HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems

Jan-Philipp Steghöfer, Pascal Behrmann, Gerrit Anders, Florian Siefert, and Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, Germany
{steghoefer, anders, siefert, reif}@informatik.uni-augsburg.de, PascalBehrmann@gmx.de

***Abstract*—The formation of hierarchies within large-scale systems can solve problems of scalability and distributed control. In this paper, we suggest a self-organising partitioning control scheme that uses a distributed set partitioning algorithm to dynamically introduce and resolve hierarchy layers in a decentralised fashion according to the needs of the application at runtime. The partitioning control can work within a predefined organisational framework and is highly adaptable to application-specific needs. We demonstrate the approach with an application from the domain of distributed power management and provide evaluations that show that a self-organising hierarchy formation can increase scalability by simplifying control decisions with negligible overhead.**

***Keywords*-Autonomous agents; Hierarchical Systems; Adaptive Systems**

I. Introduction

Complex systems, consisting of thousands of individual, heterogeneous agents, are a scalability nightmare. Centralised approaches to their control and monitoring can almost never be realised. This is due to two facts: first of all, propagating the necessary data from the agents to a central instance can take a very long time; second, the decision making process has to consider all this data and therefore can have prohibiting runtime. While waiting for the data to be aggregated and for the decision to be made, the environment of the agents can change tremendously. It can change so much that the decision that is ultimately made becomes obsolete immediately.

For this reason, many systems (as outlined in Section II) have levels of indirection and abstraction, often represented by hierarchies [1]. Using hierarchies allows to compartmentalise computation to certain areas of the system and thus to provide scalability beyond a couple of dozen agents. Information aggregation does not take as long and decision making processes do not have to deal with such a high number of variables. Additionally, hierarchies can be used to represent existing organisational structures, a feature often desired when modelling existing systems with agents. Systems of systems (SoS) and holarchies are concepts that directly integrate this hierarchical nature. More often than not, however, hierarchies are defined once by the designers of the system and do not adapt to changing circumstances during the runtime of the system.

In this paper, we present *HiSPADA*, an extension of the Set Partitioning Algorithm for Distributed Agents (SPADA, [2]). HiSPADA forms hierarchical partitions that represent levels in the system structure. These partitions and the hierarchy in which they are arranged are highly flexible and can adapt to changes in the environment, to new and leaving agents, to changing system goals, etc. We show that HiSPADA indeed reduces the complexity of control decisions while the system's other quality attributes remain almost the same.

However, to make use of a hierarchical partitioning control, some caveats have to be heeded. On the one hand, for HiSPADA to work efficiently, it must be possible to introduce new layers by creating intermediary agents that encapsulate the essence of the agents they control. On the other hand, hierarchical task decomposition must be possible. Our illustrative example exhibits these properties. The case study is a power management system in which distributed energy resources (DERs) are partitioned into Autonomous Virtual Power Plants (AVPPs) [3]. These AVPPs coordinate the power plants to meet power demands and to stabilise the network frequency. The system consisting of hierarchical AVPPs and DERs can be considered a system of systems.

To meet the overall power demand, schedules are created that assign a fraction of the demand to each individual power plant by solving a constraint optimisation problem. The more power plants there are in the system, the more complex the calculation of their schedules and the longer the runtime of the scheduling algorithms. AVPPs compartmentalise this complexity by reducing the number of power plants being part of the scheduling process within each individual AVPP. To find a good partitioning of power plants into AVPPs, the time required for scheduling has to be part of the decisions in the AVPP formation process. Additionally, existing utilities and transmission grid infrastructure impose given organisational structure within which it is possible to form AVPPs. The structure changes when AVPPs can not fulfil their power demand or other obligations.

This paper is structured as follows: in Section II, we introduce SoS and holarchies, illustrate systems in which predefined hierarchies are used for the same purpose as in this paper, introduce hierarchical clustering, and outline the ideas behind dynamical hierarchies. Section III then describes the non-hierarchical self-organisation algorithm that forms the basis of HiSPADA, which itself is explained in detail in Section IV along with its prerequisites. We present an evaluation of the algorithm in Section V and conclude the paper with a discussion and outlook on future work in Section VI.

 ISBN: 978-1-61208-257-8

II. Holarchies, Hierarchical Self-Organisation, and Systems of Systems

In [1], Horling and Lesser discuss a number of organisational paradigms for multi-agent systems. Most importantly, they identify hierarchies and holarchies as the main approaches to deal with complexity and scalability issues.

Holarchies are a special kind of hierarchical organisation. The concept of a "holon", originally described by Koestler in [4], refers to a recursive, self-similar structure, while a holarchy is a hierarchy of self-organised holons. In modern uses of this concept, e.g., for holonic manufacturing systems, a holarchy is defined as a system of such holons cooperating to achieve a common goal [5]. A holarchy used in this context is usually not changed at runtime but predefined by the designer to meet the specific system requirements.

Similarly, many self-organising systems use predefined, static hierarchies to reflect existing hierarchical structures. In the Organic Traffic Control [6] project, e.g., hierarchies consist of individual traffic lights, intersections, and entire roads. On each level, the system is able to learn traffic patterns and therefore adapts to the traffic flow. The different levels allow recognition of patterns on different scales and optimise, e.g., to create green waves during rush hour.

Hierarchical clustering algorithms are, e.g., regarded in sensor networks. There, groups of sensors represented by a cluster head are formed. The head serves as a communication hub for the entire group. The number of hops (nodes a message has to go through to reach its destination) and the communication range are vital decision variables. As [7] shows, hierarchies can significantly reduce the complexity of cluster formation. This result shows that hierarchies increase scalability. However, results and algorithms from sensor networks can not be readily applied to other domains as they work under specific assumptions. Conversely, it will be difficult to adapt HiSPADA to sensor networks as it does not account for energy efficiency or limited communication.

Holarchies and hierarchies are directly related to systems of systems (SoS). SoS are composed of systems that are themselves complex systems. They are usually distributed in nature and very large [8]. A lot of work on SoS originates in a military context (see, e.g., [9]) where the interconnection of different, complex systems is a must to provide battlefield information and control of a wide array of weapon systems and sensors. Although these systems are heavily connected, they remain independent in many ways. Key characteristics of SoS thus include functional and administrative independence of sub-systems and geographic distribution [10]. Furthermore, the behaviour of SoS is often emergent and its development evolutionary. This definition also applies to the case study used here and other open, heterogeneous systems.

Hierarchies also play a crucial role in the emergence of new functionality. The artificial life community has been looking into the synthesis of dynamical hierarchies [11] that show different emergent behaviour at various scales. Another research direction are hierarchies that self-assemble without an explicit algorithm [12] such as the one presented here. While these approaches are very interesting, their usefulness for large-scale technical systems has not yet been proven.

III. Set Partitioning with SPADA

SPADA [2], the Set Partitioning Algorithm for Distributed Agents, solves the so-called set partitioning problem (SPP) in a general, decentralised manner. In the SPP, the goal is to partition a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ into $k \leq n$ pairwise disjoint subsets, i.e., partitions, that exhibit application-specific properties. For example, if the objective is to group similar or dissimilar elements together, the SPP is equivalent to clustering or anticlustering [13]. This can be achieved by complementing the SPP with an appropriate metric. In case such a metric defines how well agents can work together on a common task, the SPP is equivalent to coalition structure generation [14]. Since SPADA has been designed to solve the SPP in general, it can be applied to these specific problems as well. This distinguishes SPADA from other centralised and decentralised approaches, which are often specialised to a specific problem in a specific domain. In the following, we give a short summary of SPADA's basic functionality and characteristics. A more detailed description can be found in [2]. We use the term "reorganisation" to denote the process performed by SPADA.

All operations SPADA performs to come to a solution can be mapped onto graph operations that operate on an overlay network, which is called *acquaintances graph*. The acquaintances graph is defined by the agents participating in the SPP and acquaintances relationships between them, symbolised by directed links. To simplify graph operations that modify partitions, it is stipulated that each partition is a directed tree of marked links, which results in a directed forest for the partitioning as a whole.

The root of each such tree is the corresponding partition's leader. Each partition thus has always exactly one leader. It is responsible for optimising its partition according to application-specific criteria. Each leader therefore periodically evaluates if it is beneficial to integrate new agents into its partition or to exclude members from it. The latter can be beneficial if the partition's or an agent's properties have changed so that the partition's formation criteria no longer favour including the agent. Integrating and excluding agents is performed by modifying the acquaintances graph.

To decide termination, leaders periodically evaluate application-specific termination criteria formulated as predicates based on local knowledge. These predicates are constraints that can also be monitored at runtime and be used to trigger reorganisation. If these are met, the leader marks its partition as terminated. As long as a partition is terminated, its structure is not changed until a member is integrated into another partition. SPADA can thus make selective changes

 ISBN: 978-1-61208-257-8

(a) Flat, single layer system structure. (b) Hierarchical system structure.

Figure 1. Different system structures. The flat, single layer system structure can be created with coalition formation, clustering, or a set partitioning algorithm such as SPADA while hierarchies can be introduced when using the HiSPADA control loop in combination with one of these mechanisms.

to an existing partitioning, which is very useful in dynamic environments. It has been shown in [2] that SPADA's local decisions lead to a partitioning whose quality is within 10% of the solutions found by a centralised metaheuristic.

In many cases, it is useful that the partitions created are represented by an intermediary. In our case study, this intermediary is the AVPP. The leader of a partition instantiates a new AVPP agent after partitioning has finished. The AVPP then assumes control of all power plants in the partition.

IV. Self-Organising Hierarchies with HiSPADA

The original SPADA algorithm partitions the set of agents representing the entire system. It creates a flat hierarchy as shown in Fig. 1 (a). To achieve hierarchical self-organisation as depicted in Fig. 1 (b), HiSPADA uses the original SPADA algorithm to partition only a subset of agents – the *neighbourhood*. The introduction and resolution of layers is handled by the HiSPADA control loop, depicted in Fig. 2. This control loop constitutes a *partitioning control*. The HiSPADA control loop (cf. Section IV-B) runs on those intermediaries that can be reorganised, i.e., that represent partitions that can be reorganised.

A. Prerequisites and Interaction with SPADA

In principle, the partitioning control is independent of the concrete set partitioning algorithm used. Therefore, it would be possible to use, e.g., a particle swarm optimiser to find appropriate partitions and introduce hierarchies by using the partitioning control described here. HiSPADA's only requirements are (a) that it must be possible to limit the underlying algorithm to a certain neighbourhood and (b) that it must be possible to use application-specific formation and termination criteria to, e.g., define a minimal number of partitions to be formed. For reasons of conciseness, we will, however, focus our explanations on the control of SPADA.

The hierarchy is represented by a tree-structure formed by father-child relationships between agents. To achieve this structure, an agent is introduced that serves as the root of the tree. When the hierarchy is updated, these relationships change. SPADA has a similar but distinct notion that expresses membership in a partition. As described above, each partition formed by SPADA has a leader that is at the root of a tree of marked links. HiSPADA makes no use of these structures used by the underlying set partitioning algorithm but only of intermediaries that represent the partitions.

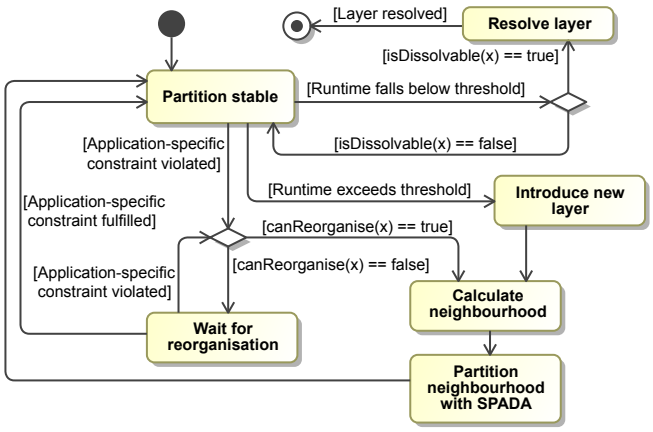


Figure 2. The HiSPADA control loop. The partitioning control running on intermediary x reacts to the violation of specific constraints, depicted here as guards, and reacts by resolving layers of the hierarchy, introducing new ones, or reorganising existing ones.

To form a hierarchy, these intermediaries are themselves partitioned. Thus, whenever an intermediary switches to another partition, the controlled agents switch with it. In the case study, a leader is always represented by an AVPP.

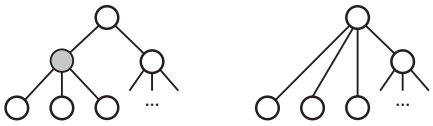
The HiSPADA control loop is usually dormant as long as the system is stable. It monitors the system however (more precisely: each instance of the control loop monitors the agent it runs on) and reacts to the violation of constraints. These constraints are depicted as guards on the transitions in Fig. 2. A run of SPADA and the introduction or resolution of layers is thus an attempt to restore these constraints. This behaviour conforms to the Restore Invariant Approach [15].

B. The HiSPADA control loop

HiSPADA has three major functional aspects: resolve an existing layer of the hierarchy by removing intermediaries; introduce a new layer in the hierarchy by creating intermediary agents; reorganise a level in the hierarchy by changing the relationships between intermediaries and the agents they control. Fig. 2 shows the control flow of HiSPADA, including these three aspects. In the following, we will use the term "hierarchy level" to denote all agents that are controlled by the same father or grandfather.

Resolve hierarchy levels: The resolution of existing hierarchy levels can occur for a number of application-specific reasons. In the power management example, it is triggered when the runtime of the scheduling algorithm falls below a given threshold. In that case, the AVPP that encountered this constraint violation is dissolved and all power plants are added to the father of the dissolved AVPP. In general, the predicate $\texttt{isDissolvable}(x)$ is checked before the actual resolution. It tests whether intermediary x can be dissolved at all.

$$\texttt{isDissolvable}(x) \Leftrightarrow x.\texttt{mayBeDissolved} \wedge$$
$$x.\texttt{timeInExistence} \geq \texttt{minTimeInExistence}$$



(a) Before resolution (b) After resolution

Figure 3. Resolution of a hierarchy level. The initiating agent is marked in grey. Children of the initiator become children of their previous grandfather.

(a) Before reorganisation. (b) After introduction of new layer.

Figure 4. Introduction of a new hierarchy level. The initiating agent is marked in grey, new agents are black. The child agents of the initiating agent form the neighbourhood, marked as a dashed rectangle, that is partitioned with SPADA. New agents are introduced to form an intermediary layer.

x.`mayBeDissolved` is false if agent x is a higher-level structure that is part of a predefined hierarchy. To avoid thrashing, it also checks if the period of grace (`minTimeInExistence`) that prevents newly formed hierarchy levels to be resolved right away has already expired. The age of the agent is stored in x.`timeInExistence`.

Fig. 3 shows a hierarchy in which a layer is resolved. After the resolution and before the initiating agent is deleted, it informs the new father agent of the changes made. The father agent then has to react appropriately by adopting its new children and by, e.g., requesting essential data from the new children or running the control algorithm again.

Introduce new hierarchy levels: In the case study, new intermediaries are introduced when an AVPP requires too much time to calculate the schedule for the power plants it controls. Other applications can of course give specific conditions under which this action is performed.

When a new hierarchy level is introduced, a father agent f creates an intermediary level for its child agents. For this purpose, f's HiSPADA control loop initialises SPADA with its child agents as the neighbourhood and a minimum number of two partitions, thus ensuring that the agents are not subsumed in just one partition. A way to guarantee this is to require that each leader knows at least one other leader. SPADA then uses this and other, application-defined criteria to create a suitable partitioning. Fig. 4 illustrates this process. The newly created agents become children of f.

Reorganising a hierarchy level: Whenever intermediary x introduces a new hierarchy level is introduced or detects violation of an application-specific constraints, it uses HiSPADA to reorganise a hierarchy level. For this purpose, it limits the scope of reorganisation to a certain neighbourhood, thus preventing it from crossing organisational boundaries. The original SPADA has no such limitation and reorganises the entire system.

HiSPADA has to consider some limitations when a hierarchy level has to be reorganised. First of all, reorganisation can not occur while an agent's father or children are being reorganised. Otherwise, it would be possible that some agents are part of several reorganisation efforts at once, possibly resulting in changes that would violate the tree-structure of the hierarchy. This limitation is captured in the $\texttt{canReorganise}(x)$ predicate that is tested before reorganisation occurs. The predicate $\texttt{father}(p, q)$ denotes that p is the direct predecessor of q in the hierarchy. If an agent p currently is reorganising or is being reorganised, $\texttt{reorganising}(p)$ evaluates to `true`.

$$\begin{aligned}\texttt{canReorganise}(x) \Leftrightarrow\ & \texttt{isDissolvable}(x)\\ & \wedge \neg\exists y \in \mathit{Agents} : \texttt{father}(x, y) \wedge \texttt{reorganising}(y)\\ & \wedge \neg\exists z \in \mathit{Agents} : \texttt{father}(z, x) \wedge \texttt{reorganising}(z)\end{aligned}$$

The second limitation is the restriction of the algorithm to neighbourhoods. In this case, the neighbourhood of the initiating agent is defined as all its children and "nephews", i.e., its siblings' children (cf. Fig. 5). Therefore, we define the neighbourhood N_x of an agent x as follows:

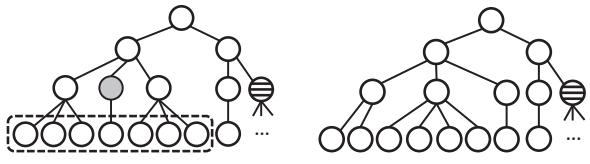
$$\begin{aligned}N_x := \{y \in \mathit{Agents} \mid \exists a, a_y \in \mathit{Agents} :\\ \texttt{father}(a, x) \wedge \texttt{father}(a, a_y) \wedge\\ \texttt{father}(a_y, y) \wedge \texttt{canReorganise}(a_y)\}\end{aligned}$$

This neighbourhood definition ensures that only agents with fathers that can be reorganised are part of the neighbourhood, thus ensuring that predefined hierarchies or levels that are still in their period of grace are not changed. In theory, neighbourhood definitions that include more distant relatives are possible and can be implemented easily in HiSPADA. However, recursing the hierarchy up too far reduces the benefit as including more agents in the neighbourhood makes the partitioning problem more complicated and agents that are only distantly related to each other have usually been separated by HiSPADA in the course of hierarchy creation. Partitioning them closer together is not useful and might even jeopardise the system goal. Also, this approach ensures that HiSPADA tries to provide a local solution with only a limited amount of communication.

After the neighbourhood has been determined, SPADA is initialised with the set of agents in N_x and a minimal number of new partitions. Then, the partitioning algorithm is executed the same way as in the non-hierarchical case. After the algorithm has been run, existing intermediaries are reused or – depending on the number of created partitions – new ones are created or old ones removed. Fig. 5 shows an example for the reorganisation of a hierarchy level.

Bootstrapping the system: HiSPADA assumes very little about the initial conditions of the system. As the partitioning control runs on the agents of the system, a hierarchy will develop in the system if the constraints monitored by the control loop are violated. If there is no hierarchical structure to begin with, an initial run of SPADA

 ISBN: 978-1-61208-257-8



(a) Before reorganisation. (b) After reorganisation.

Figure 5. Reorganisation of a hierarchy level. The initiating agent (corresponding to intermediary x) is marked in grey and the neighbourhood is marked by a dashed rectangle. The dashed agent can not be reconfigured.

can establish a partitioning on all agents in the system. This ensures that the initial partitions are suitable for the purposes of the system as SPADA uses application-specific metrics in the process. The (flat) hierarchy introduced by SPADA can easily be transformed into a tree by establishing a root agent that subsumes the intermediaries representing the newly formed partitions. This root is not dissolvable and stays at the root of the hierarchy throughout the runtime of the system. Instead of running SPADA, the system can also be partitioned randomly. In such a case, however, HiSPADA will take a while to find a suitable hierarchy if the initial partitioning did not make use of application-specific criteria.

If an organisational structure exists, corresponding partitions have to be initialised. These agents have appropriate relationships with their children to depict the organisation. HiSPADA can then work on this hierarchy by introducing intermediaries and resolving hierarchy layers formed by those intermediaries. It is therefore possible to let the partitioning control find a suitable sub-hierarchy for each of the predefined organisational enitities. Of course, there is a trade-off to be made between the fine-grained depiction of existing structures and the organisational prowess of HiSPADA: if the predefined structure is too rigid, the partitioning control is not able to tackle the scalability issues it is intended for.

V. Evaluation

HiSPADA has been tested in a simulation environment for power management applications, simulating 435 power plants and 12 consumers. In the application, a constraint optimisation problem is solved to assign the overall power demand to individual power plants while minimising the gap between the power demand and the scheduled power production. The scheduling problem is NP-complete and the runtime of the solution algorithm increases polynomially with the number of agents involved. Power plant models and power demand are based on real-world data. Such an application is a typical operational scenario for HiSPADA, since the main task is computationally much more expensive than hierarchy formation and can be hierarchically decomposed.

If no hierarchies exist (Scenario A), schedules are created by a centralised control system for all 435 power plants. If the original SPADA is used to establish a flat hierarchy in the system (Scenario B), the power plants' schedules are created by the respective AVPPs. The schedules of the AVPPs are again created by a root AVPP, as described in Section IV. In case HiSPADA can work without predefined organisations (Scenario C) and within a predefined hierarchy with an initial height of 3 (Scenario D), schedules are created in each AVPP on different levels of the hierarchy. We define the height of the hierarchy as the length of the longest downward path to a leaf from the root.

TABLE I
Evaluation results for scalability and system stability

	Sc. A	Sc. B	Sc. C	Sc. D
Max. sequential runtime of scheduling in ms	3624 $\pm$ 55	925 $\pm$ 638	499 $\pm$ 220	484 $\pm$ 213
Avg. height of hierarchy	–	2	4.99	3.52

The evaluation's focus is on performance indicators for scalable operation. The most important criterion is the average runtime of the scheduling algorithm: if the partitioning control works, the overall runtime should be reduced when hierarchies are introduced while maintaining stable system operation. The latter can be measured by comparing the gap between power production and demand as calculated by a central authority with the sum of the gaps calculated on the different levels of the hierarchy. As the deviation between these numbers was minimal and the quality of the scheduling thus was not impaired, we did not include it in Table I which lists the results for runtime and height of the hierarchy.

All results were averaged over 100 simulation runs for each scenario. The maximum sequential runtime provided in the table is the maximum of the sums of the runtime in each branch originating from the root. After the root node has calculated a schedule for its direct children, these in turn schedule their children. The scheduling is recursively performed on lower levels until the schedules for the physical power plants are created. As each branch performs the scheduling concurrently, no overhead is introduced.

The evaluation results in Table I show that the average maximum sequential scheduling time is reduced significantly with the introduction of hierarchies. In Scenario B, the high standard deviation can be explained by the variation of partition sizes created by SPADA in different runs. HiSPADA ensures more homogeneous partitions and thus less variation in the scheduling times. At the same time, the average height of the hierarchy has a direct effect on the interaction between SPADA and the partitioning control. The deeper the hierarchy, the smaller the neighbourhoods on which SPADA is executed but the more scheduling runs are performed, making it necessary to find a proper balance. Since SPADA scales very well for small sets (see Fig. 6) and HiSPADA reduces set sizes, hierarchies additionally reduce the overhead for restructuring the system. Very large systems will profit most from these reductions. The results for Scenario D also show that HiSPADA can work effectively with predefined organisations. If necessary, HiSPADA adds additional layers to further decompose the computation task and thus reduces scheduling runtime.

 ISBN: 978-1-61208-257-8

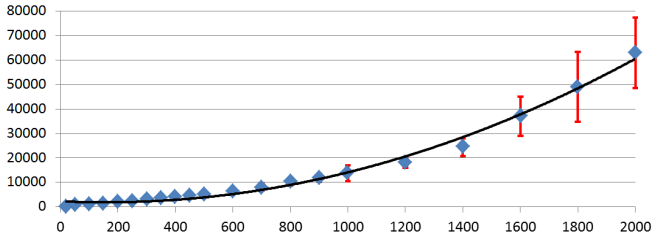


Figure 6. Runtime of SPADA for different neighbourhood sizes. The algorithm scales nearly linearly for sets below 1000 agents. Hierarchies limit the neighbourhood size and allow to leverage this property.

VI. Discussion and Future Work

We have introduced HiSPADA, a hierarchical partitioning control that – in combination with a set partitioning algorithm such as SPADA – forms hierarchical layers in large multi-agent systems. The formation of hierarchies is driven by scalability metrics provided by the application. We gave an example limiting the duration of scheduling in a power management scenario. HiSPADA introduces layers to limit the number of agents controlled by an intermediary and removes layers if they are no longer necessary. As it only removes layers that have been introduced by the partitioning control itself, it respects representations of existing organisational structures. The evaluation shows that scalability issues can be tackled this way and that the benefits of the hierarchy formation outweigh the overhead of the partitioning control.

In systems in which a hierarchical task decomposition is not possible, HiSPADA can be useful to establish a hierarchy in which control can be delegated to sub-ordinate levels or to subsume certain responsibilities on a higher level of the hierarchy. In such cases, normative systems are helpful to regulate the delegation of power and responsibilities at the different levels [16], a topic considered for future work.

Finally, an important issue in heterogeneous multi-agent systems that use hierarchical task decomposition are the models used in the decision making process. An hierarchical scheduling mechanism like the one used in the example requires models of the power plants that are scheduled. Since power management systems are long-lived and consist of a variety of different power plants of different types and from different vendors, the models used in the process are only known at runtime. Therefore, model synthesis is necessary to form a new, combined model from the individual parts on each hierarchical layer whenever the structure of that layer changes. Likewise, as each layer in the system is in turn regarded as a power plant, the combined model has to be abstracted so it can be used on a higher layer to make control decisions. Both the processes of model synthesis and abstraction are important topics of future work.

Acknowledgment

This research is partly sponsored by the German Research Foundation (DFG) in the project “OC-Trust” (FOR 1085).

References

[1] B. Horling and V. Lesser, “A survey of multi-agent organizational paradigms,” The Knowledge Engineering Review, vol. 19, no. 04, pp. 281–316, 2004.

[2] G. Anders, F. Siefert, J.-P. Steghöfer, and W. Reif, “A decentralized multi-agent algorithm for the set partitioning problem,” in PRIMA 2012: Principles and Practice of Multi-Agent Systems, ser. Lecture Notes in Computer Science, I. Rahwan, W. Wobcke, S. Sen, and T. Sugawara, Eds. Springer Berlin / Heidelberg, 2012, vol. 7455, pp. 107–121.

[3] G. Anders, F. Siefert, J.-P. Steghöfer, H. Seebach, F. Nafz, and W. Reif, “Structuring and Controlling Distributed Power Sources by Autonomous Virtual Power Plants,” in Proc. of the Power & Energy Student Summit 2010, October 2010, pp. 40–42.

[4] A. Koestler, The ghost in the machine. London: Hutchinson, 1967.

[5] A. Colombo, R. Schoop, and R. Neubert, “An agent-based intelligent control platform for industrial holonic manufacturing systems,” Industrial Electronics, IEEE Transactions on, vol. 53, no. 1, Feb. 2005, pp. 322–337.

[6] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, “Organic traffic control,” in Organic Computing – A Paradigm Shift for Complex Systems, ser. Autonomic Systems, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer Basel, 2011, vol. 1, pp. 431–446.

[7] S. Bandyopadhyay and E. Coyle, “An energy efficient hierarchical clustering algorithm for wireless sensor networks,” in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, 2003, pp. 1713–1723.

[8] V. Kotov, “Systems of systems as communicating structures,” in Object-oriented technology and computing systems re-engineering, H. Zedan and A. Cau, Eds. Chichester, USA: Horwood Publishing, Ltd., 1999, pp. 141–154.

[9] W. H. Manthorpe, “The emerging joint system of systems: A systems engineering challenge and opportunity for APL,” Johns Hopkins APL Technical Digest, vol. 17, no. 3, 1996, p. 305.

[10] A. P. Sage and C. D. Cuppan, “On the systems engineering and management of systems of systems and federations of systems,”Information, Knowledge, Systems Management, vol. 2, no. 4, 2001, pp. 325–345.

[11] T. Lenaerts, D. Chu, and R. Watson, “Dynamical hierarchies,” Artificial Life, vol. 11, no. 4, 2005, pp. 403–405.

[12] A. Dorin and J. McCormack, “Self-assembling dynamical hierarchies,” in *Proceedings of the 8th International Conference on Artificial life*, ser. ICAL 2003. Cambridge, MA, USA: MIT Press, 2003, pp. 423–428.

[13] V. Valev, “Set partition principles revisited,” in Advances in Pattern Recognition, ser. LNCS. Springer, 1998, vol. 1451, pp. 875–881.

[14] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci, “An Anytime Algorithm for Optimal Coalition Structure Generation,” Journal of Artificial Intelligence Research, vol. 34, 2009, pp. 521–567.

[15] F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, “Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach,” in Organic Computing – A Paradigm Shift for Complex Systems, ser. Autonomic Systems. Springer Basel, 2011, vol. 1, pp. 79–93.

[16] A. Artikis, M. Sergot, and J. Pitt, “Specifying norm-governed computational societies,” ACM Transactions on Computational Logic (TOCL), vol. 10, no. 1, 2009, pp. 1–42.

 ISBN: 978-1-61208-257-8