AN APPROACH TO ROLE BASED MANAGEMENT FOR DISTRIBUTED SYSTEMS

Emil C. Lupu, Morris Sloman

Imperial College Research Report No. DOC 95/9

Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
Email: e.c.lupu@doc.ic.ac.uk, m.sloman@doc.ic.ac.uk

Abstract

Roles have been widely used for modelling the authority, responsibility, functions and interactions associated with manager positions within organisations. In this paper we discuss the issues related to specifying roles for both human and automated managers of distributed computer systems. The starting point is that a role can be defined in terms of the authorisation and obligation policies for a particular manager position which specify what actions the manager is permitted or is obliged to do on a set of target objects. This permits inidviduals to be assigned or removed from postions without respecifying the policies for the role. However these policies are insufficient for specifying the interaction protocols between managers and the targets they manage or between different manager roles. There is a need to specify the coordination and synchronisation relating to manager interactions.

This report describes notations and techniques applicable to specifying the interactions and activities related to manager roles in a distributed system and indicates which are the most suitable for implementation within a role based management framework. Structuring a management framework in terms of roles enables a more flexible and dynamic approach to the management of a complex system.

This study assumes the existence of an underlying monitoring service and of a specification language for management policies. The role based framework is composed of a set of tools enabling the creation of roles from policies, the specification of the relationships between roles and of protocols for role interaction. In addition, the issues related to conflicts which can occur between policies within a role or between interacting roles are briefly discussed.

Keywords: distributed systems management, management roles, role interactions, management policy, obligation, authorisation, policy conflicts.

1 Roles and Policies

Role Theory, which is widely used for enterprise modelling, postulates that individuals occupy positions in an organisation [1, 2]. Associated with each position is a set of activities including required interactions that constitute the role of that position. A role thus identifies the authority, responsibility, functions and interactions, associated with a position such as vice persistent, board director, security administrator, operator responsible for reactor number three. Role theory provides us with very useful concepts for a management framework. The development of new IT techniques has introduced the ideas of assisting a human manager in his tasks [3]. The efforts have been oriented in two directions: assisting the manager at the information level and providing support for office automation. The idea of automated managers undertaking management tasks is being addressed in network management and Distributed Artificial Intelligence. However formal description techniques have not been applied to the interactions between the human and automated management. Building on concepts from role theory we intend to provide a framework where both human and automated agents coexist and which provides support for the communication and coordination between these agents.

The starting point for this work has been to define a manager role as the set of authorisation and obligation policies for which a particular manager position is the subject [4, 5]. We now discuss the concepts and notation used for specifying management policy for the actions of both human and automated agents.

Large distributed systems may contain millions of objects so it is impractical to specify policy for individual objects. Objects are grouped in domains for specifying a common management policy or to structure and partition management responsibility. A **domain** is a collection of objects (actually references to object interfaces) which have been explicitly grouped together for the purposes of management (c.f. file system directories or folders). If a domain holds a reference to an object, the object is said to be a **direct member** of that domain. A domain is an object, so may also be a member of another domain and is said to be a **subdomain**. Its members are **indirect members** of the parent domain. A **domain service** is provided for the manipulation of the membership information. Further, **domain scope expressions** can be specified determining the set of objects to which a policy applies. For example D_1-D_2-O_3 represents the objects that are members of D1 with members of D2 and O_3 excluded. Details on the domain structure and the relevant services can be found in [4-6].

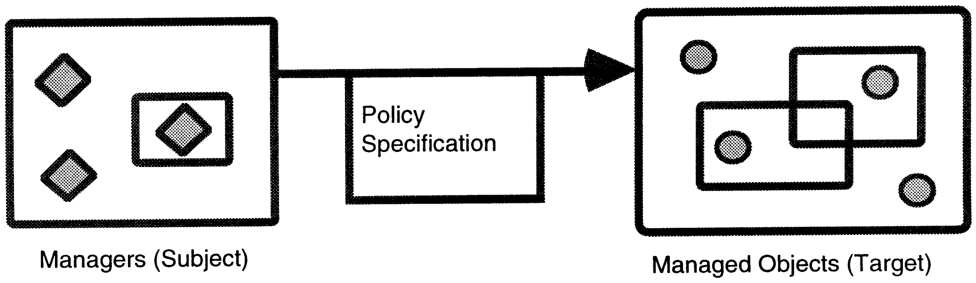


Figure 1 Policies as relationships

Policies establish a relationship (Figure 1) between manager and managed object domains. This relationship expresses either an authorisation – what activities the managers are permitted or forbidden to perform or an obligation – what activities the managers must or must not perform on the managed objects. The **mode** of the policy distinguishes between positive authorisation (permitted: A+), negative authorisation (forbidden: A-), positive obligation (must: O+) and negative obligation (must not: O-). The general format of the policies is given below with optional arguments within brackets:

policy_id mode [*trigger*] *subject* { *action* } *target* [**when** *constraint*] ;

The subject represents the set of managers assigned to carry out the actions on the set of target objects. Obligation policies can be triggered by events or time and constraints can be used to restrict the applicability of a policy. The policy format and use is described in [4, 5, 7] Example of policies are:

```
payment_1 O+ at [31/Dec/_] accountant
        { pay(percentage_of_profits, credit_transfer_pounds_sterling) }
        subcontractor ;

purchase_1 A- u:users { purchase() } services
        when u.type == anonymous ;
```

Policies can specify actions at different levels of abstraction. A refinement hierarchy can therefore be built from the more abstract policies to the enactable leaf level policies (rules). Abstract policies can only be interpreted by humans while leaf level policies are interpreted by automated managers. The enactment of the obligation policies can be triggered by composite events which are specified in a language also used by the monitoring system [8].

2 Concepts for Role Based Management

Role theory identifies concepts such as roles and positions for enterprise modelling in order to define organisational structure and activities for individuals in the organisation. We aim at using this type of expertise and modelling for network and distributed systems management. In [1] a **Role** is defined as *"a collection of rights and duties"* and a **Position** describes a status within the organisation. The role specifies management actions (i.e. policies) which represents the behaviour or dynamic aspects of the position which is essentially a static concept. We define a role as a coherent set of authorisations and obligation policies which have a particular manager position as a subject. The advantage of using a position as the subjects of the policies that individuals can be assigned to or withdrawn from their positions without having to respecify policies. In addition a position can temporarily exist without appointed managers. An organisation or distributed system may contain many roles with authority, responsibility, delegation or functional dependence relationships between them. It is necessary to specify these relationships in terms of interaction protocols which define ordering, coordination or synchronisation between activities pertaining to individual roles.

2.1 Implementation aspects of Roles

Managers can be humans or automated agents, the role structure referring in the same way to both. Humans managers will interact with a process representing them in the system (Figure 2). The underlying system deals in the same way with an automated agent and a representative of a human agent. We assume therefore the existence of a **User Representation Domain (URD)** which is a persistent representation of the human manager containing the manager adapter objects to which policies apply [5].

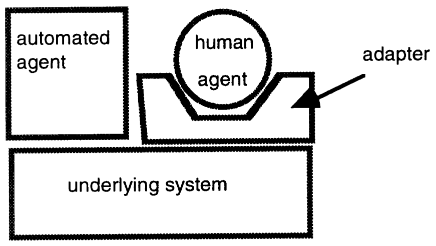


Figure 2 Human or Automated agent

A manager position is represented by **Manager Position Domain** in which are inserted the URDs of the managers sharing the position. A manager can be appointed to several positions by including his URD in the corresponding position domains.

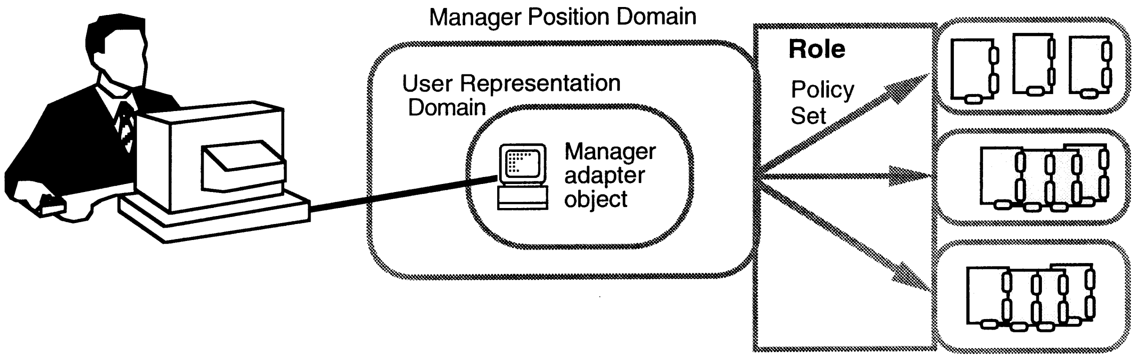


Figure 3 Position Domains and Roles

2.2 Roles and policy classes

It would be very useful to be able to parameterise a role with specific positions and target domains. It would thus be possible to define the role policy set as a class from which particular instances can be created. For example a role class could be defined for a region manager and this could be used to create North, South, East and West region manager roles. Each of the 4 roles instances relates to different manager positions each with their own specific target domains, but specifies the same policy activities and constraints for each manager position. The subject and target domains of the policies constituting a role class may not be specified. Policy classes are templates with variable subject or target domains. For example many companies have standard templates for their contractual agreements with the subcontractors. The obligations of the company can then be specified without knowing which of the company agents will implement them and to which subcontractors they apply. Contract templates are then specified by use of the policy classes.

In general polices propagate to subdomains of a parent domain which is a subject or target of a policy. Propagated policies may apply to a position domain but are not a component of the role associated with that position. For example the departmental policies for the use of computers propagate to all the Ph.D. students but are not a specific part of their obligations and authorisation of their research role.

2.3 Role Relationships

In order to achieve a management framework for distributed systems we have examined problems related to the specification of relationships between the roles which interact with each other for various purposes to request data, report availability of shared resources, agree on decisions, etc. A suitable interaction protocol has to be specified for this purpose. The communications between managerial roles and ways of specifying interactions are investigated in section 3.

Roles relate to each other as do sub-divisions of the tasks to be performed in the organisation. A particular class of these **relations** are for example the supervision of work and the responsibility towards superiors. Hierarchical decomposition of the organisation is thus a part of the relationships modelled. Among other relations are delegation and functional dependence. We distinguish between organisational relations and task related relations. The first are relations of authority, supervision, delegation, responsibility. The second are dependent on the tasks specified by the policies of the different roles. Among them are resource sharing, information access and co-ordination obligations. Identifying all the different possible classes of relations requires further study. The specification of the relationships between roles is examined in section 4.

Different managers interpret policies concurrently and policies provide only events, disseminated by the monitoring system, as a means to synchronise their activities. This is a low level and difficult means of specifying coordination and synchronisation between roles. We intend to provide a higher level notation for the specification of **sequences of activities** and conflicting activities (activities which cannot be performed at the same time). This will be further investigated in section 5.

The conflicts between roles relate to the consistency among management policies. There are three types of conflict occurring **modality conflicts, task related conflicts** and **temporal conflicts**. Modality conflicts arise from the positive/negative character of the obligation and the authorisation policies. These conflicts can be easily detected in terms of overlapping domains. The task related conflicts are specified by meta-policies and the temporal conflicts are detected from the sequences of activities. Policy conflicts are also considered in section 6.

Conclusions

This section has introduced the different concepts regarding the definition of the roles in terms of policies. Domains are used for structuring the different objects of the system and to implement the positions. We have further introduced the concepts of relationships between roles and the interactions occurring between the managers sharing a position or appointed to different positions. Both interaction protocols and the inter-role relationships will be investigated in more detail in the following sections. Finally the problems of conflict between the management policies and of temporal ordering of activities have been introduced.

3 Role Interaction

Interaction between the different managers occurs at several levels in a role based framework. The managers sharing a position have to interact between them in order to co-ordinate the interpretation of the different policies specified by the role. Managers performing different roles have to interact in order to achieve their tasks, co-ordinate their actions, share information and agree on their commitments. This section attempts to elaborate on the issues of interaction protocols between roles and proposes a framework suitable for both human and automated managers.

3.1 Related work

Communication protocols in computer networks and telecommunication disciplines have often been specified using finite state machine or Petri-Nets. This approach is largely unsatisfactory when dealing with complex interactions among human agents, the main limitations of these notations being the following:

- All the possible 'states' of the agents in the interaction have to be foreseen.
- Two party interactions can be specified but the notation introduces undesirable complexity for multi-party interactions.
- The messages exchanged have no semantic value – they are transitions from one state to another.

A different approach, by computer scientists from an organisational and artificial intelligence background, attributes a meaning i.e. a semantic value to the type of exchanged messages and derives a behaviour from the message. This meaning is termed an “illocutionnary” value. For example statements such as “The URL of our document archive is `ftp://dse.doc.ic.ac.uk/dse-papers`” have an assertive value on the state of the system while statements such as “I will perform the backup of the system by 9.00 pm” have a commisive value on the activities of the manager. The first system introducing such ideas was **The Coordinator** [9, 10]. Winograd identifies the concept of Conversation for Action. (CfA) based on the Speech Act theory [11]. Higher level protocol specifications follow along this line, among them de Greef's notation for conversation, logics like Modal Action Logic [12] , Conversational Clichés [13] and Information System design [14]. This work is largely inspired by the desire to support co-operation between agents representing human managers.

3.1.1 De Greef's notation: a high level specification of multi-agent interaction

We introduce de Greef's notation with the specification of a haggling interaction between a buyer and a seller. The notation and the algorithm are presented in [15]. Exchanged messages can be graphically represented in a network style specification as given in the figure below. This type of graphic representations are also used in Winograd's work.

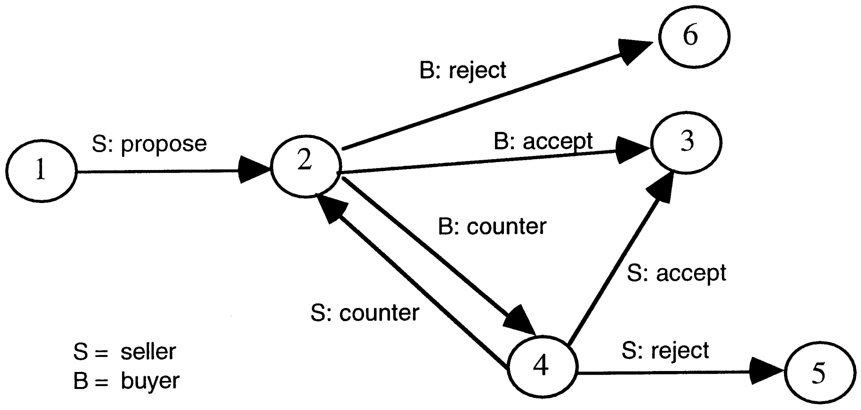


Figure 4 The haggling protocol

De Greef's notation inherits features of a logic programming language, provides sets and introduces the notion of agents. The notation is powerful enough to express multi-agent algorithms such as the Contract Net [16]. Support for multi-agent interaction is provided by the use of sets of receiver agents and by the ability to collect messages from multiple senders.

The haggling interaction is represented in de Greef's specification language as:

```
seller, buyer : haggle (price) =
   seller: propose(p) -> buyer,
   seller, buyer : continue_haggle(p,0,price).

seller, buyer: continue_haggle(ask,bid,price) =
   ( buyer : reject -> seller
   | buyer : counter(newbid) -> seller, bid < newbid < ask,
        ( seller : reject ->buyer
        | seller : counter(newask) -> buyer,
                                        newbid < newask < ask,
        seller, buyer : continue_haggle(newask,newbid,price)
        | seller : accept -> buyer, price = newbid
        )
   | buyer : accept -> seller, price = ask
   ).
```

The interaction is composed of an initial proposal sent by the seller and a recurrent call `continue_haggle`. The buyer can reject an offer, accept it or make a counter proposal. In this later case the seller can reply by accepting or rejecting the counter proposal or issuing a new counter-proposal provided the price asked is higher than the buyers' bid and smaller than the last price asked. After two counter-proposals the haggling continues with the new parameters. This notation has been implemented for inter-agent communication in April [17, 18].

3.1.2 A temporal specification of the sequences of messages

There are several ways of describing the sequence of messages between the different interacting parties. An interesting visual approach based on the use of Petri Nets and "*Interactions*" for specifying the task coordination and interdependencies can be found in Singh's Role Interaction Nets [19]. This use both the notion of time and multi agent system in the specification. An *interaction* is a bi-directional synchronous mechanism that encapsulates communication and computation among the involved parties in a joint action. It is used for the coordination of agents and the specifications have a direct translation into Petri-Nets. A specification is based on a two dimensional graph. The horizontal dimension represents the concurrency of the different interacting agents and the vertical dimension represents the time as shown in the Figure 5. This allows the visualisation of the different sequences of activities for one role and the concurrency between different roles.

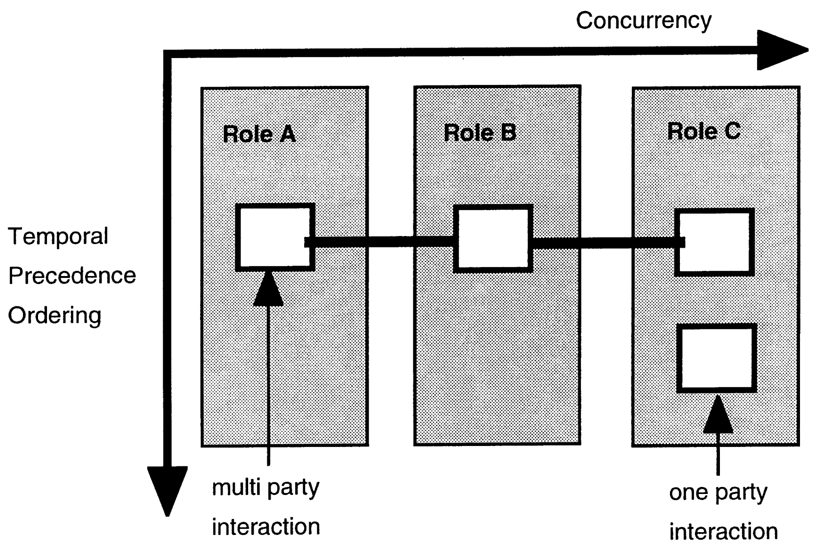


Figure 5 Graphical & Temporal specification of interactions

3.2 Issues for Role Interaction

The issues of a relating to a role interaction protocol are:

- the type of managers (human or automated) involved in the interaction
- two party or multi party interaction
- the semantic value of the messages exchanged (illocutionary value of the messages)
- the content of the exchanged message

Human vs. Automated interactive entities

A communication protocol between automated agents has to be deterministic. Although heuristic approaches can be developed include choices to represent non-determinism in the protocol, the behaviour of the automated agent then becomes unpredictable. Interactions occurring between human managers are always partly non-deterministic due to human choice – the several possible answers a human manager can make to a message. Most of the work in the area constrains the human managers to the pre-defined protocol and concentrates on capturing the illocutionary value of the messages exchanged. The protocol we are aiming at should provide a flexible framework for communication between the human agents as well as a deterministic approach for communication between automated managers. For example, a human manager might respond to a counter proposal by a request for further details even if this was not specified in the initial protocol. Determinism for automated managers can be ensured by restricting the use of choice operators.

Two party vs. multi party interaction

The problem of multi party interaction is an underlying concern when modelling interactions between agents. Although techniques such as the Contract Net have been available for a long time most of the recent work does not deal with multi-party interactions. De Greef's notation [15], and the April language [17, 18] identify the need for sets in the interaction specification language to express the multiple senders or receivers in an interaction and for collecting the multiple messages received by one manager. We believe that multi-party interactions are essential e.g. for specifying joint decisions or task allocation for roles.

The illocutionary value of the messages exchanged

In order to automatically derive a meaning from the messages exchanged they must be typed (request, reply, proposal, statement, etc.) and a finite Universe of Discourse (UoD) [14] must be defined for each interaction. A UoD is the set of all the types of messages used within the interaction. An illocutionnary value can than be given to each type of message and rules can be build for deriving knowledge of the state of the system and the activities of the managers. We do not discuss illocutionary value in this paper but make provision for it and recognise its benefits for assisting the managers in their tasks.

The content of the exchanged messages

In the early studies of the interaction protocols the value of the message is the value of the illocutionnary act it represents. These are entirely predefined in the protocol. A message has however the same value regardless of the stage the interaction protocol has reached. Schmidt and Simone describe in [20] the case of a "bug form" which is filled in by the various managers in the process of the development of a software product. The success of this protocol relies partially on the fact that the constraints placed upon the different managers and the past interactions between them are explicitly represented within the exchanged message – the bug form. In particular when the designer receives the bug form he has knowledge of the different other stages – tests, specification scheme, etc. This highlights one of the main limitations of the actual protocol specification. In both The Coordinator and the de Greef specification language a received message triggers the same action regardless of the previous "history" of the exchanged messages[1]. Specifications like "reject the negotiation only after the second counter offer received" are desirable and should be accommodated.

3.3 The Proposed Interaction Specification Notation

The specification notation is separated into a global specification of the sequence of messages and a local production rule based specification for each manager, in order to achieve the flexibility for an interaction system that includes human managers. The global specification includes the description of the interacting parts and a pre-defined sequence of messages. It will be graphical, but can be automatically transformed into a set of production rules which can then be changed, if necessary, within a manager.

[1] In de Greef's specification language variables can locally be maintained in order to remember the previously exchanged messages. This approach is not satisfactory since it implies keeping logs of the messages exchanged with each manager.

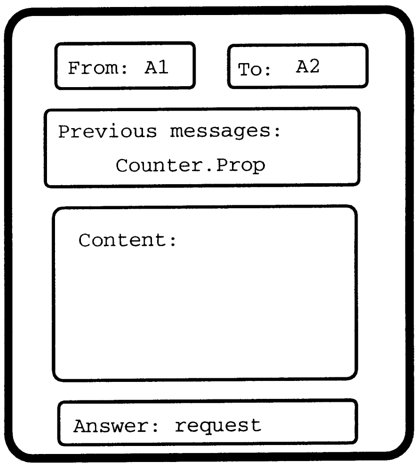
3.3.1 Configuring a sequence of messages

Configuring a sequence of messages can be achieved with the network style specification used by both de Greef and Winograd. This also has the advantage of representing the interaction as a finite state machine with each state representing a manager at some point in the interaction (see figure 4). It is therefore possible to translate this global specification into sets of production rules by representing the various regular expressions recognised at each state of the finite state machine. The regular expressions represent then the history of the exchanged messages. They are distributed to each manager and constitute the local specification of the protocol. This representation by regular expressions is possible because the messages are typed and the Universe of Discourse is finite.

3.3.2 A local specification for the messages exchanged and their semantics.

The content of a message

The exchanged message should contain the sequence of the types of the previous exchanges in the interaction (we assume here trusted interactions). This is like the bug form which carries the signature of the different agents finding, analysing or correcting the bug. The message should also contain the source manager and the destination manager. A special feature that will enable changes in the protocol is to allow the sender of the message to specify the types of desired answers. The "message form" could have the following representation:



In textual form a message is represented by:

```
        message: 'TO:' <destination>, 'FROM:' <sender>,
            'LOG:' /* sequence of the types of messages exchanged during
               the interaction */
            'CONTENT:' <content>, <possible answer> .
```

As an example the message `[TO: accountant, FROM: Mr. Beecham, LOG: request.payment, CONTENT: card_no: xxx exp_date: xxx, request_id]` means that the accountant has received a payment from Mr. Beecham subsequent to a request for payment. The payment was made by a credit card and Mr. Beecham offers the accountant the possibility to reply by requesting additional identification elements. ‘Request_id’ and ‘payment’ must belong to the Universe of Discourse and the sequence `request_id.payment` can match a regular expression and trigger one of the rules defined in the protocol of the accountant.

Formal specification

A BNF style description of the notation of the production rules we are aiming at for locally specifying the interaction protocol is:

```
rule ::=
    regular exp :: guard => reply_possibility
reply_possibility ::= basic_reply
              | reply_possibility or reply_possibility
              | reply_possibility , reply_possibility
basic_reply ::= send_message
              | activity_seq
              | { reply_possibility }
activity_seq ::= activity
              | activity , activity_seq
send_message ::= [type, content , answer] -> manager
```

The regular expressions match the list of types of the messages exchanged (present in the incoming message) and trigger the rules. Guards are predicates expressed on the contents of the message. The guards provide the means to distinguish between messages of the same type and with the same history in order to trigger different rules. The body of the production rules is composed of activities to be performed and various permitted replies.

Production rules for the protocol specification

The possible replies a human manager can send to a message are given by a set of production rules triggered by the regular expressions on the sequence of messages. As an example the haggling over a price between a buyer and a seller could be represented by:

Seller

```
(1) [propose, price, ] -> buyer

(3) propose.{counter}.counter => [reject, reasons, ] -> buyer
      or [accept, delivery, ] - > buyer
      or [counter, asked_price, ] -> buyer,
      received bid < ask < lastprice
```

Rule (1) specifies that the seller should initiate the interaction by proposing a price. Rule (3) specifies that the after receiving the initial proposal or one or more counter-offers, the seller can reject the last counter-offer by giving reasons, can accept it and deliver the product or can make a new counter-offer.

Buyer

```
int last price

(2) propose.{counter} => [reject, reasons, ] -> seller
      or [accept, delivery, ] -> seller
      or [counter, new_offer, ] -> seller
```

The advantage of such a notation is that new rules can be very easily added to change the alternatives defined by the protocol. For example, the following rule could be added for the seller to permit rejection of an offer only after at least two counter offers (assuming reject is not present in rule 3) :

(4) propose.counter^2.{counter} => [reject, reasons,] -> buyer

Dynamic aspects

In order to outline the dynamic features that can be used in this framework we have to consider the possibility that the seller wants to permit the customer to enquire about the capabilities of the sold item. In order to do this the rule (3) should be written as:

```
(5)  propose.{counter}.counter =>
    [counter, newask, require] -> buyer, newbid < newask < ask
    or [reject, reasons, ] -> buyer
    or [accept, delivery, ] -> buyer
```

Receiving this message will permit the buyer to make enquiries if the `require` triggers one of the rules of the buyer for requiring details.

3.4 From a de Greef specification to production rules

We aim in this section to give a short overview of the differences between the De Greef notation and the proposed notation. There are many similarities between the two, mainly on the specification of a set of activities to be undertaken upon receipt of a message.

De Greef's notation and it's main implementation in April [17, 18] specify a network style graph and is aimed at interactions between automated managers. It's advantages rely in a recursive notation where the history of the interaction is described by nested structures (the behaviour of the seller in the specification of section 3.1.1 is nested within the behaviour of the buyer). The implementation in April reflects this nesting and offers good support for the implementation of sets.

The proposed notation is based on a global network style specification and local sets of production rules and offers a more flexible framework for the specification of an interaction protocol. The global network style specification can be translated into a set of rules which can be edited and changed at a local level. The history of exchanged messages during the interaction is made explicit through the use of regular expressions. The protocol can be changed without having the need to recompile the specification but by addition of rules. Finally the use of production rules offers the possibility to derive plans or commitments from the interaction. When dealing with human managers the different rules triggered by the receipt of a message represent different potential answer to model human choice. In order to ensure determinism for the interactions between automated agents or between a human and an automated agent, each rule should be limited to only one reply (restriction on the use of 'or') and each incoming message should trigger only one rule. Finally we believe that it is possible to derive a set of production rules from a de Greef specification.

3.5 Further Work

Further work is needed for the translation of the graphical specification of the sequence of messages into a set of rules. An interpreter for the rules has to be built (eg using Prolog). A more flexible extension of the graphical specification must be provided to enable the specification of occurrences of messages. The user interface has to be implemented.

Further research is needed for the case of automated managers and for ways of hierarchically structuring an interaction protocol. In particular the provision for structured specification has to be made (see definition of "macros" in [19]).

By using production rules the possibility of deriving commitments or abstracting plans should be easy to add [21]. Interactions based on abstraction of plans are investigated in feature interaction within intelligent telecommunication systems [22, 23].

Conclusions

The specification of an interaction protocol involving human agents must make provision for non-determinism and allow high level specifications. We distinguish between the global specification of a sequence of messages (regardless of the behaviour of the agent concerned) and the specification of a protocol implementing the communication framework within the agent. In particular the protocol can be changed within a manager by changing the set of production rules. Restrictions are made on the specification language of the protocol in order to ensure determinism for automated managers.

The specification of a sequence of messages can be achieved in the network specification style used by both de Greef and Winograd. This can be translated in a minimal number of rules that define the local behaviour of the agent with respect to the interaction.

The message exchanged should contain the sequence of messages previously exchanged within the same interaction. This sequence matches a regular expression and triggers one or several rules of the local behaviour, giving in the case of human managers the different possible replies. Alternative answer can also be expressed within the message by specifying the type of the answer accepted in the 'answer' field of the message. Guards on the content of the message are incorporated offering the possibility of triggering different rules for the same message type.

The next section investigates the specification of relationships between roles and the importance of the interaction protocols within the relationships.

4 Relationships between Roles

This section discusses characteristics of the relationships between managerial roles for distributed systems and introduce some ideas about their specification and implementation.

Roles and their relationships were first introduced by sociologists and psychologists for analysing organisations. However most of the work deals with "power" based sociological relationships [2]. We are more concerned with what Hirscheim calls the "analytical taxonomy" [24]: the perspective of an environment in which managers perform a variety of formal and structured actions. In terms of policies a relation can be defined as a *link* between two or more roles. A relationship is composed of interactions occurring between the roles and plus the policies applying to the interaction. The idea of associating relationships with interactions and obligations is also formulated in [25] where the commitments arising from interactions form the relationship. An example of a relationship is a contractual agreement between two or more parties which defines the obligations and authorisations of the contractual parties as well as a minimal set of interactions (periodic meetings, delivery of products, etc.). Relationships based on the interdependencies of tasks are also studied in [26].

4.1 Characteristics of a relationship

The characteristics of a relationship between management roles can be described in terms of the topology, the type of relationship and the functional interdependence. Relationships are often represented in a graphical way [27, 28] where the topology may be one-to-one, one-to-n and n-to-n as outlined in the Figure 6.

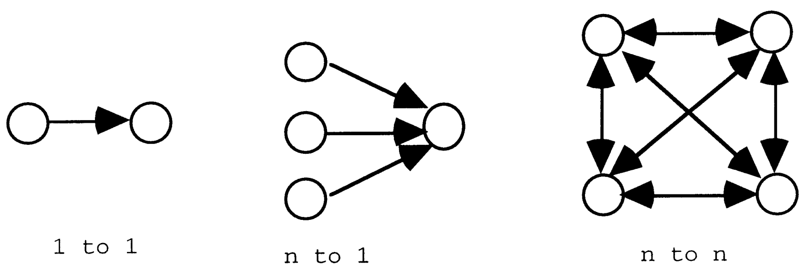


Figure 6 Arity of a relationship

Further the topology represents the connectivity of the graph i.e. fully connected, centrally connected or independent elements.

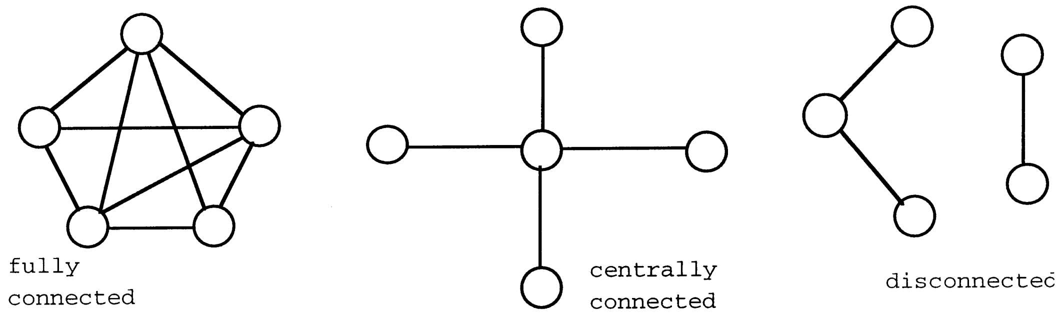


Figure 7 Connectivity of a relationship

A client/server relationship is always a centrally connected graph while disconnected graphs represent independent groups in the organisation. The topology of the relation can also be used to distinguish between a centralised and a decentralised market in the customer/supplier relationship as outlined in [29]. Further we can distinguish between vertical (hierarchic) relationships and horizontal (peer to peer) relationships. However there are many problems with a classification of the relationships based on their topology as the organisational structure influences the topology of the relations (e.g. centralisation of the decision making process [29]) so the same relationship can have different topologies according to the organisational structure and different relationships can have the same topology. For instance supervising the activities of the sub-ordinates and granting mandatory access rights both indicate a hierarchic topology.

4.3 Activity based relationships

We now consider the issues of an environment in which the managers perform a variety of actions specified by policies. The different relationships between the management roles are thus related to the interdependencies between the activities managers perform. We are investigating the different types of relationships and their characteristics as summarised in the table below.

Relationship arising from tasks performed by a role	task decomposition	• Delegation • Task Assignment • Remote Execution • market like bidding (contract net)
	responsibility	• Supervision • Responsibility to superiors • Report
	task ordering	• Functional dependency • Pre-emption
Service or Information provider relationships		• Client / Server • Contractual Relation • Provide advice • Reporting
Collaborative relationships		• Collaborative peer-to-peer • Conversational • Group agreement • Reporting

As in any type based classification the same relationship can belong to several types. For example the reporting obligation belongs to both a sub-class of the task related relationships and to the service class. The report can be in the first case the result of an activity or the statistics over a set of queries in a client/server relationship in the second case.

We have identified three main classes of relationships referring to the assignment of tasks, the information transfer and the collaborative design. The task related relationships further divide in 3 sub-classes distinguishing between the task decomposition, the responsibilities associated with the tasks and the task ordering.

The delegation and the task assignment relationships are very close to each other. However we distinguish between the assignment of a task obtained by the refinement of a higher level policy and the delegation of one of the activities of the role. These are generally 1 to 1 or 1 to n relationships and are associated often with the supervision of activities and the report to supervisor relations. The delegation and the task assignment relationships are specified by authorisation policies granting these rights.

Remote execution differs from the task assignment and delegation as it refers to an occasional activity to be performed with the outcome being returned to the manager originating the request. The information on the activity to be performed is conveyed by the interaction protocol.

The market like bidding relation corresponds to a particular case of task assignment in which the task is assigned to the manager placing the highest bid. The associated interaction protocol is presented in [16].

The responsibility sub-class contains the supervision, the responsibility to the supervisors and the reporting obligations. These are 1 to 1 relationships grouping obligations such as checking the result of each activity, reporting the results over a period of time or asking confirmation from the supervisor before performing a task.

The task ordering relationships establish the preferences and the sequences in which tasks should be performed. Policies assigned to different roles can be executed concurrently unless an interdependence relationship is specified between them.

The service relationships establish the obligations of the different related parts like in the contractual relationships and the interactions such as the haggling protocol. Locally these relationships have a centrally connected graph.

Collaborative relations often occur in committees and team work. Collaborative relations between separated teams can be structured in terms of the obligations and the interactions between the different parts. The issues to address refer to the case of a small team where the relations are informal and where the obligations of the members are not easy to separate. The main burden is then placed on the interaction protocols which have to cover a variety of interactions ranging from informal discussions to constructed voting schemes. Here deriving commitments [13] from the interaction protocols can prove very useful. These relationships are generally represented by fully connected graphs.

Specifying a relationship

A relationship is defined by the responsibilities of the related parts and the interaction protocols ruling the conversations between them. The responsibilities correspond to policies which are the obligations and authorisations. The idea of representing the relations in terms of the obligations of the related parts is also expressed in [25, 30]. In particular in [30] relationships are defined through contracts. We believe however that the interactions are tightly associated with the relations and that the specification of a relationship should include the specification of the associated interaction protocols.

The interaction protocol leads to decisions or commitments on the enforcement of the obligations of the related parts. The receipt of a message within the interaction protocol is for instance an event that can trigger a policy. Vice versa a policy can lead to the initiation of an interaction protocol (e.g. reporting the success or failure of an undertaken task).

Example

We present the relationships between the customers (users), a software retailer (Presentation_AG) and its subcontractors which allows customers to purchase products from subcontractors. The retailer presents the products in a multimedia session. The users can purchase the products and pay the retailer for the products purchased. The retailer has a sales manager responsible for the sale of the subcontractor products to the users and a security manager responsible for the authentication of users. We describe the interaction protocols for the two relationships linking the users to the sales manager and to the security manager, using policies and production rules.

<u>Relationship between the users and the security manager</u>

The different policies defining the obligations and authorisations of the related parts are:

```
p_purchase_1 A+ u:users { browse() } Presentation_AG
        when u.type == anonymous ;

p_purchase_2 A- u:users { purchase() } Presentation_AG
        when u.age < 16 ;

p_purchase_3 A+ u:users { purchase() } Presentation_AG
        when u.type != anonymous ;

p_purchase_4 O+ sales_manager { set() } prices
        when /* within limits */ ;

p_purchase_5 O+ on payment(p:product) sales_manager
        { deliver(product) } users;

p_purchase_6 O+ on purchase users {
        pay(purchase_price, pounds_sterling)
    } sales_agent ;

p_purchase_7 O+ on purchase sales_manager { send(invoice) } user;
```

The policies authorise non-anonymous users aged more than 16 to purchase goods and obligate them to pay once the product is purchased. Anonymous users are authorised to browse but cannot purchase. The sales manager has the obligation to set the prices and to deliver the product once the payment is received.

Because of the simplicity of the example we present here directly the production rules of the interaction protocol for each agent.

Sales Manager

```
[pay,(amount, list_of_items, payment_modes), ] -> user

pay.payment =>  send_event(payment(p),...) ,
            [receipt, receipt_form, ] -> user
```

User

```
pay :: member(Visa_card,payment_modes) =>
      [payment, (Visa_card,card_no,exp_date), ] -> SalesAgent

pay => [payment, (cheque), ] -> Sales Agent
```

These rules specify an interaction between the user and the sales manager for the payment. The sales manager initiates the interaction by sending an invoice for payment to the user. This invoice contains the permitted payment alternatives. The invoice of type "pay" triggers the two rules defining the interaction protocol for the user. The human user then has the choice between paying by cheque or VISA card. In the case of an automated agent only one rule may be triggered. The use of guards is particularly useful in this case and the first matching rule will be triggered if the payment can be made by VISA card.

<u>Relationship between the users and the security manager</u>

The policies are:

```
p_access_1 A+ users/NAP[2]/England + users/PA[3]/England
      { connect_to() } Presentation_AG ;

p_access_2 A- u:BlackList_users { connect_to() } Presentation_AG ;

p_access_3 O+ security_manager {
        /* permanent address should not be used to deny access */
        } u:Users
        when u.type == corporate ;

p_access_4 A+ security_manager { suspend() } user_connection;

p_access_5 O+ on connection_request security_manager
      { authenticate } u:users ;

p_access_6 O+ on fraudulent_connection_attempt security_manager {
      enter(action) } security_log ;
```

These policies specify that users accessing the service from England and users with a permanent address in England are authorised to connect. Users on the Black List are not authorised to connect. These two policies can create a conflict if a user belongs to both the domains `users/NAP/England` and `BlackList_users`. These conflicts are briefly examined in section 6. The policies also specify that the security manager is authorised to suspend a

[2]Network Access Point

[3]Permanent Address

connection and any fraudulent connection attempt will be logged. The security manager has to authenticate the user and corporate users should not be denied access when located at their permanent address. The interaction protocol is given by the following production rules:

User

```
[connect, ,]  -> security manager

connect.request :: request = authentication =>
       [send, (client_No, PIN), ] -> security manager
 /* if the request regards the authentication the user sends its
client number and its PIN */
```

Security Manager

```
connect  => [request, (client_no, PIN) , ] -> user

connect.request.send :: send = client no, PIN   && (member(client
no,corporate users) || member(From, authorised NAP) || client_no =
anonymous)
       =>    [authorisation, authorisation_Id , ] -> user
/* The restrictions on the access are implemented within the
interaction protocol. The guard implements the criteria on which the
authorisation to connect is issued.*/
```

This example also outlines the connection between the policies and the interaction protocols. Policy p_access_6 is implemented by the interaction protocol since it specifies the sending of a payment invoice to the user. The receipt of a message constitutes an event which can trigger a policy – the security manager has to authenticate the user on receipt of a connection request.

4.4 Organisational structure

Often organisational structure is described in terms of hierarchies relying on "power" relationships which are not defined consistently. The analysis of power relationships is complicated as the its characteristics vary between managers and between different organisation. A power relationship should be specified in terms of obligations authorisations and interactions. Obligations such as supervising the activities or reporting to the supervisor what tasks have been achieved can then be easily defined. The interactions needed between the managers can be specified in the protocol suggested in section 3.

Conclusions

A relationship is composed of responsibilities and interaction protocols. The specification of a relation is therefore composed of the specification of the policies and of the interaction protocols. Policies and interaction protocols interrelate in the following way: (1) the receipt of a message can form an event which triggers a policy (2) a policy can be implemented or refined into an activity which is the sending of a message initiating an interaction protocol.

The role framework must also cover issues such as conflicts between policies and the specification of sequences of activities.

5 Concurrency of Activities within Policies

As mentioned there is a need to specify the concurrency between the activities specified within the policies. There are many well known notations for the specification of the concurrency among processes – among them CCS, CSP and Petri-Nets are widely used. Our aim is not to provide a new calculus system for the specification of concurrency but rather to provide an easy way of specifying sequences of activities within the role framework. Policies are grouped into roles and are interpreted concurrently by the various managers. It is necessary to specify whether activities must be performed sequentially or concurrently and when they must synchronise. In this section we propose a graphical notation based on Petri-Nets which can then be translated into a textual notation.

5.1 Graphical Concurrency Notation

An activity expression is represented by square box with a transition (circle) preceding and following it triggered by an event (token). The required concurrency operators are represented as follows:

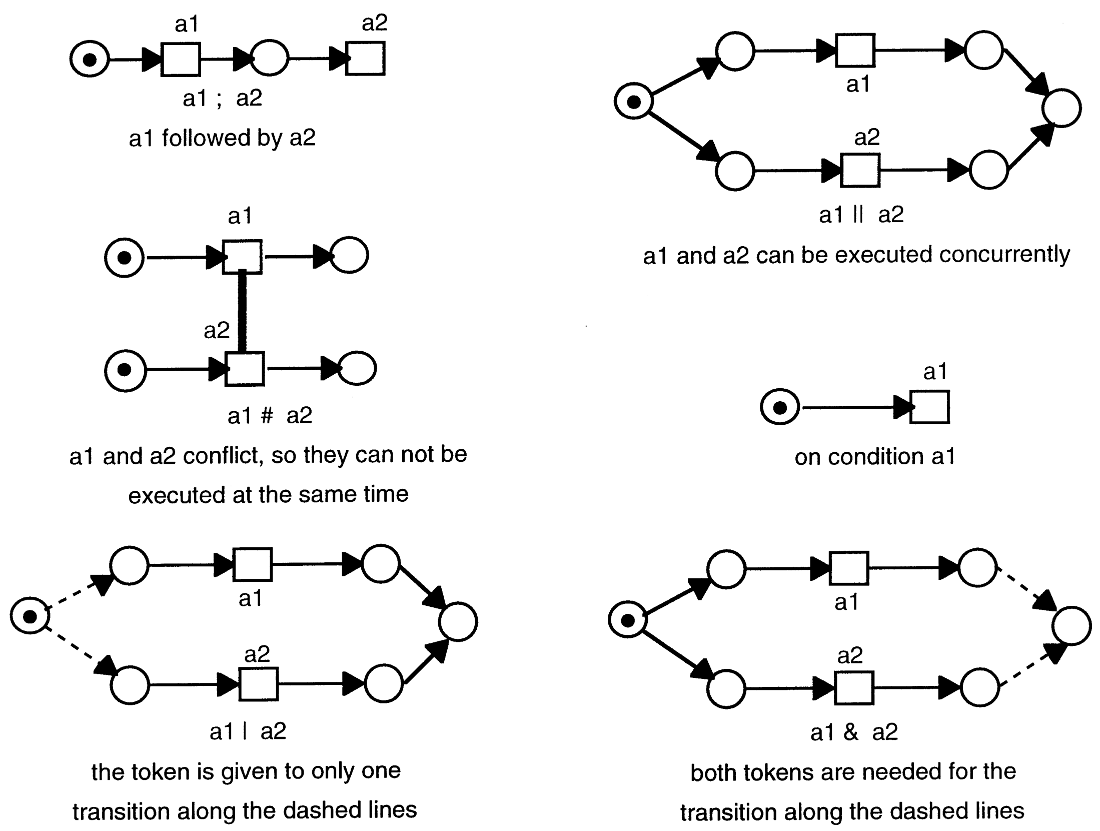

a1
a2
a1 ; a2
a1 followed by a2
a1
a2
a1 || a2
a1 and a2 can be executed concurrently
a1
a2
a1 # a2
a1 and a2 conflict, so they can not be executed at the same time
a1
on condition a1
a1
a2
a1 | a2
the token is given to only one transition along the dashed lines
a1
a2
a1 & a2
both tokens are needed for the transition along the dashed lines

Figure 8 Petri Net style concurrency representation

This Petri Net style representation offers the advantage of representing the events as tokens firing the different transitions, but this does increase the complexity of the representation.

5.2 Textual Concurrency Notation

A role consists of policies which specify activities. We refer to the activity a2 within the policy P3 of the role "Accountant" by `Accountant:P3:a2`. A sequence of activities separated by semicolons forms a block enclosed in brackets:

```
{accountant:p2:register_payment ; accountant:p3:issue_check}.
```

An activity can be followed by a sequence:

```
R1:p2:act2 ; {R2:p3:act4 ; R1:p3:act5}.
```

However the specification of a sequence of activities must also provide ways of expressing concurrency. We introduce the "||" operator for this purpose. `A1 || A2` allows the two activities to be executed concurrently and the execution of the sequence continues when any of these activities are completed. Alternatives between activities can be expressed with the "|" operator:

```
{{accountant:p2:register_payment;accountant:p3:issue_check}|
accountant:p4: delay_payment }; accountant:p5:produce_report.
```

The "&" operator expresses the need for synchronised activities, e.g. `{A1 & A2}` states that both A1 and A2 can be executed concurrently but they both have to be completed before the sequence proceeds. We also permit conditional expressions of the form:

```
{on (condition) sequence}
```

Finally an operator "#" indicating conflicting activities, belonging to different sequences, which cannot be executed concurrently is also useful.

The syntax of proposed notation is specified in the following BNF:

```
spec ::= complex_activity | conflict_exp;

conflict_exp ::= sequence # sequence
complex_activity ::=
         activity_exp
   | conditional_activity
   | complex_activity ; complex_activity
   | { complex_activity & complex_activity }
   | { complex_activity | complex_activity }
   | { complex_activity || complex_activity}

conditional_activity = on (condition) activity_exp;

activity_exp ::= opt_event activity opt_event;

opt_event ::= /* empty */
   | event ;

activity ::=  role : policy_id : policy_activity;
```

Policies are triggered by events monitored within the distributed system and we use the GEM notation [8] to combine simple events for form composite, more abstract events. The semantics of the above notation can be defined in terms of events for the beginning and end of an activity execution. Events can also be used to express an order between two sequences of activities. Further, explicit events are declared within a policy specification in order to trigger the activities. The notation of an activity in a sequence is therefore extended to permit optional events e.g. [event] activity [event] specified by the user.

5.3 Event based semantics

The implementation of the concurrency operators can be based on compound events. All events, other than those explicitly declared by the user, will be automatically generated by a translator of the concurrency specification. We show the translation of the operators into event composition operators. An event generated by the compiler is denoted by "i" and an event declared by the user is denoted by "e".

```
a1i1 ; i2a2 => i2 = i1
      /* The event i1 marking the end of a1 triggers i2 for
      the start of a2. If a1 is not triggered then i2 >= i1
      (i2 must occur after i1)*/

a1i1 ; e2a2 => i2 = e2 & i1
      /* the event i2 triggering a2 is triggered, is the
      combination of the event i1 marking the end of a1 and
```

```
          the event e2 specified by the user.  e2 and i1 can occur
          in any order*/

    a1 e1 ; * => i1 = e1
          /* If an event e1 is declared by the user as marking the
          end of a1 then the next activity is triggered by an
          event i1 = e1. If the next activity is not triggered
          than it must occur later (i1 > e1) */

    i1{i2 a2 i2f | i3 a3 i3f} i4 =>
           i1 = i2 or i3, i4 = i2f or i3f

    i1{i2 a2 i2f || i3 a3 i3f} i4 =>
           i1 = i2 & i3, i4 = i2f or i3f

    i1{i2 a2 i2f & i3 a3 i3f} i4 =>
           i1 = i2 & i3, i4 = i2f & i3f

    i1 on (condition) i2 a2 i2f i3
          => i1 = i2, i3 = i2f

    i1 a1 i1f  # i2 a2 i2f  => i1f < i2 or i2f < i1
```

Conclusion

We have introduced a graphical and textual notation for specifying concurrency and synchronisation of activities. This specification can be translated in compound events which are implemented by the monitoring service.

6 Conflicts

We briefly discuss in this section the conflicts occurring between the management policies and the principles for a conflict detection mechanism. In particular the different types of conflict have to be investigated and the scope of the conflicts detection checking has to be examined.

Policies represent managers and managed objects as sets. Preventing these sets from overlapping is not practical nor desirable. A classification of the conflicts based on the overlapping sets of managers, actions and managed objects is presented in [31] . Here we will distinguish between three types of conflicts according to the specifications from which they arise. The main classes of conflicts are modality conflicts, temporal conflicts, and application specific conflicts.

6.1 Types of Conflicts

Modality conflicts occur if there is both a positive and negative authorisation or obligation policy with the same subjects, targets and activities. Work in this area, restricted to the authorisation policies, can be found in [32]. In the conflict detection mechanism we distinguish between **potential conflicts** which represent the possibility of a conflict and are detected by analysing the overlaps between the domain scope expressions and **real conflicts** need precedence relationships based on domain nesting or priorities in order to resolve the conflict.

Temporal conflicts occur in the specification of sequences and in the specification of temporal constraints of the policies. In sequences of activities the conflicts expressed can be directly specified and therefore inconsistencies can be directly translated in temporal causality over the events structure. However the problem of conflict detection regarding the temporal constraints of the policies needs further work.

Application specific conflicts are characterised by inconsistent activities present in two different policies. The inconsistency, being application specific does not appear in the format of the policies. Additional specifications are thus needed for the detection of these conflicts. Although overlaps occur in these conflicts detecting them will only lead to the detection of potential conflicts. In order to specify the inconsistencies between policies we introduce the concept of **meta policies.** A meta policy is a logical predicate over the managers, activities and managed objects sets expressed in the policies. The predicate is a normal form containing membership operations on these sets. The fact that a manager is not authorised to both enter the payment and sign the payment cheque can be represented by:

any(P_1,P_2) belonging(policy_domain_expression)
false <- intersect(P_1.managers, P_2.managers) && belongs('enter', P_1.activities) && belongs('sign' , P_2.activities) && belongs('cheque', intersect(P_1.targets, P_2.targets)

This type of expression can be written or translated in a logic based programming language like Prolog.

6.2 Conflict Detection

The role structure is decentralised and must be scalable. Therefore the cost of large conflict checks in terms of computation would be too high. Although this work is at its early stages several remarks can be made. A role must be checked for conflicts. This will ensure that all the activities assigned to the managers sharing the position can be performed. Relationships specify the obligations and authorisations of the related parts. Conflict checking must ensure that the related parts do not have conflicting obligations. The conflict check will therefore ensure consistency between the obligations and authorisations each relationship assigns to the roles.

Conclusions

Several aspects of a role based management framework have been examined in this work. The framework is based on a set of tools enabling the specification of policies and means of structuring them into roles. Interaction between the managerial roles has also been examined and the concept of relationship has been formulated based on the respective obligations and authorisations of the related parts and on their interactions.

Finally a way of specifying sequences of activities and ensuring temporal coordination between tasks has been outlined as well as a graphical representation. The conflicts occurring between management policies have been pointed out and the concepts of potential conflicts, real conflicts and meta policies have been introduced. Further work is however needed along this path in particular on the operational semantics of a role.

Acknowledgements

We acknowledge financial support from the EPSRC within the RoleMan project for the development of this work. We are grateful to BT for permission to use the Travel Co scenario as example of management policies. Many thanks for comments which have improved both the content and the form of this report to Masoud Mansouri-Samani, Damian Marriott and Nikolaos Skarmeas.

References

[1] B. J. Biddle and E. J. Thomas, "Role Theory: Concepts and Research," . New York: Robert E. Krieger Publishig Company, 1979, pp. 450.

[2] B. J. Biddle, *Role Theory, Expectations Identities and Behaviour*: Academic Press Inc., 1979.

[3] C. A. Ellis and G. J. Nutt, "Office Information Systems and Computer Science," in *Computer Supported Cooperative Work: A book of readings*, I. Greif, Ed.: Morgan Kaufmann Publishers Inc., 1988, pp. 199-247.

[4] K. Becker, U. Raabe, K. Twidle, and M. Sloman, "Domain and Policy Service Specification," Imperial College, Technical Report IDSM deliverable D6 and SysMan deliverable MA2V2, 1993.

[5] M. S. Sloman, "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, vol. 2, no. 4, pp. 333-360, 1994.

[6] K. P. Twidle, "Domain Servicies for Distributed Systems Management," Imperial College - Department of Computing, Ph.D. Thesis 1993.

[7] D. A. Marriott, M. Mansouri-Samani, and M. S. Sloman, "Specification of Management Policies," presented at 5th IFIP International Workshop on Distributed Systems: Operations and Management (DSOM), Toulouse (Fr), 1994.

[8] M. Mansouri-Samani and M. S. Sloman, "GEM-A Generalised Event Monitoring Language for Distributed Systems," Imperial College - Department of Computing, Research Report 95/8, 1995.

[9] F. Flores, M. Graves, B. Hartfield, and T. Winograd, "Computer systems and the Design of Organisational Interaction," *ACM Transactions on Office Information Systems*, vol. 6, no. 2, pp. 153-172, 1988.

[10] T. Winograd, "A Language/Action Perspective on the Design of Cooperative Work," in *Computer Supported Cooperative Work: A Book of Readings*, I. Greif, Ed.: Morgan Kaufmann Publishers Inc., 1988.

[11] J. R. Searle, *Speech Acts: An Essay in the philosophy of Language*: Cambridge University Press, 1969.

[12] J. Pitt, M. Anderton, and J. Cunningham, "Normalized Interaction Between Autonomous agents: A Case Study in Interorganizational Project Management," presented at International Workshop on the Design of Cooperative Systems (COOP'95), Antibes-France, 1995.

[13] C. Laufer and H. Fuks, "ACCORD Conversation Cliches for Cooperation," presented at International Workshop on the Design of Cooperative Systems (COOP'95), Antibes-France, 1995.

[14] P. Johannesson, "Representation and Communication - A Speech Act based approach to Information Systems Design," *Information Systems*, vol. 20, no. 4, pp. 291-303, 1995.

[15] P. de Greef, K. Clark, and F. McCabe, "Towards a Specification Language for Cooperation Methods," presented at 16th German AI-Conference, GWAI'92, Berlin, 1992.

[16] R. G. Smith, "The Contract Net Protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. 29, no. 12, pp. 1104-1113, 1980.

[17] F. G. McCabe and K. L. Clark, "Programming in April - An Agent PRocess Interaction Language," Imperial College-Department of Computing, Technical report 1994.

[18] F. G. McCabe and D. A. Chu, "April Reference Manual," Imperial College - Department of Computing, Reference Manual 1994.

[19] B. Singh and G. L. Rein, "Role Interaction nets (RINs): A Process Description Formalism," MCC, Technical Report CT-083-92, July 1992.

[20] K. Schmidt and C. Simone, "Mechanisms of Interaction: An approach to CSCW Systems Design," presented at International Workshop on the Design of Cooperative Systems (COOP'95), Antibes-France, 1995.

[21] A. Finkelstein and H. Fuks, "Multi-party Specification," presented at Fifth International Workshop on Software Specification and Design, Pittsburg,Pennsylvania, USA, 1989.

[22] N. D. Griffeth and H. Velthuijsen, "Win/Win Negociation among Autonomous Agents," presented at DAI Worshop, 1993.

[23] H. Velthuijsen, "Distributed Artificial Intelligence for Runtime Feature-Interaction Resolution," *IEEE Computer*, August, pp. 48-55, 1993.

[24] R. A. Hirscheim, "Understanding the Office: A Social-Analytic Perspective," *ACM Transactions on Office Information Systems*, vol. 4, no. 4, pp. 331-344, 1986.

[25] A. H. Bond, "A Computational Model for Organizations of Cooperating Intelligent Agents," presented at ACM Conference on Office Information Systems, Cambridge (MA), 1990.

[26] T. W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys*, vol. 26, no. 1, 1994.

[27] O. A. Oeser and F. Harary, "A Mathematical Model for Structural Role Theory, II," *Human Relations*, no. 17, pp. 3-17, 1964.

[28] O. A. Oeser and F. Harary, "Role Structures: A description in Terms of Graph Theory," in *Role Teory: Concepts and Research*, B. J. Biddle, Ed. New York: Robert E. Krieger Publishing Company, 1979, pp. 92-102.

[29] T. W. Malone, "Computer Support for Organisations: Towards an Organisational Science," MIT Sloan School of Management, Working Paper 85-012, Management in the 1990s, Sept. 1985.

[30] N. Skarmeas, "Organisations through Roles and Agents," presented at International Workshop on the Design of Cooperative Systems (COOP'95), Antibes-France, 1995.

[31] J. D. Moffet and M. S. Sloman, "Policy Conflict Analysis in Distributed System Management," *Ablex Publishing Journal of Organisational Computing*, vol. 4, no. 1, pp. 1-22, 1994.

[32] A. Heydon, M. Maimone, J. Tygar, J. Wing, and A. Zaremski, "Miro: Visual Specification of Security," *IEEE Transactions on Software Engineering*, vol. 16, no. 10, pp. 1185-1197, 1990.