Challenges in Exception Handling in Multi-Agent Systems

Eric Platon[1,2] and Shinichi Honiden[1]
[1]National Institute of Informatics, Sokendai
2-1-2 Hitotsubashi, Chiyoda
101-8430, Tokyo, Japan
{platon,honiden}@nii.ac.jp

Nicolas Sabouret[2]
[2]Laboratoire d'Informatique de Paris 6
8, Rue du Capitaine Scott
75015 Paris, France
nicolas.sabouret@lip6.fr

ABSTRACT

Exception handling has received little interest in the agent community despite its challenges to build more reliable agent systems. In this paper, we survey existing work on exception handling for Multi-Agent Systems. We tried to identify in the present literature what research directions are required and likely to improve current techniques. In particular, we think that the agent proactivity and context in the systems are potential characteristics to exploit for agent-level exception handling.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Distributed Artificial Intelligence—*Multiagent systems*; D.2.2 [**Software**]: Design Tools and Techniques

General Terms

Design, Reliability, Theory

Keywords

Agent-Oriented Software Engineering, Exception handling, Multi-Agent Systems

1. INTRODUCTION

Exception handling in Multi-Agent Systems (MAS) differs in many ways from exception handling in sequential and traditional distributed systems. Tripathi and Miller exposed the case of mobile agents in a MAS context. Exceptions conditions can be raised for 'standard reasons' (*e.g.*, zero divide), but also due to agent host node failure, security, and communication issues [21]. Beyond, the study of MAS sets forth the need to distinguish other types of exceptions that lie at a different abstraction level. Agents are indeed thought of as *autonomous entities*. The autonomy attribute entails some form of intelligent behaviors that must be *engineered* in applications. Such behaviors introduced by the designer in agents are source of both standard and agent-specific exception conditions. Although the definition of agent-level exception handling has not been formally determined yet, typical examples are inconsistencies in agent reasoning mechanisms, flawed knowledge, opportunities, or unexpected exchanges in interaction protocols [9].

The issue in this paper is that current work and exception handling facilities in MAS appear insufficient in facing the agent paradigm and the requirements of flexibility, robustness, and reliability. Little work recognizes the importance of exception handling in MAS, and the existing research has two shortcomings: work is either too pragmatic and misses, in our sense, part of the agent paradigm; or work is too theoretic and does not meet computing and programming constraints. Also, exceptions are mostly thought of as *problems* in the execution of the system, even though they were considered early as *opportunities* for monitoring [3]. In MAS, exceptions as opportunities are actually frequent and sometimes desired to improve the performance and flexibility of the system [15].

In this paper, we propose an explanation of what exception handling should be in MAS and why current achievements are unadapted. Our approach relies on a survey of the exception handling literature in the perspective of the agent paradigm. Although our survey cannot claim to be exhaustive, we think it encompasses most of the relevant research. We exploit this survey to identify the issues of current exception handling techniques in MAS, and we present research directions of interest based on our analysis.

The organization of the paper is as follows. Section 2 surveys the related or relevant literature for exception handling in MAS. Based on this survey, we identify agent proactivity and context as relevant research directions in section 3. In section 4, we describe a series of experiments we conducted to illustrate agent-level exception handling in an example application. Finally, we summarize in section 5 the challenges for exception handling in MAS and explain our intentions for future work.

2. RELATED WORK

The literature on exception handling for MAS remains scarce despite the challenges identified by Tripathi and Miller [21] and Klein et al. [7]. We discuss hereafter research work along two dimensions to emphasize the characteristics that we consider important in exception handling.

- Degree of distribution of the exception handling framework: MAS are typically distributed (physically or log-

SELMAS'06, May 22–23, 2006, Shanghai, China.

ically), but both central and distributed frameworks were designed.

- A scale that ranges from object (passive units of an application) to agents (autonomous units), inspired by the view on autonomy from Odell [12].

The two dimensions allow us to classify related work as shown in figure 1. The classification distributes the work that we consider representative. The star mark highlights work that focuses on benevolent agents (*i.e.*, willing to cooperate and not malicious).

2.1 The Guardian

In the bottom left corner, Miller and Tripathi occupy the area of work dedicated to object-oriented and reactive agent systems in a centralized way. Their approach named 'the guardian' is a set of software constructs represented as an agent to handle exceptions in a distributed-object system, with applications in the mobile agent context [21]. The approach is centralized as all agents refer to the guardian in case of problem. It is not totally centralized though, since a set of guardians can be allocated a subset of agents only (consequently the box on figure 1 is raised toward distributed approaches), but this model does not refer to any coordination mechanism among guardians.

The advantage of the guardian is to deal with exceptions in a similar way as for sequential systems. It focuses on the software aspect of the agent and it lets aside the reasoning parts (that is why the approach does not deal with proactive agents). A detailed example of exception handling is presented by Miller and Tripathi where the direct relationship with Java facilities can be observed [10]. The guardian assists a client-server system that implements the 'primary-backup' approach to deal with server-side failures [20]. If the primary server fails, a 'global exception' is raised, so that the guardian handles the error by asking the backup to take the role of primary, and by starting a new backup.

We think the guardian does not capture the characteristics of openness and agent encapsulation in the MAS paradigm. Open MAS accept at runtime the addition and removal of agents or any other elements in the system. There are minimal assumptions on agents and other elements that only need to verify a set of interfaces. In particular, the agent architecture can be of any kind, provided the interfaces are verified. One technical consequence is that open MAS are loosely coupled systems. The guardian initially targets distributed objects, so that it appears too tightly coupled for an agent system architecture. Another consequence of openness is the spectrum of 'agent profiles'. Malicious agents can be part of open MAS (which makes research on MAS difficult in general), along with benevolent agents. The guardian approach assumes that agents are benevolent, although security concerns are considered, and it does not cope currently with arbitrary agent profiles.

The encapsulation is a closely related matter as it guarantees some autonomy to the agent, which can ignore messages explicitly or answer false results. In other words, agents have a private state and they appear as 'black boxes' to others. In the guardian approach, access to the agent state is granted. Typically, the guardian is allowed to 'command' an agent, e.g. to wait or to restart a task, which does not verify the encapsulation and autonomy of agents.

2.2 The Sentinels

In the bottom right corner of figure 1, the sentinels approach from Haegg is applicable to agents with more proactive behaviors than the guardian [4]. Sentinels are agents introduced in a MAS application to provide a fault-tolerance service layer. The approach has been extended in the work of Klein et al. with an exception handler repository [6, 7]. The sentinels appear as a centralized solution, as explained by Haegg in its original work. However, it adds communication capabilities among agents that extend the guardian model toward a distributed framework. The sentinels were developed in a MAS research and it features more agent-specific exception handling capabilities. For example, sentinels are able to deal with problems in the agent beliefs, whereas the guardian focuses on software aspects. A detailed application from Haegg is a system and its sentinels for a power distribution company. Application agents negotiate energy consumption credits for load-balancing on an electric grid. Sentinels can detect and remedy to erroneous behaviors in negotiation processes by inspecting 'checkpoints' in the agent code.

Nevertheless, sentinels also violate assumptions of the agent paradigm. Encapsulation is not respected since sentinels can access and execute code in the so-called 'agent-head' [4], which should be a black-box. Similarly to the Guardian, agents are supposed benevolent and this hypothesis does not scale to open MAS.

2.3 Stigmergic systems

The upper central part of figure 1 refers to stigmergic systems [1]. Stigmergy is an interaction model where agents put marks in the environment (messages with no intended recipient) that other agents exploit to determine their next actions. Stigmergy models and allows to simulate the behavior of some social insects such as termites. One termite starts to build a nest by putting a piece of material on the ground (a mark). Other termites use this information to determine where to pile the piece they carry. Stigmergy is thus an indirect interaction model as there is no direct message passing. Stigmergic systems are shown to be particularly robust to exceptions such as the death or the failure of agents [13]. The robustness of these systems is mostly due to the high redundancy of agents, which reminds the choice for modularity of software architectures that could limit the impact of exceptions in sequential systems. There is little work on stigmergic systems that discusses robustness issues, and no work on exception handling to our knowledge. Although the robustness inherent to such systems entails that no significant advance might be expected in exception handling, recent extensions of stigmergic systems to proactive agents are to be demanding for such techniques [14] (that is why the box on figure 1 stretches toward proactive agents).

2.4 SaGE

The upper left part of figure 1 groups approaches that deal with objects and reactive agent exceptions in a distributed way. In the case of agents, Souchon et al. proposed the SaGE[1] framework [18]. SaGE completes the exception handling system of Java with facilities to handle agent-specific issues. In particular, SaGE provides a mechanism for 'concerted exception handling' to resolve errors depending on

[1]We could not figure out whether SaGE is an acronym or not.

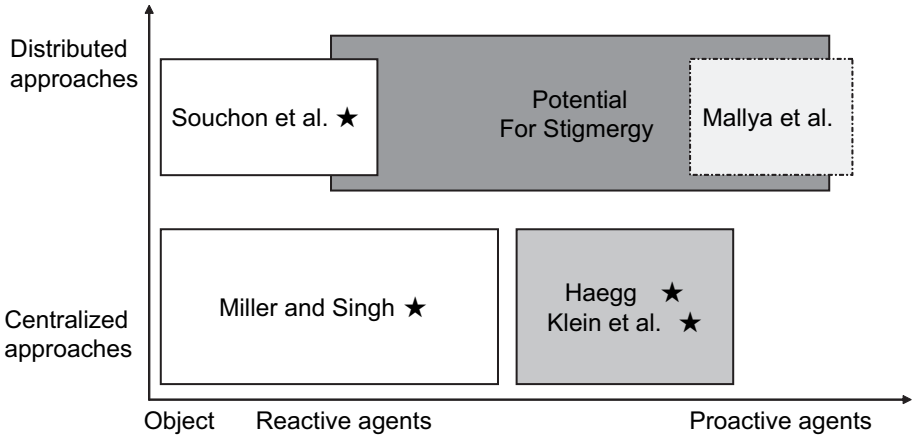

Figure 1: Related work over two dimensions: (x-axis) object to agent scale, (y-axis) degree of distribution. The star marks approaches for benevolent agents.

several and distributed agents. Souchon describes an example of such exceptions in a travel reservation scenario where service providers encounter a failure. When few providers fail, limited results can be generated in a degraded mode. Too many failures compared to the number of providers trigger a specific method in the agent code for concerted exception handling to terminate the transaction for the reservation properly.

Although proactive agents can leverage this approach (as demonstrated in the aforementioned paper), we classify this work for reactive agent exceptions only, since SaGE targets agents as software components rather than proactive entities. In this perspective, SaGE does not address agent specific issues due to their proactive abilities. Compared to the previous work presented in this section, SaGE complies further with the agent encapsulation hypothesis. However, SaGE does not scale to open system issues as it also assumes benevolent agents only.

2.5 Proactive agent exceptions

The last case we could distinguish in the literature is in the top-right corner of figure 1. The work of Mallya and Singh represents the little work that has been done in this area. They deal with exception handling for proactive agents in a fully distributed way [9]. This approach relies on agent interaction protocols named 'commitment protocols'. When such a protocol is not respected along agent interactions, an exception is raised and two formal methods allow agents to handle expected and unexpected ones. Expected exceptions are foreseen by the designer who could write a specific handler (here, another protocol), which is the most common construct if we consider, e.g., the `try{}catch()` statements in Java. Unexpected exceptions are not coded beforehand and some constructs allows to dynamically build a handler from a basic set.

This method has been illustrated for a hotel reservation protocol. An expected exception can be the case where there is no vacancy in the hotel. The system design usually foresee this issue and a specific handler is available in the system to deal with it. An unexpected exception can be the delegation of a reservation to another hotel in case of fire. At design time, the handling of such an exception might not have been prepared. Mallya and Singh propose to rely on an external exception handler repository to fetch a specific handler and merge it automatically with adequate system protocols.

Although this approach is very attractive and verifies the agent paradigm (encapsulation and openness), it still remains theoretical and it does not rely yet on validated results, even in later work [8]. The current issues seem the computational complexity of an handler selection and the dynamic assembly of new handlers.

2.6 Survey conclusion

Fig. 1 shows the spread of endeavors in dealing with exception handling in the MAS community, and the potential of some approaches. In the broader research on exception handling in Software Engineering, the approach for MAS can be compared, for instance, to the work of Murata and Borgida, where agents and humans are equivalent [11]. In the present paper, we rely on this body of work to identify research directions for MAS, toward a distributed approach for proactive agents that both respects the agent paradigm, as Mallya's work, and remains practical, with the experience gained from the other papers cited in this section.

3. CHALLENGE AND RESEARCH DIRECTIONS

MAS call for specific exception handling to deal with the autonomy of agents and their distribution. In this section, we elaborate on the major characteristics of MAS to identify challenges and research directions in proposing an appropriate handling facility.

3.1 Full-fledged MAS

In this paper, we consider full-fledged MAS as follows.

- Agent paradigm respect: The two main characteristics of agents are encapsulation (the black-box image) and autonomy.
- Distributed systems: The general case refers to multi-process systems that span over an arbitrary number of machines, or to multi-threaded systems that run on one machine.
- Open systems: Agents are not supposed to be collaborative or benevolent, encapsulation is enforced, and

the system infrastructure is reduced to a minimal set of services and their (standard) interfaces.

In Fig. 1, such properties define systems that would lie in the upper right corner. From our survey of related work, it appears that no approach exists that can deal with such system properties simultaneously and concretely. The survey refers to these three properties, but the current achievements are either abstract and difficult to engineer, or lack of generality as would be expected for a work similar to Goodenough [3]. In other words, the techniques exposed on figure 1 should be a foundation for the development of exception handling mechanisms that address full-fledged systems.

3.2 Characteristics of exception handling for MAS

In order to comply with the properties of MAS and their engineering, appropriate exception handling mechanisms should verify explicitly with the following properties.

- Generic: At first a set of recommendations for exception handling in MAS so as to be instantiated for different system requirements (e.g. business process, simulation) and languages.
- Concrete: The technique should allow to engineer concrete systems. MAS often rely on social or biological metaphors that should be interpreted in engineering terms.
- Non intrusive: Respect of the agent paradigm. Exception mechanisms external to agents should not control or inspect freely agent internals.
- Distributed: Agents are autonomous and they should handle exceptions by themselves or among groups of agents. Centralized handling can also be considered when necessary and compatible with the other properties of this list.
- Redundant: The presence of non-collaborative agents in open systems entails a need for heuristics to find 'alternatives' when one cannot handle an exception.

Concerning the two first requirements, we tend to think that an appropriate work would be similar to the approach of Goodenough with the development of extensions to existing agent frameworks, similarly to Souchon et al. [3, 18]. The agent community already proposes several languages and platforms whose exception handling facilities are mostly limited to usual mechanisms inherited from Java for the essential part [2, 5]. Genericness and concreteness are thus to be potentially useful to extend them. These two first ones are left for future work as they directly refer to the expected achievements a MAS exception handling solution.

The three last requirements are specific to MAS and our study led us to considering two characteristics of agents to treat exceptions at their level, namely their proactivity and their contexts. In the remainder of this section, we focus on the three last requirements.

Non-intrusive mechanisms lead to exploit the proactivity of agents for handling agent-level exceptions. The example of the guardian exemplifies an approach that is intrusive in the sense the guardian selects the behavior of agents. A mechanism that relies on the agent capability to reason on exceptions (proactivity) seems appropriate to verify the non-intrusion property.

Considering the properties of distribution and redundancy, history of exception handling shows that an important system feature is modularity, as the syntactic units of Goodenough or, later, the object-oriented design [3]. Modularity in MAS is due to agents having only partial knowledge about the whole system. This agent 'local scope' means that agents have only access to a subset of their environment that we call agent *context*. The work of Miller and Tripathi on the guardian and the work of Haegg on the sentinels show that the context can be efficiently exploited in distributed systems to determine appropriate exception handlers [10, 4]. In their models, the context refers to the information available to executing processes (e.g. fields of an object). In the agent paradigm, the notion of context is related to information and resources available in the agent environment [22]. In other words, a MAS exception handling mechanisms should allow agents to leverage information and resources in their contexts. Agent proactivity then permits to reason with extended data and potentially be able to handle or delegate an exception.

3.3 Potential Research Directions

The result of our current analysis is that research on the agent context would benefit to exception handling techniques in MAS. The current agent context is in practice reduced to information received by message-passing, which is the usual way of interacting among agents. However, the number of messages passed to an agent is decided by the protocols authorized for this agent. It means that a given agent only knows about its context through such a restricted set of messages. Although these messages are usually sufficient for activities under normal conditions, they contain little extra information that agents can exploit in case of exception (either reactively or pro-actively) to try to terminate properly or resume their execution.

For the above reasons, we think that a potential research direction is the design of techniques to enrich the agent context automatically at runtime with application-dependent information. Enriching the context with care should yield timely information to treat some exception conditions or organize a concerted resolution [10, 18]. A simple example of enriching the context is an agent-based supply chain system. An agent A of the chain receives orders from several agents and delegates some sub-tasks to some others. A basic context of A consists of information on the identity of acquaintances. In case of failure of one acquaintance, A cannot deduce such an event from the context and it must be explicitly informed by another agent or a service of the infrastructure. A mechanism could however automatically enrich the context with agent state information, so that A would just check it. Such an approach would verify the criteria we identified for full-fledge MAS. The issues for such direction would then be how to build the right context dynamically, and how agents can exploit it in an efficient manner. We think that exception handling in MAS should continue this line of work: Exception handling in MAS would be the combination of exception handling for sequential and distributed systems [21], and the exploitation of a context with a rich semantics adapted to agents.

A second research direction is closely related to the first

one. When the context provides extra information, agents must be able to exploit them, so that adapted reasoning mechanisms are necessary. Several architectures have been proposed for proactive agents and it seems that none integrates such mechanisms, even though the KGP model allows to treat 'unexpected messages' that could be thought of as exceptions [19]. Another area of work focuses on goal-driven agents that execute hierarchies of plans. When a plan fails in realizing a goal, alternative plans are deduced by exploring the hierarchy. In addition, exceptions can be *faults* as in any software, but the cases of *opportunities* are frequent in MAS [15] and it requires specific reasoning schemes to distinguish them from errors to exploit them adequately. Agent proactivity is therefore a relevant research direction to explore advanced but efficient mechanisms for exploiting opportunities.

The last research direction is an issue of integration of different exception techniques. The related work shows several endeavors for extending the techniques for sequential and distributed systems with agent-specific ones. However, the requirements we presented in this section emphasize the paradigm shift between agents and traditional techniques. For example, the technique of Mallya and Singh [9] appears difficult to integrate with traditional ones. Furthermore, an interesting research direction would be to study when some exceptions at the agent-level can imply lower-level exceptions, and conversely (*e.g.*, when exceptions are raised, they are sometimes wrapped or transformed into more adapted exceptions). Multi-level exceptions might create intricate and powerful situations, but we envision that formal methods such as process algebra may be necessary to deal with such cases.

4. PRELIMINARY EXPERIMENTS

Our ongoing work follows the research directions identified in this paper since recently. We are convinced that some important results can be obtained from these possible tracks. In our latest endeavors, we developed an agent-oriented event notification system that enriches the context of each agent with 'potentially relevant information' for their activities. Early results in preliminary experiments show that agents can leverage the enriched context to take advantage of some situations [15].

The experiments were conducted on an electronic market place developed for trading agents (type 'bazaar'). The purpose of the system was to show how often agents can face exceptional events and how they can exploit these events for their own benefits. In the market, buyer and seller agents follow the Contract Net protocol (CNet) to trade some items [17]. This protocol has a strict order: Agents cannot join a running protocol and they must either create their own CNet or wait for being invited. The agent context is limited to the information received by the messages specified in the protocol. Our experiments first runs the system with this capabilities only.

In another series of runs, the agent context is enriched with two types of information. Overheard messages and agent states are introduced dynamically by a specific service depending on the market topology (called environment [16]). The topology defines how agents are related, i.e. agents that trade the same type of items (either buyer or seller) are in the same neighborhood of the topology. An agent can thus access in its context to the information in messages from the agents in the CNet it has joined, to information in messages of other agents in the neighborhood (overhearing), and to state information of nearby agents. Overheard messages allow agents to be aware of opportunities. If a deal fails exceptionally, an agent is further aware of potential agents to contact. State information is the list of items an agent is interested in. In particular, seller agents can know which buyers should be contacted after a too long and exceptional idle period.

In these experiments, agents process in order the messages they receive. The reasoning capabilities of agents are reaction rules adapted to the CNet and to the enriched context. For example, sellers react respectively to a message and an overheard message with the following rules.

Algorithm 1 Two example rules of seller agents

if message received & I am recipient **then** send offer **end if**	**if** message received & I am *not* recipient **then** remember alternative **end if**

The left rule is a standard message in the CNet, whereas the right rule aims at storing information that can be exploited in case of exception. In this experiment, the capabilities are fixed by design and agents can only deal with expected exceptions. This work only simulates an open MAS, even though agents autonomously exit the market, enter, and trade on the market (behavior specified as a state machine). Agents have the rational behavior to try to maximize the number of successful deal with the lowest price they can negotiate.

In this series of experiments, exceptions can be interestingly opportunities of interactions. In fact, this scenario has been designed to show that agents can frequently face such cases in some settings (statistically). The following Fig. 2 shows the number of failed deals in different market configurations, *i.e.* different number of buyers B and sellers S. Failures to deal items in the market can be of different kinds, including receiving better offers and canceling other negotiations, or lacking time to contract some clients.

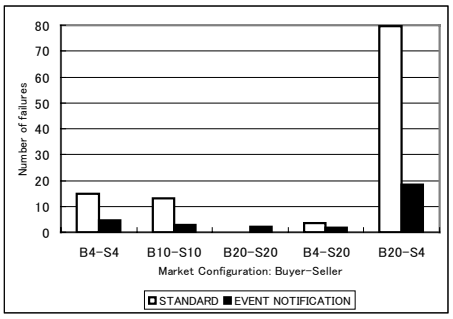

Figure 2: Number of failed deals in different market configurations

The graph values rely on statistical averages of several runs in different configurations. In most cases, the system with event notification allows reducing significantly the number of failed deals by offering opportunities and initiative mechanisms to the agents. In this experiments, we did not allow agents to resume negotiations. The B20-S20 run differs from others as the event notification system does not

perform better. Our interpretation is that in such a balanced and 'crowded' market the number of agents is such that the probability of failure is lower than other settings. The event notification has a communication cost to enrich the context with overheard messages and state information. The overhead cost becomes higher than the benefit gained from the event notification, which causes a slightly worse performance.

5. CONCLUSION AND OPENING

This paper attempts to identify challenges and research directions of exception handling in MAS. We consider MAS as open, distributed, and strongly abiding by the agent paradigm. As such, our survey in the field shows that the way to go is still long to reach the goal of exception-safe and reliable MAS that can be compared to traditional systems.

Our current analysis identified so far three research directions that are potentially relevant to achieve better exception handling techniques.

- Enriching the agent context to be able to deal with exceptions
- Exploiting accordingly the agent proactivity, either for faults or opportunities handling
- Integrating agent-level exceptions with traditional ones

Our future work aims at pursuing these research directions and to propose a generic and concrete agent-oriented exception handling approach. The genericness is expected to be validated by applying the approach to an existing agent architecture.

Acknowledgment

The authors would like to thank the anonymous reviewers of this paper who participated in improving significantly the initial version of this work.

6. REFERENCES

[1] Brueckner, S. *Return from the Ant — Synthetic Ecosystems for Manufacturing Control*. PhD thesis, Humboldt University, Berlin, Germany, 2000.

[2] El-Fallah-Seghrouchni, A., and Suna, A. Claim and sympa: A programming environment for intelligent and mobile agents. In *Multi-Agent Programming*, R. H. Bordini, M. Dastani, J. Dix, and A. El-Fallah-Seghrouchni, Eds., vol. 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2005, pp. 95–122.

[3] Goodenough, J. B. Exception handling: Issues and a proposed notation. *Commun. ACM 18*, 12 (1975), 683–696.

[4] Haegg, S. A sentinel approach to fault handling in multi-agent systems. In *DAI* (1996), C. Zhang and D. Lukose, Eds., vol. 1286 of *Lecture Notes in Computer Science*, Springer, pp. 181–195.

[5] Jade agent framework. http://jade.tilab.com/, ver. 2005.

[6] Klein, M., and Dellarocas, C. Exception handling in agent systems. In *Agents* (1999), pp. 62–68.

[7] Klein, M., Rodríguez-Aguilar, J. A., and Dellarocas, C. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous Agents and Multi-Agent Systems 7*, 1-2 (2003), 179–189.

[8] Mallya, A. U. *Modeling and Enacting Business Processes via Commitment Protocols among Agents*. PhD thesis, North Carolina State University, Raleigh, United States, 2005.

[9] Mallya, A. U., and Singh, M. P. Modeling exceptions via commitment protocols. In *Autonomous Agents and Multi–Agent Systems* (New York, NY, USA, 2005), ACM Press, pp. 122–129.

[10] Miller, R., and Tripathi, A. The Guardian Model and Primitives for Exception Handling in Distributed Systems. *IEEE Trans. Software Eng. 30*, 12 (2004), 1008–1022.

[11] Murata, T., and Borgida, A. Handling of irregularities in human centered systems: A unified framework for data and processes. *IEEE Trans. Software Eng. 26*, 10 (2000), 959–977.

[12] Odell, J. Objects and agents compared. *Journal of Object Technology 1*, 1 (May-June 2002), 41–53.

[13] Parunak, H. V. D. "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems. *Annals of Operation Research 75* (1997), 69–101.

[14] Parunak, H. V. D. Expert Assessment of Human-Human Stigmergy. Analysis for the Canadian Defence Organization, Altarum Institute, Ann Arbor, Michigan, May 2005.

[15] Platon, E. Artificial intelligence in the environment: Smart environment for smarter agents in open e-markets. In *Proceedings of the Florida Artificial Intelligence Research Society* (2006), AAAI.

[16] Platon, E., Sabouret, N., and Honiden, S. Overhearing and Direct Interactions: Point of View of an Active Environment, a Preliminary Study. In *Proceedings of Environment for Multi–Agent Systems at AAMAS'05* (2005), D. Weyns, H. V. D. Parunak, and F. Michel, Eds.

[17] Smith, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers 29*, 12 (1980), 1104–1113.

[18] Souchon, F., Dony, C., Urtado, C., and Vauttier, S. Improving exception handling in multi-agent systems. In *SELMAS* (2003), C. J. P. de Lucena, A. F. Garcia, A. B. Romanovsky, J. Castro, and P. S. C. Alencar, Eds., vol. 2940 of *Lecture Notes in Computer Science*, Springer, pp. 167–188.

[19] Stathis, K., Lu, W., Kakas, A. C., Demetriou, N., Endriss, U., and Bracciali, A. PROSOCS: A platform for programming software agents in computational logic. In *From Agent Theory to Agent Implementation* (2004).

[20] Tanenbaum, A. S. *Distributed Operating Systems*. Prentice Hall, 1994.

[21] Tripathi, A., and Miller, R. Exception handling in agent-oriented systems. In *Advances in Exception Handling Techniques* (2000), A. B. Romanovsky, C. Dony, J. L. Knudsen, and A. Tripathi, Eds., vol. 2022 of *Lecture Notes in Computer Science*, Springer, pp. 128–146.

[22] Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. Environments for Multiagent Systems, State-of-the-Art and Research Challenges. In *Environment for Multi–Agent Systems'04* (2005), D. Weyns, H. V. D. Parunak, and F. Michel, Eds., vol. 3374 of *LNAI*, Springer, pp. 1–47.