jDALMAS: A Java/Prolog Framework for Deontic Action-Logic Multi-Agent Systems

Magnus Hjelmblom and Jan Odelstad

Department of Mathematics, Natural and Computer Sciences
University of Gävle, SE-801 76 Gävle, Sweden
mbm@hig.se, jod@hig.se

Abstract. A norm-regulated Deontic Action-Logic Multi-Agent System (DALMAS) is regulated by a normative system consisting of norms, which are expressed in an algebraic notation based on the Kanger-Lindahl theory of normative positions. A general-level Prolog implementation of the abstract DALMAS architecture makes it possible to implement specific systems in Prolog. This work describes jDALMAS, a Java library that may be used to create DALMAS applications in Java. A jDALMAS application has a client/server architecture, where a Prolog implementation of a specific system acts as a logic server. Together, the general-level Prolog implementation and the jDALMAS packages offer a framework for implementation of specific systems. Two examples of such implementations are presented here.

Keywords: deontic action-logic, normative positions, norm-regulated MAS, DALMAS, logic server.

1 Introduction

During the last decade, the study of norm-regulated multi-agent systems has emerged as an active field of study within AI. The notion of norm-regulated DALMAS (deontic action-logic based multi-agent system) was introduced by Odelstad and Boman in [11]. DALMAS is an abstract architecture for a global clock, global state and global dynamics multi-agent system. A norm-regulated DALMAS is regulated by a normative system which consists of norms expressed in an algebraic notation (cf. [8], [9]) based on the Kanger-Lindahl theory of normative positions [7].

In earlier work, a general-level Prolog implementation of the DALMAS architecture was developed. [5] The implementation consists of a Prolog module, *dnrDALMAS*, which may be used to implement specific DALMAS systems as logic servers. This paper presents *jDALMAS*, a general Java library that may be used to create Java applications that connect to dnrDALMAS servers. Together with the dnrDALMAS module, the jDALMAS library offers a framework for the implementation of specific DALMAS systems in Java, for example as Swing applications with graphical user interfaces.

Two such graphical applications have been developed to illustrate the function and use of the jDALMAS library. The Java code for jDALMAS and the two specific

A. Håkansson et al. (Eds.): KES-AMSTA 2009, LNAI 5559, pp. 110–119, 2009.

implementations is available in electronic form, together with the Prolog code for dnrDALMAS and the two specific logic servers.

The abstract architecture for DALMAS, and its theoretical foundations, is outlined in section 2.1 - 2.3. Two specific systems are described in section 2.3.1. Section 2.4 gives an overview of the dnrDALMAS implementation, and a description of jDALMAS and the two specific implementations is given in section 3.

2 Background

2.1 Deontic Action-Logic

The term *action-logic* will be used in this paper to denote a logical system which includes the binary action operator *Do*. The intended interpretation of $Do(x, p)$ is "x sees to it that p".

Deontic logic may be used to create normative sentences, using norm-building operators such as *Shall* or *May*. The intended interpretation of *Shall* is "it shall be that" ("it shall be the case that"), and the intended interpretation of *May* is "it may be that". A conditional normative sentence consists of a combination of a descriptive sentence and a purely normative sentence. A typical conditional norm has the form $p \rightarrow Shall\ s$. The interpretation of this sentence would be "if p, then it shall be that s".

Stig Kanger contributed to the deontic logic by combining it with action-logic. [6] For example, the intended interpretation of $Shall\ Do(x, q)$ is "it shall be the case that x sees to it that q". The result of combining *Shall* or *May*, *Do* and $\neg$ is a powerful language for expressing normative sentences. A conditional norm may for example have the form $c(x,y) \rightarrow Shall\ Do(x,\neg d(y))$. According to the logic of *Shall* and *Do*, this means that "if $c(x,y)$, then it shall be the case that x sees to it that not $d(y)$". A more detailed presentation of deontic action-logic is given in [5], section 3.

The use of deontic logic within the design of multi-agent systems has been explored by many researchers. One example is the use of Constraint Handling Rules (CHR) to express deontic constraints within the area of agent communication. [2] Another example is IMPACT [3], an agent platform where deontic operators of permission, obligation and prohibition is the basis for the specification of what an agent is obliged to do, may do or cannot do. A third example is Sergot's Norman-G [12], a Prolog program based on the theory of normative positions.

2.2 Normative Positions

The Kanger-Lindahl theory of normative positions is based on Kanger's "deontic action-logic". The theory contains three systems of types of normative positions. The simplest of these systems is a system of seven one-agent types of normative positions, based on the logic of *Shall* and *Do*. [7]

$\mathbf{T}_i$ (where $1 \leq i \leq 7$) denotes the i:th type of *one-agent positions*. For example, $\mathbf{T}_2(\omega,d)$ denotes the deontic action-logic sentence $May\ Do(\omega,d) \wedge May\ Pass(\omega, d) \wedge \neg May\ Do(\omega, \neg d)$, where $Pass(\omega, d)$ is an abbreviation of $\neg Do(\omega,d) \wedge \neg Do(\omega,\neg d)$. The complete list of one-agent types is presented in [5], section 4, together with a detailed discussion and references.

The types $\mathbf{T}_1 - \mathbf{T}_7$ may be used as operators on descriptive conditions to get deontic conditions. If, as an example, d is a unary condition, then $T_i d$ $(1 \leq i \leq 7)$ is the binary condition such that

$$T_i d(y,x) \text{ iff } \mathbf{T}_i(x,d(y))$$

where $\mathbf{T}_i(x,d(y))$ is the i:th formula of one-agent normative positions.

2.3 Norm-Regulated DALMAS

In recent years, the study of norm-regulation of mult-agent systems has developed into a sub-discipline of AI called Normative Multi-Agent Systems, attracting the attention of specialists from different areas such as computer science, logic, sociology, and cognitive science. Central topics include the the use of norms as a mechanism in multi-agent systems and the use of multi-agent systems to study the concept and theories of norms and normative behaviour. An overview of this field of study is given by Boella et al. in [4], where ten research challenges for the NORMAS community are identified.

The abstract DALMAS architecture is one possible approach within this area. A deterministic DALMAS ***D*** is defined formally in [11], pp. 152f, as an ordered 9-tuple $\langle \Omega, A, S, A_f, \Delta, \Pi, \Gamma, \tau, \gamma \rangle$. The arguments are specific sets, operators and functions which are models for the theory of DALMAS, defining the unique features of a specific DALMAS:

1. Ω is the *set of agents* in ***D***.
2. A is an *act set*. An element a in A is a function such that $a(\omega,s) = s^+$ means if agent ω performs act a in state s, then the resulting state will be s^+.
3. S is the *state space* of ***D***, that is, the the set of all states that may be reached when the agents perform feasible actions.
4. A_f is a function such that $A_f(\omega,s)$ is *the set of feasible acts* for agent ω in state s.
5. Δ is a deontic structure-operator, such that $\Delta(\omega,s)$ is ω's *deontic structure* on $A_f(\omega,s)$ in state s. In a *simple* DALMAS, the deontic structure for ω in state s is the set of permissible actions for ω in state s. An act is permissible if it is not prohibited by the DALMAS's normative system.
6. Π is a preference structure-operator such that $\Pi(\omega,s)$ is ω's *preference structure* on $A_f(\omega,s)$ in state s. The preference structure is an ordering of the acts in $A_f(\omega,s)$ according to the "utility" of the acts for the agent. In other words, Π determines which acts are the most preferable for the agent in the current state.
7. Γ is a function such that $\Gamma(\omega,s)$ is *the set of actions for* ω *to choose from* in state s, ordered according to the preference structure. In a *simple* DALMAS, the choice-set consists of the most preferred of the permissible actions.
8. τ is a turn-operator such that $\tau(\omega_1) = \omega_2$ means that ω_2 is to move after ω_1.
9. γ is a tie-breaking function, where $\gamma(\{a_1, \ldots, a_n\}) = a$ means that a is *the act to choose* out of a set of permissible and equally preferred actions.

When an agent is to move, it chooses an act out of a set of feasible acts. This leads to a new state, depending on the state of the system when the act is performed. The choice of act is determined by the combination of the DALMAS's preference structure and deontic structure.

The representation of norms is based on Lindahl-Odelstad's algebraic representation of normative systems, which in turn uses the Kanger-Lindahl theory of normative positions. For example, if c is a binary condition and d is a ternary condition, the norm $\langle M_1c,T_7d\rangle$ may be interpreted as $M_1c(\omega_1,\omega;\ \omega_m,s) \rightarrow T_7d(\omega_1,\omega_2,\omega;\ \omega_m,s)$ where ω_i is an agent, ω_m is the agent to move and s is the current state of the system. A deeper discussion is given in section 5 of [5].

2.3.1 Examples: Colour and Form and Waste-Collectors

Two simple systems, Colour & Form and Waste-collectors, are used in [5] and this paper as running examples to illustrate the ideas behind DALMAS. The Waste-collector system was originally used in [11] as an example, inspired by [14].

Colour & Form is a very simple multi-agent system consisting of only two agents called chroma and forma. States of the system are represented by the values of the colour ("black" or "white") and form ("circle" or "square") attributes for each agent.

The agents can choose between two acts: *change colour* and *change form*. The agents' behaviour is controlled by

1. a simple utility function which is defined such that agent chroma prefers to change colour, and agent forma prefers to change form; and
2. a normative system containing one norm that prohibits an agent to choose an act leading to a state where both agents have identical attribute values.

In other words, if $\omega_1 \neq \omega_2$, then the moving agent must not act so that all of ω_1*'s* attributes are identical with all of ω_2*'s* attributes. This norm may be expressed in logical form (with the universal quantifier $\forall$ omitted) in the following way:

$$\mathit{Diff}(\omega_1,\omega_2;\ s) \rightarrow \neg \mathit{May}\ \mathit{Do}(\omega_1,\mathit{Eq}(\omega_1,\omega_2;\ s))$$

Diff is defined such that $\mathit{Diff}(\omega_1,\omega_2;\ s)$ iff $\omega_1 \neq \omega_2$, and *Eq* is defined such that $\mathit{Eq}(\omega_1,\omega_2;\ s)$ iff agents ω_1 and ω_2 have identical attributes in state s. Using the appropriate T_i operator (see section 2.2), the norm may be expressed

$$M_1\mathit{Diff}(\omega_1,\omega_2,\omega;\ \omega_m,s) \rightarrow T_7\mathit{Eq}(\omega_1,\omega_2,\omega;\ \omega_m,s)$$

where the operator M_1 unifies agent ω_1 with the acting agent ω_m. The algebraic form of this norm is $\langle M_1\mathit{Diff}, T_7\mathit{Eq}\rangle$.

The **Waste-collector** system is a system of agents operating in an environment which consists of a grid of squares ordered in rows and columns. Each square is assigned coordinate in the form of an ordered pair of integers. Some squares contain an amount of waste, represented by a number. The agents can move one square at a time in four directions: up/north, down/south, left/west and right/east, and at the same time pick up waste from the square where the agent starts its move. The agents may also pass[1], that is, do nothing except pick up waste (if any) from the current square. To summarize, an agent may choose one of the following actions: go_{north}, go_{south}, go_{west}, go_{east}, or *pass*. An agent in the system tries to cooperate with other agents to collect as much waste as possible. Each waste-collector has a utility function, such that the utility of an act depends on the amount of waste in the squares surrounding the target

[1] Note that the *pass* act, meaning that an agent does nothing, should not be confused with the *Pass* operator, meaning that an agent is passive with regard to some state-condition s.

square. Also, there are some restrictions on how the waste-collector agents may act, especially on how they may move near other agents. These restrictions may be expressed as norms in a normative system. For simplicity, let us consider a normative system with a single norm stating that it is not permissible to move to a square that is already occupied by another agent. Omitting the ∀ quantifier, this norm may be expressed in logical form in the following way:

$$Diff(\omega_1,\omega_2;\, s) \rightarrow \neg May\ Do(\omega_1, Lap_9(\omega_1,\omega_2;\, s))$$

The intended meaning of $Lap_n(\omega_1,\omega_2)$ is that the protected spheres (that is, the square where the agent is presently located and the eight squares surrounding it) overlap with n squares. See figure 1 of [5] for an illustration. This may be expressed in algebraic form as $\langle M_1 Diff, T_7 Lap_9 \rangle$.

2.4 dnrDALMAS

Previously, a Prolog implementation of the abstract DALMAS architecture has been developed. [5] The implementation consists of a Prolog module, *dnrDALMAS*, which may be used to create standalone text-based applications or logic servers for specific DALMASes. The module is written in SICStus Prolog, version 4.

The dnrDALMAS module contains a set of predicates for creating and initialising specific DALMAS implementations and querying the state of the system. An implementation of a specific system defines a set of (user-defined) primary predicates written in Prolog. These primary predicates correspond to the primary functions characterizing the system. The heart of the implementation is the `prohibited/3` predicate, which determines if an act is permissible or not according to the norms in the normative system.

Using dnrDALMAS, two implementations of the example systems in section 2.3.1 have been created. These implementations function both as standalone applications with simple text-based user interfaces and as logic servers within the jDALMAS framework.

The implementation of dnrDALMAS and the two implementations of specific systems are thoroughly described and discussed in [5]. The details will not be repeated here. The code is publicly available via *sourceforge.net*.

3 The jDALMAS Framework

jDALMAS is a general Java library consisting of a set of Java classes that can be used to create Java applications that communicate with dnrDALMAS servers. The Java part of the application may then provide for a (graphical) user interface, while the dnrDALMAS server provides for the DALMAS logic. The code for jDALMAS is publicly available via *sourceforge.net*.

A jDALMAS application has a client/server architecture, where a Java client communicates with a dnrDALMAS server.

3.1 Server Side

The concept of a logic server is somewhat analogous to a database server, but a logic server provides for application logic rather than for data storage and persistence.

Basically, the server side of a jDALMAS application consists of a dnrDALMAS server, which is a logic server for a specific system built with the dnrDALMAS Prolog module. The `prologbeans` module [13] in SICStus Prolog is used to initialize and start a server listening on a certain (user-defined or OS-assigned) port, register general queries that are to be accepted by the server, and register a set of event listener callback predicates.

To facilitate the administration of a dnrDALMAS server, the jDALMAS framework provides for a simple graphical control panel for the server.

3.2 Client Side

A jDALMAS client is a Java Swing applet. It uses the PrologBeans Java library [13] in SICStus Prolog to communicate with the dnrDALMAS server. The user interface consists of a generic control panel (see figure 1) that lets the user configure the specific system. Through the panel, the user may:

- add agents to the agent set;
- add acts to the act set;
- register and/or update functors of primary and secondary Prolog predicates;
- add functors of ground-predicates to the ground set;
- add functors of consequence-predicates to the consequence set; and
- add norms to the norm set.

To create an implementation of a specific system, the user needs to perform the following steps:

- create Prolog definitions of the appropriate predicates, for example user-defined primary and/or secondary predicates, or state-conditions used in the grounds or consequences of norms;
- deploy the user-defined Prolog files on the server;
- create a "logic handler" that translates between the client's own representation of knowledge and the representation used by the dnrDALMAS server; and
- determine how the specific system's knowledge base shall be visualized on the client side, and write the necessary Java code.

When the client control panel is started, the user adds agents to the agent set and acts to the act set. The user also adds functors of state conditions to the ground set and consequence set, and adds norms to the norm set. Finally, the user registers functors of primary or secondary predicates, and adds knowledge to the knowledge base which is initially empty.

The client application should display the specific system's current knowledge base in an appropriate way, and let the user perform the desired actions on the system. For instance, the user may want to ask the system for its next situation, which is the resulting situation when the agent to move chooses and performs its most preferred act that is not prohibited by the norm-system.

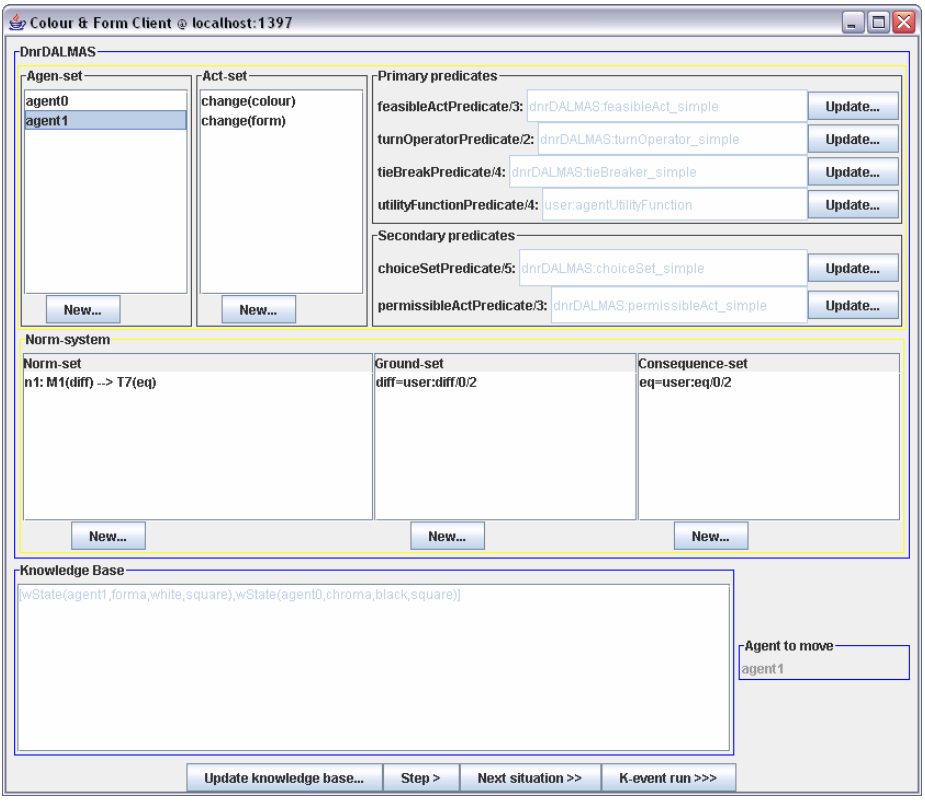

Fig. 1. Control panel for a specific jDALMAS client

3.2.1 Example 1: CFDALMAS

CFDALMAS is a graphical Java client for a Colour & Form system; see section 2.3.1. Aside from the client control panel shown in the previous section, the user interface consists of an agent frame for each agent. The agent frame (figure 2) shows a graphical representation of the agent's state.

3.2.2 Example 2: WasteDALMAS

WasteDALMAS is a graphical Java client for the Waste-collector system described in section 2.3.1. The user interface consists of two parts: a client control panel and state

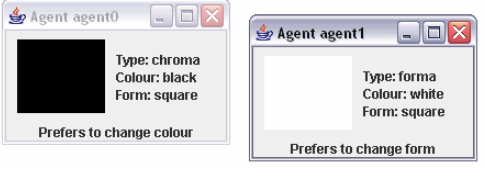

Fig. 2. Graphical representation of a state of the Colour & Form system

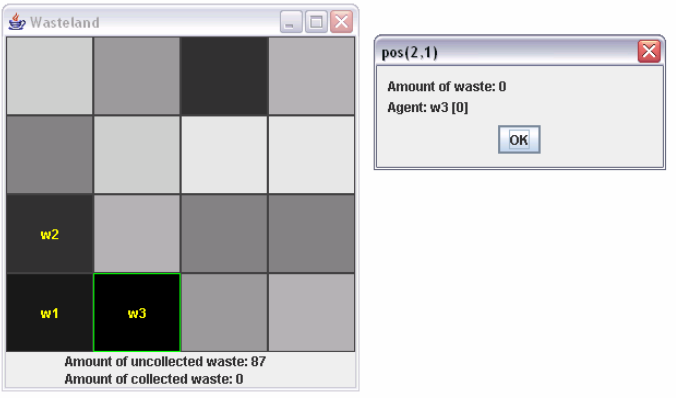

Fig. 3. Graphical representation of a state of the Waste-collector system

frame that shows a graphical representation of the state of the system. The client control panel is very similar to the one shown in section 3.2.

The state frame (see figure 3) shows the state of each square in the wasteland grid, including the position of agents and the amount of waste carried by each agent. The background colour of a square indicates how much waste the square contains; the darker the square, the less waste it contains. The square containing the agent which is next to move is marked by a green border. A dialog frame which contains more detailed information about a certain square is shown if the user clicks on the square.

4 Discussion

The jDALMAS framework is still work in progress. The current version works well, but some improvements and extensions can be made. An issue to deal with is how the implementations should behave in situations where the deontic structure is empty, that is, when all feasible acts for the acting agent are prohibited by the normative system. The current version of the framework leaves this issue to the user: if, in a given situation, there are no permissible acts, the run of the system fails. Another approach in this situation could be to let the moving agent be "confused", in the sense that it does nothing. However, this would mean that the "pass" (i.e. "do nothing") act will always be permissible, even if prohibited by the normative system. Another issue is to extend the framework to handle non-elementary norms, i.e. norms whose grounds and/or consequences are Boolean expressions.

5 Conclusion and Future Work

jDALMAS offers a framework for the creation of specific DALMAS applications, making it possible to implement a wide range of norm-regulated multi-agent systems. It also offers the possibility to design and experiment with different normative systems for a given agent system. For a system such as the Waste-collectors it would be

possible to experiment with different combinations of normative systems and utility functions, to see which normative system gives the best overall performance or desired behaviour according to some evaluation function or the user's preferences. If combined with some learning mechanism, such as a genetic algorithm, the system itself could try to find an optimal combination for a given class of problems. [11], pp. 164f.

Other applications of DALMAS can be found in many areas. Somewhat related are the fields of education, edutainment and entertainment. A norm-regulated DALMAS could for example be part of a game environment which lets the user act as a "legislator", creating and playing with different "laws" for the game world.

Another idea is to use the framework for experiments involving humans. Is it, for example, possible for human observers to conclude which normative system is used by a particular application, just by examining the application's behaviour? This could be an interesting tool within many different disciplines.

It could also be rewarding to explore the use of DALMAS within decision support. Possible applications within this area could be to use norm-regulated systems as an analytical tool for the creation and/or evaluation of decision-theoretical expert systems. In a given situation, such a system could for example facilitate decision making by eliminating those feasible acts that are prohibited by some normative system.

Ahonen-Jonnarth and Odelstad have discussed the area of forest cleaning as a possible area of application for norm-regulated multi-agent systems. A single cleaning agent may be regarded as a one-agent system where the agent's decision making is regulated by a normative system. [1], [10]

References

[1] Ahonen-Jonnarth, U., Odelstad, J.: Evaluation of Simulations with Conflicting Goals with Application of Cleaning of Young Forest Stands. In: Proceedings of ISC 2006 (Fourth Annual International Industrial Simulation Conference), Palermo, Italy, June 5-7 (2006)

[2] Alberti, et al.: Logic Based Semantics for an Agent Communication Language. DEIS Technical Report no. DEIS-LIA-03-001. LIA Series no. 62

[3] Arisha, K.A., Ozcan, F., Ross, R., Subrahamanian, V.S., Eiter, T., Kraus, S.: IMPACT: A platform for collaborating agents. IEEE Intelligent Systems 14(2), 64–72 (1999)

[4] Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. Computation and Mathematical Organizational Theory, special issue on normative multiagent systems 12(2-3), 71–79 (2006)

[5] Hjelmblom, M.: Deontic Action-Logic Multi-Agent Systems in Prolog. Institutionen för matematik, natur- och datavetenskap, Högskolan i Gävle (2008) ISSN 1403-8749;30, http://hig.diva-portal.org/smash/record.jsf?pid=diva2:118137

[6] Kanger, S.: New foundations for ethical theory, Part 1 (1957). In: Holmström-Hintikka, et al. (eds.) Collected Papers of Stig Kanger With Essays on His Life and Work, pp. 99–119. Kluwer, Dordrecht (2001)

[7] Lindahl, L.: Position and Change: A study in Law and Logic. Reidel, Dordrecht (1977)

[8] Lindahl, L., Odelstad, J.: An Algebraic Analysis of Normative Systems. Ratio Juris 13(3), 261–278 (2000)

[9] Lindahl, L., Odelstad, J.: Normative Positions within an Algebraic Approach to Normative Systems. Journal of Applied Logic 2, 63–91 (2004)

[10] Odelstad, J.: Agents, Norms and Forest Cleaning. In: Boella, G., van der Torre, L., Verhagen, H. (eds.) Normative Multi-Agent Systems. Dagstuhl Seminar Proceedings, 07122 (2007) ISSN 1862-4405,
http://drops.dagstuhl.de/portals/index.php?semnr=07122

[11] Odelstad, J., Boman, M.: Algebras for Agent Norm-Regulation. Annals of Mathematics and Artificial Intelligence 42, 141–166 (2004)

[12] Sergot, M.J.: A computational theory of normative positions. ACM Trans. Comput. Logic (TOCL) 2, 581–622 (2001)

[13] SICStus Prolog User's Manual. The Intelligent Systems Laboratory, Swedish Institute of Computer Science, Kista (2005)

[14] Steels, L.: Cooperation between distributed agents through self organization. In: Decentralized, A.I., Demazeau, Y., Muller, J.P. (eds.) Proc. of Maamaw 1989, pp. 175–196. Elsevier Science Publisher, Amsterdam (1990)