the realname agent from the secretary storage.

Only the members of the group may issue this command.

Remark: The law of the group should restrict the use of this command only to privileged agents.

• **rejoin(nickname,realname)**  This command is a logical combination of a join command followed by a restoreSCS command. The same rules apply as to join and restoreSCS command.

The possible answers to rejoin are:

- reply(rejoin, accepted(anickname)). The secretary sends back the acceptance as member of the group to the agent along with the nickname assigned to it. The CS is replaced during the ruling of the agent law.

- reply(rejoin, rejected(alreadyActive)) is sent back when the agent is already active in the group. The secretary already holds a correspondence entry with the same nickname and the same name.

- reply(rejoin, rejected(differentLaw)) is sent back when the agent operates under a different law than the law of the already functioning group.

- reply(rejoin, rejected(groupFull)) is sent back when the group was closed by either close() group command or by setGroupSize().

- reply(rejoin, reject(rnameActive)) is sent back when another agent has presented the same realname to the secretary, and that member is still active.

- reply(rejoin, rejected(notsaved)) is sent back to the agent when no Control State is saved currently under realname.

This message may be issued by any agent.

• **saveAndQuit(real-name)**  This command removes the agent from the group of the receiving secretary, saving the Control State of the agent under the name realname. The command is a logic sequence of saveSCS followed by a quit command.

Only the members of the group may issue this command.
Remark: The group law should verify the realname passed in to this command. The verification may be based on the existence of some term in the Control State of the agent.

- **getLaw** Returns the local name of the law the group operates under. It may be useful when one agent faces the situation to join a group and it wants to know more about it. The possible answers are:
  
  - reply(getLaw,lawname(lawname)) when at least one agent ever registered to the secretary.
  
  - reply(getLaw,lawname(none)) when no agent ever registered with the secretary.

  This message may be issued by any agent.

**References**