

MIgories: an abstract model for interaction *

Adrian Silvescu & Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science, Iowa State University
Ames, IA 50010, U.S.A.

April 3, 2005

Abstract

Recent advances in high throughput data acquisition and data storage technologies call for designing distributed agents that are able to learn. This paper proposes a new abstract framework for modelling interactions among agents in multi-agent organizations. The proposed model – the *model of interaction categories*, or MIgories exhibits *compositionality* of interactions as well as *emergence* of behavior that is not explicitly designed at the organizational level. The proposed framework is expressive enough to model some of the commonly observed interactions in both natural as well as artificial organizations. More importantly, it allows us to specify and analyze interactions in multi-agent organizations at a fairly high level of abstraction, independent of specific implementations.

1 Introduction

Recent advances in computers and communications, have made it possible, at least in principle, to design and implement large communicating applications consisting of large numbers of relatively autonomous agents. Multi-agent organizations consisting of interacting agents offer an attractive approach for specification, design, and implementation of such systems. Examples of such systems include distributed intelligent information networks [HMW98], electronic marketplaces [AJ99], virtual enterprises, distributed electric power systems [BC98], etc. Coordination of dynamic interactions among autonomous, distributed entities is one of the key problems in the design of such systems.

Researchers have investigated coordination mechanisms inspired by natural systems (e.g., brains, immune systems, social organizations, economies) as well as artificial systems computers, multi-computers, operating systems computer

*This research was supported in part from grants from the National Science Foundation (NSF 9982341), Department of Defense, the John Deere Foundation, the Carver Foundation, and Pioneer Hi-Bred Inc.)

networks, programs, factories). For some examples see [Uh84, HU90]. In multi-agent systems, the need for specification of inter-agent interaction and coordination mechanisms has begun to receive considerable attention [Fe99, W99]. Examples of coordination mechanisms that have been investigated include inter-agent negotiation (e.g., using the contract network protocol) [Sm80, Sa98].

Most of the current programming languages employed in the development of distributed communicating applications use object-oriented, or more recently, agent-oriented programming paradigms. In the discussion that follows, the primary distinction that we make between objects and agents hinges on the observation that objects are relatively static entities whereas agents are relatively dynamic and active entities that are capable of autonomous behavior. In their pure form these self-centered paradigms (agent, object oriented) implement coordination patterns as an integral part of the objects or agents. This approach limits the ability to abstract the coordination process from the coordinated entities, which leads to a loss in flexibility and transparency [FA93, CD99].

The fact that coordination structures and processes can be specified relatively independently of the internal structure (and implementation) of entities to be coordinated was noted by several researchers e.g., [GC92, Ki97]. Partly in response to these findings, several languages and models for coordination have emerged over the recent years. Some examples include Linda [GC92], Synchronizers [FA93], CoLaS [CD99], Manifold [Ar96], Gamma [B96], ActorSpace [Ag93], AspectJ [LK98], Interaction-Oriented Programming [S96]. Although some of these languages make serious advances in improving the methods for specification of coordination among participating entities or processes in a system, they do not provide a natural means of composing complex interactions from simpler interactions. In other words, the coordination languages lack *compositionality* [CD99].

In this paper, we develop a formal framework for specification of organizational structures in terms of *roles* and *interactions* among agents. The proposed framework – *model of interaction categories* (MIGories) – enables high-level specification of organizational structures to support coordination among agents in large multi-agent systems. MIGories display properties of compositionality (i.e., ability to combine simpler interactions into more complex interactions) as well as emergence of behaviors that are not explicitly designed at the organizational level. This approach to specification of multi-agent interactions at a fairly high level of abstraction makes the essential characteristics of the interactions transparent in a manner that is independent of the details concerning specific architectures and implementations of agents. The generality of the proposed model facilitates specification and analysis of interactions in a wide variety of natural and artificial organizations including multi-agent systems.

The rest of the paper is organized as follows: Section 2 describes the proposed model of interaction categories (MIGories). Section 3 provides an illustrative example. Section 4 compares the proposed framework with some existing approaches to specification of interactions in multi-agent systems. Section 5 concludes with a brief summary, and a discussion of promising directions for further research.

2 Model of Interaction Categories

Experience gained from the design of coordination languages has led to an elucidation of some of the desirable properties coordination models [CD99, O99]. In our opinion, an abstract model of coordination has to have the following properties:

- *Orthogonality* – It should be possible to specify interactions among entities independently of the specification and design of the coordinated entities.
- *Encapsulation* – The coordination structures should have access only to the public interface of the coordinated entities and not their (private) internal structure or implementation details.
- *Organization and Abstraction* – The model should be able to support coordination of multiple entities at a high level of abstraction.
- *Compositionality* – It should be possible to compose coordination patterns can be composed in order to obtain new patterns.
- *Dynamicity* – It should be possible for new entities and new coordination structures or interactions to be added to or removed from the system.

With these requirements in mind, we proceed to define our model of interaction categories (MIGories). We start with the notion of *rôles*, which provide an an abstraction of the more familiar concept of interfaces used in software engineering. This is followed by definitions for abstract and concrete MIGories.

An early development of these ideas was done in [Sil97].

2.1 Rôles

The intuitive meaning that we associate to a rôle is basically the same as in natural language. The Webster dictionary defines role as *a function or part performed especially in a particular operation or process*. Usually a rôle will be associated with a set of requirements that a certain entity is supposed to fulfill in order to be able to play that corresponding rôle. In an event based model for example, a rôle can consist in a set of events that the player of that particular rôle has to be able to respond to or generate.

Definition 1 (rôle-set).

We define a rôle-set as a 2-uple $\mathcal{R} = (R, \oplus)$ where:

- R is a set of rôles
- $\oplus : R \times R \rightarrow R$ is a binary operation on R

A rôle-set is essentially a set of roles along with a binary operation (\oplus) that enables us to obtain a rôle that is the *sum* of two rôles.

Example: A person that is a *Father* and a *Banker* will plays the rôle $Father \oplus Banker$.

Most of the time we do not want the result of a *sum* of rôles to depend on the order in which the sum is done, and we would also like to identify a neutral element for the operation \oplus . We therefore define a *standard rôle-set* as follows:

Definition 2 (standard rôle-set).

We define a *standard rôle-set* as a 3-uple $\mathcal{R} = (R, \oplus, \epsilon)$ where \mathcal{R} is a commutative monoid, i.e. the following properties hold:

- (R, \oplus) is a rôle-set
- \oplus is an associative and commutative operation
- $\forall r \in R \quad r \oplus \epsilon = \epsilon \oplus r = r$

A natural issue to consider the existence of an order relation among rôles. The meaning of this order relation $r_1 \leq r_2$ is that if an entity can play the rôle r_2 then it can also play the rôle r_1 . This leads to the following definition of a *well-behaved rôle set*:

Definition 3 (well-behaved rôle set)

We say that $\mathcal{R}_{\leq} = (\mathcal{R} = (R, \oplus), \leq)$ is a *well-behaved rôle-set* where \mathcal{R} is a rôle-set and \leq is a partial order on R , if the following compatibility condition holds:

$$\forall r_1, r_2 \in R \quad r_1 \leq r_1 \oplus r_2 \quad \text{and} \quad r_2 \leq r_1 \oplus r_2$$

Example of well-behaved standard rôle-set: Let Req be a set of requirements. Then $\mathcal{R}_{\leq} = ((2^{Req}, \cup, \emptyset), \subseteq)$ is a *well-behaved standard rôle-set*. Since $(2^{Req}, \cup, \emptyset)$ is evidently a monoid, we simply need to check the compatibility condition i.e.:

$$\forall L_1, L_2 \in 2^{Req} \quad L_1 \subseteq L_1 \cup L_2$$

It is easy to see that the compatibility condition holds in this case.

Remark: In fact the symbol \oplus can be viewed not just as a single operator, but as a family of operators. For instance, in the example above, the person playing the *Father* \oplus *Banker* rôle may be an excellent *Banker* and an excellent *Father*, or an excellent *Banker* but a horrible *Father*. This case can be represented by two operators: $\oplus_{good,good}$ and $\oplus_{good,bad}$. For the sake of notational simplicity however, we will ignore this issue in the rest of the paper. However, it is worth keeping in mind that \oplus can denote a family of operators.

2.2 MIgories

Now we are in a position to define abstract and concrete MIgories. The basic difference between the two is similar in spirit to the difference between a *class* and an *object* that is a member of the class in object-oriented programming, or the intuitive difference between a concept and an instance of the concept. The *abstract MIgories* will offer us the general description of an organizational model while *concrete-MIgories* will provide instantiations of such organizations.

However, the manner in which abstract MIGores are instantiated is radically different from the class-object notion of instantiation.

In this section we will use the term *agents* to refer to the coordinated entities and the term *interactions* to refer to coordination structures.

Notation: We will denote by $[r_1, \dots, r_n]$ the multiset that contains the (not necessary distinct) elements: r_1, \dots, r_n .

Definition 4 (abstract MIGories)

We say that $(\mathcal{R}_{\leq}, \mathcal{I}, \mathcal{O})$ is an **abstract Model of Interaction Category (abstract MIGory)** if:

- $\mathcal{R}_{\leq} = (\mathcal{R}, \leq)$ is a well-behaved standard rôle-set.
- \mathcal{I} is a set of sets $I([r_1, \dots, r_n])$ and the elements $i \in I([r_1, \dots, r_n])$ are called models of interactions among rôles r_1, \dots, r_n .
- \mathcal{O} is a set of composition laws $\circ_f : I([r_1^1, \dots, r_{k_1}^1]) \times \dots \times I([r_1^m, \dots, r_{k_m}^m]) \rightarrow I([r_1, \dots, r_n])$ where f is a function $f : [r_1^1, \dots, r_{k_1}^1, \dots, r_1^m, \dots, r_{k_m}^m] \rightarrow \{1, \dots, n\}$ and:

$$r_i = \bigoplus_{r \in f^{-1}(i)} r$$

- the operations defined above are commutative, associative with each other and have a neutral element $i_{\square} \in I[\square]$.

An abstract-MIGory is defined by a set of rôles along with possible interactions among these rôles, and operators for composing these interactions. Intuitively, the composition of interactions is performed by making an agent play a sum of the respective rôles. In this way the interactions containing these rôles will become connected through these respective agents, thus producing a new interaction. However some somewhat strange compositions can also occur, such as those given in Examples 2 and 3, namely, *independent* and *singular* compositions, respectively.

Example 1: Consider the interaction $Family \in I([Father, Mother, Son])$ and $Bank \in I([President, Vicepresident, Employee])$ then if the *Father* and the *Son* are *President* and *Vicepresident* i (respectively) of the *Bank* then we can specify a composition operator:

$$\begin{aligned} \circ_f : I([Father, Mother, Son]) \times I([President, Vicepresident, Employee]) \\ \rightarrow I([Father \oplus President, Son \oplus Vicepresident, Mother, Employee]) \end{aligned}$$

that maps the two interactions *Family* and *Bank* into a new interaction $BankFamily \in I([Father \oplus President, Son \oplus Vicepresident, Mother, Employee])$.

Example 2 (independent composition): Consider the same two interactions *Family* and *Bank* as in the preceding example, with one difference: There is no relation between any of the *Family* members and *Bank* members. This leads to a composition operator:

$$\circ_f : I([Father, Mother, Son]) \times I([President, Vicepresident, Employee])$$

$$\rightarrow I([Father, Son, Mother, President, Vicepresident, Employee])$$

that will give us the interaction *FamilyAndBank*. This operator will perform an *independent* composition (i.e. although the two interactions are combined into a single one, they actually do not yield any new interactions).

Example 3 (singular composition): Consider the following interaction: $Trial \in I[Judge, Prosecutor, Jury, Accused, Advocate]$. Then if the *Accused* is a lawyer, he might try to defend himself and play the rôle $Accused \oplus Advocate$. Thus we have a composition operator

$$\begin{aligned} \circ_f : I([Judge, Prosecutor, Jury, Accused, Advocate]) \\ \rightarrow I([Judge, Prosecutor, Jury, Accused \oplus Advocate]) \end{aligned}$$

And this operator will map the interaction *Trial* into *Trial'*, where in *Trial'* the *Accused* defends himself (i.e. plays also the rôle *Advocate*). We will call such a composition *singular* because it is applied to only one interaction.

The interface between agents and the abstract-MIgories are rôles. In order to capture the fact that only some agents are able to play certain rôles we give the following definition:

Definition 5 (agent-set) *We say that $(\mathcal{A}, \mathcal{R}_{\leq}, roles)$ is an agent-set if:*

- \mathcal{A} is a set of agents.
- \mathcal{R}_{\leq} is a well-behaved standard set of rôles
- $roles : \mathcal{A} \rightarrow 2^{\mathcal{R}}$ is a function that identifies the rôles that a specific agent can play.
- $roles$ satisfies the following compatibility relation with respect to the partial order on the rôles

$$\forall a \in \mathcal{A} \forall r, r' \in \mathcal{R}, r' \leq r \text{ and } r \in roles(a) \Rightarrow r' \in roles(a)$$

The compatibility condition for *roles* basically says that if an agent can play a rôle r then it can also play any rôle that has fewer requirements (i.e. is smaller with respect to the partial order relation) than this rôle r .

The set \mathcal{I} specifies types of interactions, therefore in order to be able to talk about actual organization we will have to talk about instantiated interactions, therefore the following definition:

Definition 6 (instantiated interactions-set) *We say that $(\mathcal{X}, type)$ is an instantiated interactions-set if:*

- \mathcal{X} is a set
- $type : \mathcal{X} \rightarrow \bigcup_{I \in \mathcal{I}} I$ is a function that gives the type of a certain instantiated interaction from the set \mathcal{X}

Notation: Given a set A we will denote by $[A]$ the set of all multisets with elements from A .

Definition 7 (concrete-MIgories) We say that $(\mathcal{R}_{\leq}, \mathcal{I}, \mathcal{O}, \mathcal{A}, \text{roles}, \mathcal{X}, \text{type}, \text{cast})$ is a concrete Model of Interaction Category (concrete-MIgority) if :

- $(\mathcal{R}_{\leq}, \mathcal{I}, \mathcal{O})$ is an abstract-MIgority
- $(\mathcal{A}, \mathcal{R}_{\leq}, \text{roles})$ is an agent-set
- $(\mathcal{X}, \text{type})$ is an instantiated interactions-set
- $\text{cast} : \mathcal{X} \rightarrow [A]$ is a function that for every instantiated interaction identifies the agents that will play the roles in that interaction.
- and the agents assigned by cast for an instantiated interaction have to be compatible with the roles that they have to play in the interaction. i.e. $\forall x \in \mathcal{X}$ if $\text{type}(x) \in I([r_1, \dots, r_n])$ and $\text{cast}(x) = [A_1, \dots, A_m]$ then

$$n = m \text{ and } \forall i \in \{1, \dots, n\} \ r_i \in \text{roles}(A_i)$$

Example of concrete-MIgority: We will illustrate the concrete-MIgorities again with the *BankFamily* example:

- Let the Agents be Bill, Tom, Kate and Peter.
- Let the types of interactions, roles and composition operators be as in **Example 1**.
- roles will make the following assignments:
 - Bill will be the *Father* \oplus *President*, *Father*, *President*
 - Tom will be the *Son* \oplus *Vicepresident*, *Son*, *Vicepresident*
 - Kate will be the *Mother*
 - Peter will be the *Employee*
- Let the instantiated interactions set be $\{\text{CITIBANK}, \text{OneFamily}\}$ where $\text{type}(\text{CITIBANK}) = \text{Bank}$ and $\text{type}(\text{OneFamily}) = \text{Family}$,

Then cast will do the following assignments:

- $\text{cast}(\text{CITYBANK}) = [\text{Bill}, \text{Tom}, \text{Peter}]$.
- $\text{cast}(\text{TheNextDoorFamily}) = [\text{Bill}, \text{Tom}, \text{Kate}]$.

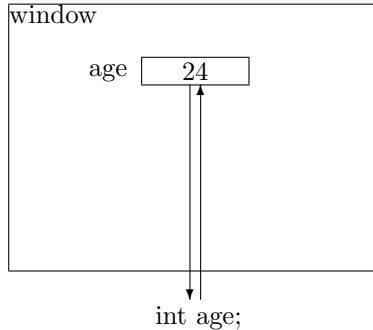


Figure 1: **The variable-editbox model of interaction**

3 Examples

The first example is presented in figure 1.

This interaction model basically means that whenever the value of the editbox labeled *age* changes then the value of the integer variable *age* should change accordingly and whenever the value of the integer variable *age* changes this should be reflected in a change in the editbox labeled *age*. Therefore we will have an interaction $i_{EV} \in I([Editbox, Variable])$.

Another very familiar model of interaction is that presented in Figure 2.

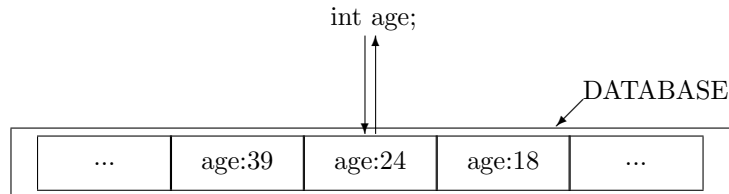


Figure 2: **The variable-database field model of interaction**

This second model of interaction has the meaning that whenever the integer variable *age* changes then the corresponding database field *age* associated currently with the integer variable changes and of course the converse: whenever the database field changes this change should be reflected in a corresponding change in the integer variable *age*. Therefore we will have an interaction $i_{VF} \in I([Variable, Field])$.

Now that we have two interaction models at hand, we are in a position where we are able to compose them. One way is to make the integer variable *age* play the role of *Variable* in both of the interactions defined above i_{EV} and i_{VF} respectively. (i.e. say that the variable *age* from Figure 1 is the same as the one in Figure 2). The resulting model of interaction is depicted in Figure 3. This is an interaction $i_{EVF} \in I([Editbox, Variable \oplus Variable, Field])$

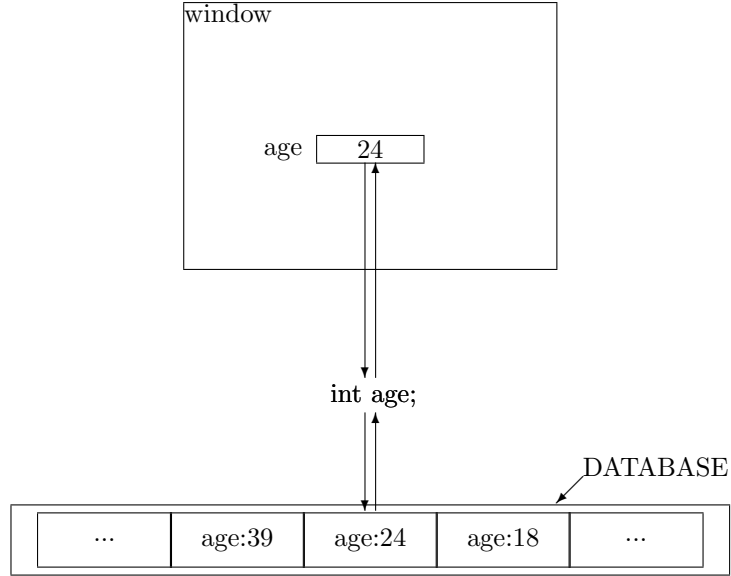


Figure 3: **The variable-database field model of interaction**

Even in this very simple case we already have an example of an *emergent* (i.e. not explicitly designed) interaction that appears between the editbox and the database field, with the following semantics: whenever the editbox changes so does the database field and whenever the database field changes so does the editbox.

4 Models and related work

In this section we explore some possible models for MIgories. These include Petri nets of different flavors [Mu89, Pe81, Je92], coordination languages and specifications such as Synchronizers[FA93] CoLaS[CD99] and Interaction-Oriented Programming [S96], and NK-model [Ka93], which is a model of interaction among genes in biological cells.

Petri nets: A Petri net [Mu89, Pe81, Je92], is a model for describing communicating sequential processes in a quite general class of systems. It consists of *places*, *transitions* and *directed arcs* that connect the places to transitions and transitions to places. Places can contain a multiset of *tokens* that might be of different types. The current state of the system (marking) is represented by a function that gives for every place the multiset of tokens that are there. Transitions are active components. Activities are modeled by flow of tokens through the net. Flow of tokens occurs as a result of firing of transitions, which changes the state of the system (marking). In order for a transition to be able to fire a certain condition with respect to the tokens contained in the set of input places has to be satisfied. When the transition fires, it removes tokens from its input places and adds some at all of its output places.

Petri Nets can be used to provide a formal model of specific instantiations of MIGories. One possible way in this sense, although not necessarily the only one, is to view Petri Nets as a kind of degenerate model for MIGories. In this case the agents will correspond to places (that are inert repositories, hence the degeneracy) and the transitions to interactions. Petri Nets however have no high level concept that would be able to deal with composition.

Synchronizers + CoLaS: Synchronizers[FA93] and it's more sophisticated descendents CoLaS[CD99] are coordination languages designed in the same spirit as MIGories, with the similar sets of requirements in mind). They focus on describing coordination mechanisms and have message-based interfaces that connect the coordinated entities (called groups). While exhibiting *encapsulation* and providing good support for the description of coordination structures, they do not exhibit *compositionality* [CD99], insufficient abstraction of the notion of interaction.

Interaction Oriented Programming: Interaction-Oriented Programming [S96] is a declarative specification based on event algebra for modeling coordination of interactions among agents. The declarative specification is compiled into executable temporal logic constraints that are processed at runtime in order to produce the desired behaviour. The method of specification is nonintrusive (i.e. respects the encapsulation of agents). The interfaces are event-based. The approach as described in [S96] appears to be somewhat domain oriented. The level of abstraction is also not very high and the specifications lack the compositionality property.

NK-models: NK-models due to Kaufmann [Ka93] are a model for studying interactions among genes in biological cells. The model assumes that N genes in the cell are either turned *on* or *off*. Furthermore the state of each gene at time t is dependent upon the state of K ($K \leq N$) other genes at time $t - 1$. These NK-models provide an excellent example of MIGories. The agents in this case are genes and the interactions are dependencies that tie one gene to the K others. There are two possible *rôles*: *dependent* and *dependee*.

Composition of all these dependency relations yields a a model of the cellular life cycle. This model exhibits complex and intricate emergent behaviors. Experiments reported by [Ka93] show that the most interesting phenomena occur when K is not too small or too large. These observation may serve as a possible

guideline for the future designers of models of interactions.

5 Discussion

Effective coordination structures are necessary components of multi-agent systems, distributed communicating applications, and distributed enterprises. Such systems call for coordination structures that allow for flexible and dynamic interactions among a large number of entities. The various models discussed above as well as specific interagent coordination protocols (e.g., the contract network protocol [Sm80, Sa98]) and communication protocols (e.g., KQML [FYM97]) have been proposed as solutions to the coordination problem. The model of interaction categories introduced in this paper can potentially be mapped to any of these models and implemented using a variety of coordination protocols and communication languages. Yet, it is an abstract model, that makes transparent the essential aspects of coordination, largely independent of the design of of the coordinated entities, the choice of implementation languages, etc. While MIGories has been motivated by the needs of coordination structures for multi-agent systems, it is general enough and flexible enough to specify interactions in a variety of natural and artificial systems.

MIGories offer a new paradigm of programming that is illustrated in figure 4.

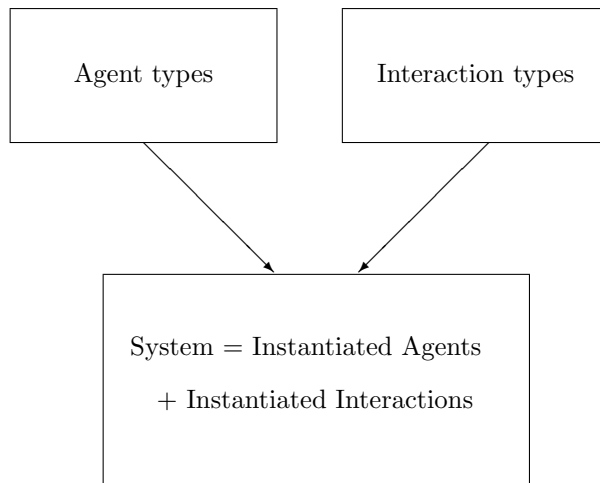


Figure 4: **Coordination-Oriented Programming**

We envision such a coordination-oriented approach to programming multi-agent systems to be comprised of the following:

- specification of agents

- specifications of the desired model(s) of interaction
- instantiation of the specified agents
- instantiation of the specified model(s) of interaction

The proposed model of interaction categories (MIGories) provide a high level abstract model of coordination among multiple (possibly autonomous) entities that separates the specification of coordination structures from the specification and implementation of entities to be coordinated. It supports composition of interactions to obtain new interactions. It is flexible enough to allow changes to multi-agent organizations through addition and deletion of agents or coordination structures, or roles. Thus, the proposed model some of the key desiderata of coordination structures elucidated in [CD99, O99].

The proposed model is expressive enough to support specification of interactions in both natural and artificial multi-agent systems. The proposed model allows design and analysis of coordination structures independent of low level implementation details.

Long-term objectives of this work include development of high level languages for specification of organizational structures which can be instantiated to realize specific concrete organizations; theoretical analysis of characteristics and global behaviors of different organizational structures; and specification and design of multi-agent organizations in terms of the necessary interactions and rôles for applications such as distributed intelligent information networks [HMW98], electronic marketplaces [AJ99], distributed electric power networks [BC98], virtual enterprises, etc. It is our hope that further analysis of formal properties of MIGories would lead to the development of improved coordination languages as well as a new coordination-oriented programming paradigm for design and implementation of large, dynamic, complex systems consisting of heterogeneous, autonomous agents.

References

- [Ag93] Agha G., Callsen C.J., ActorSpace: An Open Distributed Programming Paradigm, *4th ACM Conference on Principles and Practice of Parallel Programming*, ACM Sigplan Notices, vol. 28, no. 7, 1993.
- [AJ99] Albers, M., Jonker, C.M., Karami, M., and Treur, J. An Electronic Marketplace: Generic Agent Models, Ontologies, and Knowledge. In: *Proceedings of the Fourth International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology*. 1999.
- [Ar96] Arbab F., The IWIM Model for Coordination of Concurrent Activities, *COORDINATION'96*, LNCS 1061, Springer-Verlag, 1996, pp. 34-55.
- [B96] Banâtre J.P., Parallel Multiset Processing: From Explicit Coordination to Chemical Reaction. *COORDINATION'96*, LNCS 1061, Springer-Verlag, 1996, pp. 1-11.

- [BC98] Brazier, F.M.T., Cornelissen, F., Gustavsson, B., Jonker, C.M., Lindeberg, O., Polak, B., and Treur, J. Negotiating for Load Balancing of Electricity Use. In: *Proceedings of the Eighteenth International Conference on Distributed Computing Systems*, IEEE Computer Society Press. 1998.
- [CD99] Cruz, J.C. and Ducasse S., A Group Based Approach for Coordinating Active Objects. *COORDINATION'99*, LNCS, 1594, Springer-Verlag, April 1999, pp. 355-370.
- [FYM97] Finin, T., Labrou, Y., and Mayfield, J. KQML as an agent communication language. In Jeff Bradshaw (Ed.), *Software Agents*, MIT Press, Cambridge, MA. 1997
- [FA93] Frolund, S. and Agha G., A Language Framework for Multi-Object Coordination. *ECOOOP'93*, LNCS, 707, Springer-Verlag, July 1993, pp. 346-360.
- [Fe99] Ferber J., Multi-Agent Systems - An introduction to distributed artificial intelligence. New York: Addison-Wesley. 1999.
- [Ge85] Gelernter D., Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), pp. 80-112, 1985.
- [GC92] Gelernter D., Carriero N., Coordination Languages and their significance. *CACM*, vol. 35, no. 2, February 1992. 7(1), pp. 80-112, 1985.
- [HU90] Honavar, V. & Uhr, L. (1990). Coordination and Control Structures and Processes : Possibilities for Connectionist Networks. *Journal of Experimental and Theoretical Artificial Intelligence* 2 277-302.
- [HMW98] V. Honavar, L. Miller, and J. Wong. Distributed knowledge networks. In *Proceedings of IEEE Information Technology Conference*, pp. 87-90, IEEE Press, 1998.
- [Je92] Jensen, K. *Coloured Petri Nets*, Springer-Verlag, Berlin, 1992.
- [Ka93] Kaufmann S.A., *The Origins of Order*, Oxford University Press, 1993.
- [Ki97] Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, Aspect-Oriented Programming, *ECOOOP'97*, LNCS 1241, Springer-Verlag . June 1997.
- [Mu89] Murata, T. Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, Vol. 77, No 4, April, 1989, pp. 541-580.
- [LK98] Lopes C.V. and Kiczales G., Recent Developments in AspectJ *ECOOOP'98 Workshop Reader*, LNCS 1543, Springer-Verlag, 1998.
- [O99] Ossowski S., Co-ordination in Artificial Agent Societies. *LNAI*, LNCS, 1535, Springer, 1999.

- [Pe81] Peterson, J.L. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, N.J., 1981.
- [Sa98] Sandholm, T. (1998). Contract Types for Satisficing Task Allocation: I Theoretical Results. *AAAI98 Spring Symposium: Satisficing Models*, Stanford University, California, March 23-25, 1998.
- [Sm80] Smith, R. (1980). The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*. C-29(12):1104-1113.
- [Sil97] Silvescu A., Model of Interaction Categories (MIgories), Unpublished Draft, presented at the *IC-EATCS Annual Advanced School in "Models and Paradigms of Concurrency"*, 1997, CISM, Udine, Italy.
- [S96] Singh M., Toward Interaction-Oriented Programming. *Technical Report TR-96-15*, Department of Computer Science, North Carolina State University, May 1996.
- [Uh84] Uhr, L. *Algorithm-Structured Computer Arrays and Networks*. New York: Academic Press. 1984.
- [W99] Weiss, G. (ed.) *Multiagent Systems*. Cambridge, MA: MIT Press. 1999.