Management-by-Exception - A Modern Approach to Managing Self-Organizing Systems

René Schumann, Andreas D. Lattner and Ingo J. Timm

Information Systems and Simulation, Department of Computer Science and Mathematics, Goethe University Frankfurt
Robert-Mayer-Str. 10, 60325 Frankfurt/Main, Germany
E-mail: {reschu;lattner;timm}@cs.uni-frankfurt.de

Abstract: A well established design technique for software systems is to define their application context and its behavior in all possible scenarios. Those systems are typically deterministic in terms of their reaction to inputs from their environment. A disadvantage of those systems is their lack of flexibility. They fail when they have to work in a context not regarded at design time. It was shown that multiagent systems can offer such a flexibility. But from an external position their behavior cannot always be anticipated, which in fact is a major obstacle using this technology in commercial applications. Therefore, we propose an approach that combines the advantages of deterministic behavior and flexibility. We use a multiagent system with an internal management structure. Using the management-by-exception approach allows a maximum of local decision freedom and therefore of flexibility, while ensuring an acceptable overall system performance. **Keywords**: strategic management, multi-agent system, self-organization, management-by-exception

1. Introduction

There is a trend of decentralization in business applications. Global competition results in the demand for shorter product life cycles and forces companies to cut down development and delivery times. Thus, modern software systems are embedded in highly dynamic, complex, and distributed environments. But software engineering of systems and processes is continuously increasing in complexity of design and functionality.

Conventional engineering of monolithic software systems provides clear structures and explicit processes which allow for consistent management rules, plans, and control at any step and level of design and implementation as well as runtime behavior. During the design process all possible states of the environment are anticipated and a proper behavior is defined at design time, see e.g. [14]. Due to their design process those systems typically have a deterministic input-output behavior, as long as their inputs are defined. If this is not the case, or the system is in a context that was not regarded at design time, its behavior is undefined and cannot be adapted automatically. Therefore, those systems can be typically classified as inflexible, in the sense of [6]. This definition goes in line with the aspects that a system can adapt itself to a changing environment and changing demands towards the system. This definition can be found in the self-organizing systems as well. In this sense, self-organizing systems can have the ability to be robust against environmental changes as they can adapt themselves to these changes.

A different approach to software engineering can be found in the field of autonomous systems or distributed artificial intelligence. Here, the software engineer implements functionalities for handling of unforseen environment states by run-time adaptation. This includes the ability to recognize a decision situation, identify alternatives, assess them, decide, and implement the decision. Doing so, increased robustness and positive emergence of the system as a whole should evolve. In consequence, the system should be able to cope with dynamics and complexity, distributively and flexibly. Nevertheless, we assume the local autonomy of distributed software entities may cause suboptimal system states, as the decentralized decision making creates optimal structures and processes on a micro level, but often fails to optimize the system as a whole, which is the goal of strategic consideration on the macro level.

Moreover, the flexible behavior of autonomous systems however leads to some sort of indeterministic behavior [7]. This effect can be explained by the definition of determinism. Using a simple external view like the input-output relation a multiagent system can be non-deterministic. However, this does not mean, that an autonomous system has to be non-deterministic. The appearance of non-determinism arises from the limited view on the environmental state (situation). If the internal state of the system is included, an autonomous system might also be deterministic. That means that the system can show a behavior that is not anticipated by an user and does not go in line with his expectation.

Both aspects are major obstacles for the commercial use of this technology, and therefore hinder the use of concepts and software that enable flexibility and self organizing software systems which is needed to create software that offers competitive advantages.

With a higher level of autonomy, decision making is supposed to become faster and more flexible, but it may also lead to a loss of control for the management of the system. This gap between operational decision-making and strategic management has to be bridged to ensure reliable decision-making. We are convinced that both aspects, deterministic behavior and flexibility are relevant aspects. Thus, we present an approach that allows both. Our goal is to allow as much flexibility as possible while ensuring an acceptable overall system's performance. It does not seem appropriate to construct distributed software entities which are either purely autonomous or strongly regulated. Both aspects, autonomy and regulation, represent the opposed margins of a scale. The challenge for the engineering of distributed autonomous software is to realize systems with a situation-dependent, optimal mixture of autonomy and regulation.

We structure the rest of the paper as follows. In the next section, we discuss the different aspects of autonomy that can be

identified in autonomous software systems and multiagent systems. Then in section 3 we describe our idea that transfers the concept of management-by-exception into multiagent systems. To investigate the potentials of our approach we use a case study, investigating a job shop scheduling problem. Presented results indicate that our approach can meet our expectations. Finally, we summarize our findings and draw our conclusion of the potentials of the management-by-exception approach for self-organizing systems.

2. Levels of Autonomy

In the literature, there are discussions on different levels of autonomy [4, 10]. In early multiagent research, Castelfranchi and Conte [2] discuss a very high degree of autonomy, such as the influence of predefined norms, behavior patterns, or procedures is irrelevant, and the relevance is very low with respect to the action-selection-process within an agent, respectively. The design and the runtime behavior are very difficult to design and ensure if such autonomous systems have to interact with each other.

Timm [15] introduces four levels of autonomy (LoA): *strong regulation*, *operational autonomy*, *tactical autonomy*, *strategic autonomy* following a systematic approach using the levels of decision making known from economics and system theory: operations, tactics, and strategies. Autonomous systems are situated in an environment with the capability to perceive and interact with it. The complexity of the deliberation process within an agent is strongly related to different environmental properties as proposed by Russell and Norvig [11]. This categorization is used to specify the levels of autonomy and to describe their respective adequacy in application domains.

Strongly regulated (LoA 0) systems are the class of systems usually dealt with in "traditional" software engineering. There is no part of the system with autonomous capabilities. Any decision – regardless of the decision level – is predefined or determined by an external entity. Conventional monolithic systems are examples for this class of systems. They proved to be effective in environments with limited complexity characterized by full observability, determinism and episodic structure without further autonomous systems and with a mainly static behavior.

Operational autonomy (LoA 1) Here, an autonomous software system gains the competence to decide on an operational level with a specific "slack" in the behavior. However, this decision still follows the tactical and strategic boundaries of the system. In the BDI approach [9], operational autonomy means that there is no flexibility in the desires or the intention selection mechanism, i.e., the desires and intentions cannot be modified by the agent. However, the agent is capable to reflect or refine upon plans.

Tactical autonomy (LoA 2) Tactical decision making in autonomous software systems enables the system to deliberate on different alternatives for operational behavior. While in operational autonomy, plans are under consideration, e.g., linearization of partial plans, tactical autonomy is performed at the level of goals and intentions, respectively. Considering BDI agents, the planning and execution level is associated with the operational autonomy. The deliberation step of a BDI agent, where an agent selects or creates an intention with respect to its desires and its current state is the corresponding level for tactical decision-making.

Strategic autonomy (LoA 3) The strategic aspects represent the highest level of decision making within a software system and are conventionally determined by the system's designer in advance or by external influence, e.g., the user, during runtime. The architecture of each individual agent incorporates static strategies, e.g., by the definition of individual desires and specific algorithms. In conventional BDI, strategic autonomy by the agent itself is not feasible as the desires are determined statically. Timm [15] introduces an approach for strategic autonomy as an extension to conventional BDI. Here, the agents are enabled to dynamically compute interdependencies between desires and intentions with respect to the current state of the agent.

3. Management-by-Exception as a Metaphor

A major problem of self-organizing systems is that it cannot be guaranteed that the system organizes itself in a way that it can offer sufficient performance[1]. We therefore suggest to limit at least temporally the autonomy of the different entities. To do so, we use a management principle that is called *management-by-exception*. The main idea of the management-by-exception approach is that the management defines boundaries for performance indicators that are acceptable. In this way the everyday business can be handled by employees that can act autonomously as long as the performance indicators are within the boundaries. If a critical situations occurs, an exception which is indicated by performance indicators leaving their bounds, the issue has to be escalated to the next superior management tier (see e.g. [5]).

We assume that it is possible that the systems we address here have the ability to self-monitor their performance. This can be realized e.g. by a component responsible for monitoring the system performance. Moreover, we assume that there exists an entity within the system that is allowed to give orders to the other entities, limiting their autonomy in favor for the global system performance. In the following we assume that the performance is measured as a single value, of course this can be extended towards a performance vector. The simplification is used here to ease the explanation of the procedure. If a specified threshold is missed, all entities are ordered not to optimize their local goals but to work cooperative optimizing the overall system's performance. If this exception has been handled successfully and the performance indicator is within its bounds again, the central control can be released and all entities can perform their local optimization.

We are convinced that this management approach can be applied to multiagent systems sucessfully. Therefore we investigate two aspects:

1. Can the overall system performance be kept in acceptable bounds?
2. Is the more centralized control really an exception or is it the normal case?

The first question aims to check if it is at least possible to reach acceptable overall system performance. Of course, here it is a critical aspect what is defined as acceptable. A global optimal

[1] Thereby it depends on the context of the application in what terms and measurements performance is defined, e.g. in terms of availability or throughput.

solution may be desired but due to the decentralized approach of locally optimizing entities it is only reachable when we emulate the central control, what can be done resulting in a system that is not flexible again. The second question goes into this detail to what degree a centralized control is really needed to ensure the acceptable overall system performance.

4. Case Study: Job Shop Scheduling

We have taken a scenario from the production domain where we applied our theories and investigated the questions sketched above. In the following, we first present the production scenario and then present the results of our simulation.

4.1 The Test Setting

We use here a job shop scenario originally presented in [3] and [1]. This allows us to compare our actual results with previous work. We experimented with this scenario in [13, 12], investigating different aspects of self-organizing systems and the performance measurement within those systems. Figure 1 presents only a schematic overview of the scenario. In the original paper by Cavalieri et al. [3] the processing entities were called machines. We call it shops here because each machine can be seen as a local shop and therefore represented by its own agent. This allows us more general research in further work as well. Each shop has an input and an output buffer. It offers exactly one operation. For simplicity reasons transportation is not modeled explicitly. It is assumed that always enough transport capacity is available and transportation time is zero.

The job characteristics are taken from Brennan and O [1] where the scenario is used as well. In table 1 the duration and shop sequence are summarized. There exist five different job types which differ in their processing time for each operation and the sequence of operations needed to be done. As we actually do not want to test the reactive scheduling abilities in our research it is assumed that no disturbing events occur and processing times are always met.

Duration / Operation	Step 1	Step 2	Step 3	Step 4
J1	6/1	8/2	13/3	5/4
J2	4/1	3/2	8/3	3/4
J3	3/4	6/2	15/1	4/3
J4	5/2	6/1	13/3	4/4
J5	5/1	3/2	8/4	4/3

Table 1. Process plan for different jobs, encoded as time/operation, according to Brennan and O [1]

There exists a huge number of different dispatching strategies for shops. Here, only a subset is presented and used in the simulation studies. All these strategies are well known and were taken from Pinedo [8]. They are presented in table 2. All these strategies need no or only the duration and priority information of the job. Thus, we restrict the scenario to use no generated release and due dates because we want to concentrate on the data presented in the literature. For the WSPT strategy priority information is needed. This priority is assigned to the job by the manager. In this strategy, the processing time is multiplied with the priority. The priority for a job is a uniformly distributed value in the interval [0,1]. For the job routing different strategies can be found, as well. Examples of routing strategies might be shortest queue (SQ), shortest queue according to sum of durations (SQSD) or service in random order (SIRO). In [13], it was pointed out that in the given scenario the usage of different job routing strategies do not have significant effects. We fix the strategy for jobs in this research to the SQ strategy. The strategies for the shops are assigned randomly from the set of strategies presented in table 2.

Strategy Code	Description
SIRO	Service in random order
FIFO	First in first out
SPT	Shortest processing time first
LPT	Longest processing time first
WSPT	Weighted SPT

Table 2. Dispatching strategies for shops

The performance indicator is here the mean flow time of a job, from the starting of the first operation to the end of the last operation. It has to be kept in mind that the shop floor system is not necessarily optimizing these objectives. This implies that the results may be worse than in a scenario with a purely centralized control.

Jobs and shops are represented by agents. Moreover, there exists a supervising agent which is called the manager agent. If a job is finished, the corresponding job agent informs the manager and reports the flow time of this job. The manager agent can monitor the mean flow time according to the jobs finished so far. If this value falls below a specified threshold, the manager agent can order the shop agents to work to a specific strategy. Here this is the SPT strategy which is known to perform best in this scenario. If the actual mean flow time reaches an acceptable range again, it can allow the shops to work according to their locally preferred strategy.

4.2 Evaluation

All tests were made using the above described scenario setting with initially 100 jobs (20 per job type) starting at the same time and all shops have an empty queue. All presented results are means computed by 10 runs of each experiment. Both questions presented in the previous section should be answered in the context of this case study. This case study is discussed in more detail in [12].

To answer the first question one has to define what is an acceptable quality of the system. Actually, we used two approaches. First, we assign each shop a strategy randomly (strategy: random). Second, we assign each shop the same strategy and then compute the mean value for all strategies (strategy: mean). The mean flow times are sketched in figure 2. We decide to use the computed mean values, as this leads to a lower bound for the acceptable mean flow times. The strategies CON1, CON3 and CON10 represent strategies where the shops started with randomly assigned strategies and the manager agent supervises and monitors the overall system's performance. This monitoring is done after one, three or ten jobs have been finished. Thus these strategies represent different monitoring and control strategies. But as one can see in figure 2, all

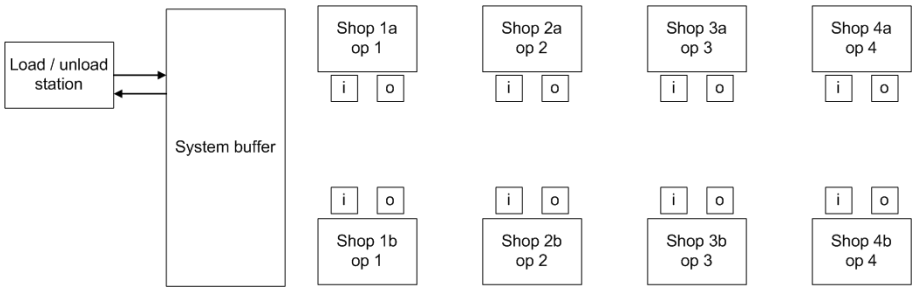

Fig. 1. Shop layout, according to [3]

regulated strategies can ensure that their results remain below the given boundary.

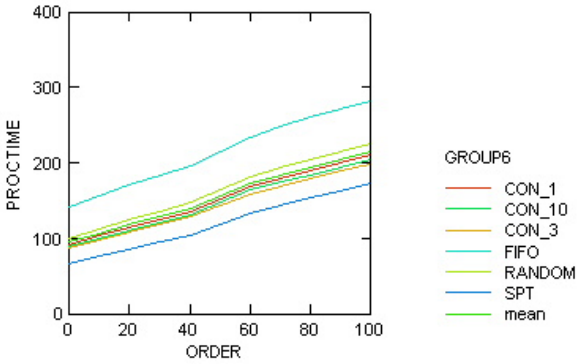

Fig. 2. Mean flow times for different strategies

To answer the second research question for this case study we investigate the fraction of time the central control was ordered. This fraction is shown in figure 3 presenting Box-Whisker plots of the relative central control time, i.e., the ratio of time interval lengths under central control divided by the total time. While the mean central control times of setting `Con01` and `Con03` do not differ a lot, the mean value of `Con10` is lower indicating that in our experiments central control is rather infrequent. Having in mind that these regulated strategies are capable to ensure an adequate level of the overall performance it can be stated, that this can be done restricting the local autonomy rarely.

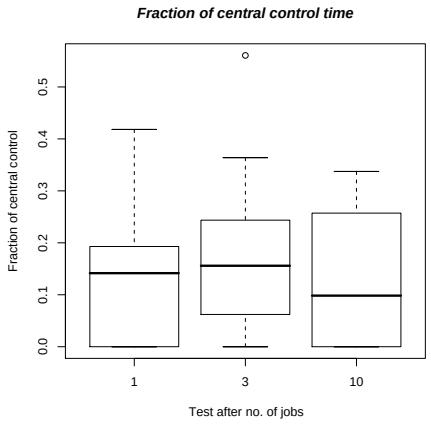

Fig. 3. Box-Whisker plot of relative central control times

5. Conclusion

In this article, the first results of implementing a modern management approach into self-organizing software systems is presented. Our first results indicate that the idea of management-by-exception is an adequate approach to design systems that on the one hand show an acceptable overall system's performance and on the other hand allow a maximal degree of local autonomy and thus flexibility of the systems. This aims to allow the emergent effects of self-organization and the flexibility of distributed autonomous systems while guaranteeing that the overall system performance remains in acceptable bounds.

As the work presented here are first results there are a number of questions open, so far. Next steps should validate the observations we made in our first case study. Therefore, a different domain should be addressed. Another important question is to investigate in more detail how the degree of central control is correlated with the defined acceptable quality. As discussed earlier if one forces the optimal solution to be computed, then this approach is equivalent to a central controlled approach. In our case study the required quality was above a random assignment and needed in most cases below 20% centralized control.

References

[1] R W Brennan and W O, A simulation test-bed to evaluate multi-agent control of manufacturing systems. WSC '00: Proceedings of the 32nd Conference on Winter Simulation Orlando, Florida, 2000, Society for Computer Simulation International, pp. 1747–1756.

[2] C Castelfranchi and R Conte, Emergent functionality among intelligent systems: Cooperation within and without minds. Journal on Artificial Intelligence and Society, Vol. 6, No. 1, 1992, pp. 78–87.

[3] S Cavalieri, L Bongaerts, M Macchi, M Taisch and J Weyns, A benchmark framework for manufacturing control. Second International Workshop on Intelligent Manufacturing Systems Leuven, Belgium, 1999, pp. 225 – 236.

[4] R Falcone and C Castelfranchi, Grounding autonomy adjustment on delegation and trust theory. Journal of Experimental and Theoretical Artificial Intelligence, Vol. 12, No. 2, 2000, pp. 149–151.

[5] W Gladen, Performance Measurement - Controlling mit Kennzahlen, 3rd ed., Gabler, Wiesbaden, 2005.

[6] S Kirn, Flexibility of multiagent systems, In Multiagent Engineering, S Kirn, O Herzog, P Lockemann, and O Spaniol, Eds. Springer, Berlin et al, 2006, pp. 53 – 69.

[7] S Kirn, O Herzog, P Lockemann and O Spaniol, Eds., Multiagent Engineering - Theory and Application in Enterprises, Springer, Berlin, 2006.

[8] M Pinedo, Scheduling: Theory, Algorithms and Systems, Industrial and Systems Engineering. Prentice-Hall, 1995.

[9] A S Rao and M P Georgeff, BDI Agents: From Theory to

Practice, Technical Node 56, Australian Artificial Intelligence Institute, 1995.
[10] M Rovatsos and G Weiss, Autonomous software. Handbook of Software Engineering and Knowledge Engineering River Edge, New Jersey, 2005, S K Chang, Ed., Vol. 3: Recent Advances, World Scientific Publishing.
[11] S Russell and P Norvig, Artificial Intelligence: A Modern Approach (Second Edition), Prentice Hall, Pearson Education, Inc., Upper Saddle River, New Jersey, 2003.
[12] R Schumann, A D Lattner and I J Timm, Regulated autonomy: A case study. Intelligente Systeme zur Entscheidungsunterstützung,Teilkonferenz der Multikonferenz Wirtschaftsinformatik Garching, Germany, 2008, L Mönch and G Pankratz, Eds., SCS Publishing House, pp. 83 –98.
[13] R Schumann and J Sauer, Implications and consequences of mass customization on manufacturing control. IMCM'07 + PETO'07 Hamburg, 2007, T Blecker, K Edwards, G Friedrich, and F Salvador, Eds., Vol. 3 of Series on Business Informatics and Application Systems, GITO, pp. 365 – 378.
[14] I Sommerville, Software Engineering, 6. edition ed., Pearson Studium, 2001.
[15] I J Timm, Strategic management of autonomous software systems, Tech. rep., Universität Bremen, 2006, TZI Report No. 35.