Design Patterns for Human-Cognitive Agent Teaming

Axel Schulte[1(✉)], Diana Donath[1], and Douglas S. Lange[2]

[1] Universität der Bundeswehr München, Neubiberg, Germany
{axel.schulte,diana.donath}@unibw.de
[2] Space and Naval Warfare Systems Center Pacific, San Diego, CA, USA
doug.lange@navy.mil

Abstract. The aim of this article is to provide a common, easy to use nomenclature to describe highly automated human-machine systems in the realm of vehicle guidance and foster the identification of established design patterns for human-autonomy teaming. With this effort, we intend to facilitate the discussion and exchange of approaches to the integration of humans with cognitive agents amongst researchers and system designers. By use of this nomenclature, we identify most important top-level design patterns, such as delegation and associate systems, as well as hybrid structures of humans working with cognitive agents.

Keywords: Assistant system · Autonomous system · Cognitive agent · Cooperative control · Delegation · Design patterns · Teaming · Supervisory control · Systems engineering · Unmanned vehicles · Vehicle guidance · Work system

1 Introduction

Today, higher cognitive functions (e.g., perception, planning, and decision-making) that are traditionally exclusively owned by the human, are becoming an integral part of automated functions. In the last one or two decades the term "autonomous system" has widely been used to describe complex automated systems working largely independent from a human operator. However, the more capable the automation has become, the more essential the challenging issue of human-system functional allocation and integration has turned out to be [1]. We share the concern of Bradshaw et al. [2] that an undifferentiated use of the term of "autonomy" and the proliferation of automation can lead to unfruitful discussions and oddly defined development programs. We see the need for a conceptual framework unifying the nomenclature and description of systems in which human beings interact with complex automation. Therefore, in this article, we attempt to identify and formally describe common grounds among researchers in this field. Despite our concerns, we want to adhere to the term of "Human-Autonomy Teaming (HAT)" to describe systems in which humans work with highly automated agents. Where those agents carry attributes like "autonomous" or "intelligent", we will assign the unified term ***cognitive agent*** in this nomenclature. We establish a procedure and a common language to describe concepts of HAT. Our goal is to contribute to a

D. Harris (Ed.): EPCE 2016, LNAI 9736, pp. 231–243, 2016.
DOI: 10.1007/978-3-319-40030-3_24

more objective debate, to facilitate the effective communication between researchers, and to provide guidance to practitioners. Our approach, in general, is twofold. Firstly, we suggest a common symbolic language, as well as a procedure to follow to describe systems, system requirements, and top-level system designs. Both have a stronger focus on human-automation work share and integration aspects than traditional systems engineering practices and tools (e.g., Unified Modelling Language, UML). We borrow the notion of design patterns from the domain of systems and software engineering and adapt it for use in human factors engineering of highly automated dynamic systems. Secondly, we encourage the analysis of current HAT research and development approaches in order to identify solutions and best practices from empirical studies. This article shall also provide advice for designers of HAT systems how to approach the design process in a strictly top-down manner.

1.1 Design Patterns in Engineering

Christopher Alexander proposed that every building and town is composed of patterns [3]. The patterns are a result of forces and processes that combine such that towns or buildings develop in particular ways. By developing a language of these patterns, Alexander et al. were able to describe the forces that produced patterns as well as the consequences of those patterns [4]. Pattern descriptions also specified their relationships to other patterns so that one could create a network of patterns to describe a project. Finally, Alexander et al. set out processes by which the patterns could be used. They envisioned the descriptions of forces and consequences as useful in making arguments to decision making bodies. They also described how one could specify a project from top-down using patterns to make decisions about the ultimate design.

Design patterns and pattern languages became a popular tool for software engineering in the 1990s. This is usually traced to Gamma et al. in 1995 [5]. That patterns describe a repeated problem and the core of a solution to that problem class is their fundamental value. Gamma et al. tried to be more explicit than Alexander concerning the components of a pattern. Four critical elements were listed: the pattern name, the description of the problem, the description of the solution, and the consequences. A more detailed template was created, and their book [5], like the second volume of Alexander's work [4] provided a catalog of some discovered patterns, each entry describing a suitable problem space, the solution template, positive and negative consequences, and providing implementation advice. The popularity of software patterns led to further efforts to catalog patterns for software analysis [6] and design [7], and data models [7]. Discussions of negative patterns often found in systems or organizations were described as anti-patterns [8] with discussions of how to correct the problems. Conferences were held to capture the experience of practitioners in program design as a counterweight to scientific activities that focused on new approaches [9].

Ultimately, a pattern literature for HAT will accomplish the same goals that Alexander initially set out in the domain of architecture and land-use. Patterns serve to communicate generalized solutions to problems faced by engineers. Alexander believed that one could look through a catalog of patterns, identify the key features and forces of a project, and select the appropriate starting points. Then using the linkages

provided among the patterns in his catalog one could bring in other appropriate patterns until one had a description of the solution in the form of a language of patterns. Our forces will include human performance issues, limitations of autonomy, communication issues, and many other critical factors. If our patterns can describe forces, features, consequences, and linkages to other patterns, pattern languages as solution descriptions may be possible within this domain.

1.2 Design Patterns in Human Factors, Ergonomics, and HAT

In the field of human factors and ergonomics, the description of design patterns also became fashionable recently. Borchers [10] is one of the first who described the linkage between human-computer interaction and design patterns. Kruschitz and Hitz [11] also provide a good overview. Kahn et al. [12] looked at design patterns for sociality in human-robot interaction.

Sheridan's well-known Levels-of-Automation (LoA) scale (e.g. [13]) is one of the early design patterns in human-automation interaction. System designers very successfully use this scale or one of its derivatives (e.g. [14, 15]) in many different application fields, sometimes even without knowing. In that sense, we see these kinds of Management-by-Consent/Management-by-Exception-based LoA-scales as a collection of often-found design patterns. They apply for supervisory control relationships [13] between human and machine. In this use case, LoA-scales provide an excellent source for deciding how to design the interaction for certain specific functions. Scales of levels-of-autonomy (e.g. [16, 17]) refer more to design options for the scope of a full system. Their focus is predominantly on the description of the independence of the system from human intervention.

Juziuk [18] provides a comprehensive listing and overview of efforts to document design patterns in multi-agent systems. This gets us closer to what we need for HAT, in that cognitive agents and their relationships to each other are considered.

In the following chapters, we want to create a generalized framework and methodology for describing a wider variety of configurations in different scopes. We will present an approach to derive system requirements and to describe top-level system designs for systems involving HAT based on design patterns.

2 Basic Concepts

The traditional systems engineering view is solely on the formulation of requirements and the design of the technical functions of a system. The human operator only appears as an actor, usually located outside the system boundary. This approach is reasonable when automation is relatively simple, in the sense that it can perform specific clear-cut part-tasks. There, one can well describe the relationship between the (technical) system and the human user through use cases calling for a certain user-system interaction.

In this article, in contrast, we want to take account for the following trends: (1) the automation in HAT will become much more capable, (2) the work share and interaction between the user and the system will be much less stable (e.g. adaptive automation [19]),

and (3) the task performance of human and automation will be highly dependent on a cognitive level. Hollnagel and Woods [20] speak of joint cognitive systems in this context. Consequently, our approach focuses on two aspects, (a) the description of the purpose we want to design a HAT system for before the actual design, and (b) the incorporation of the human user within the process and system boundary.

2.1 The Work Process

The process of meaningful, goal-oriented co-action of humans (e.g., operators) and machines (e.g., unmanned vehicles (UVs) with automation), including artificial cognitive agents, shall be called a Work Process (`WProc`) (see Fig. 1a). A Work Objective (`WObj`), i.e., the mission or the purpose of work, defines and initiates the `WProc`. The `WObj` usually comes as an instruction, order, or command (e.g., a UV mission assignment). The proper definition of the `WObj` is of high priority and most critical for the definition of the system boundaries and the design. The `WProc` is embedded into a Work Environment (`WEnv`). `WEnv` inputs to the `WProc` are the physical Environment (`Env`) (e.g., atmosphere, threats), material and energy Supplies (`Sup`) (e.g., fuel, weapons), and Information (`Inf`) (e.g., ATC clearances, airspace regulations). Finally, the `WProc` generates certain physical or conceptual effects to the environment, i.e. the Work Process Output (`WPOut`) (e.g., target photo/video, destruction of target, provision of information to other `WProc`).

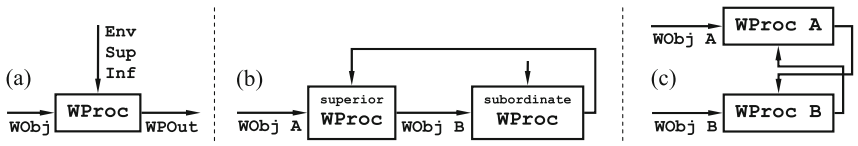

Fig. 1. (a) Work process; (b) hierarchical work processes; (c) networked work processes

The `WProc` itself imposes meaningful actions upon a particular Work Object (`WO`) (e.g., target to be destroyed, materials to be transported, premises to be secured) being part of the `WEnv`. The `WEnv` may also host other `WProcs`. In case one `WProc` interacts with other `WProcs`, this can be organized as a hierarchical structure, i.e. a superior `WProc` generates the `WObj` for one or many sub-ordinate `WProcs` and monitors their results, thereby forming supervisory loops (see Fig. 1b). Alternatively, the `WProcs` might be organized as a networked structure, i.e., parallel `WProcs` depend on each other in a way that their `WPOuts` cause environmental changes relevant to other `WProcs`, or provide supplies or information to other `WProcs`, thereby forming a more or less tight mesh of interdependent `WProcs`, each following individual `WObjs` (see Fig. 1c).

A proper work process design should be the starting point for each development of a system involving HAT. At this stage, *"it is more important to understand what the [… system] does […], than to explain how it does it"* [20]. However, defining the

WObj, the system boundary, and the interfaces of the WProc you want to design for is a hard task to do. The result will heavily influence the system design. From our experience in engineering HAT systems, we suggest a list of guidelines as in Table 1.

Table 1. Guidelines for Work Process design

1. Identify the ***Work Objective*** (**WObj**) you want to design a system for.
   - Therefore, describe the ***purpose*** your customer wants to achieve.
   - Since we want to design a ***human-machine system***, it is always a good idea to start with describing the ***job of that human*** (team).
   - If the human team members are working with very different or locally distributed workstations, ***consider*** defining ***multiple*** separate **WObjs** for hierarchical or networked **WProcs**.
2. Identify other relevant ***Work Processes*** (**WProcs**) your **WProc** is networked with.
   - Therefore, consider ***agencies actively providing orders*** or commands to your **WProc**.
   - Consider ***agencies actively providing*** relevant ***information*** or supplies to your **WProc**.
   - Furthermore, consider agencies that ***receive*** the **WPOut** you are providing.
3. Draw a network of all relevant ***Work Processes*** (**WProcs**) including your **WProc**.
   - Clearly denote the ***individual*** **WProcs**.
   - Clearly mark the most important ***information flow*** through the network. Properly distinguish between **WObjs**, **WEnv** inputs, and **WPOuts**.
   - This exercise will provide a good starting point for defining the inputs and outputs of ***your*** **WProc** (i.e., the **WProc** you want to design a system for) ***as a black box***. In this context, it might be helpful also to consider the introduction of a Work Object (**WO**).

Figure 2 shows an example of a common WProc design taken from civil aviation. The example consists of three individual WProcs, including WProc: Airline Flight, which is the process we want to design a system for. This process changes the WEnv by transporting passengers (WO: PAX). It is in a hierarchical subordinate relationship to WProc: Airline Dispatching that provides WObj: Flight and supplementary information, and to which flight and aircraft status information is fed back. With WProc: Air Traffic Control, a network is established, in which radar surveillance takes place, and requests and clearances are exchanged.

Each WProc has a certain life cycle and can be broken down into a potentially large number of subsequent and/or concurrent sub-processes. During the life cycle of a WProc, it may be exposed to many use cases. For system design, it is important to collect and describe these use cases and sub-processes that finally result in tasks to be performed either by a human, a cognitive agent, or by conventional automation. Without going deeply into well-established methods of systems engineering and cognitive task analysis (e.g. [21]) in this article, Table 2 provides some guidelines from our experience in the consequential top-down design of HAT systems.

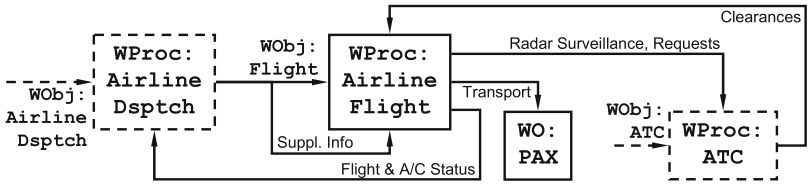

Fig. 2. Example for a common Work Process design with hierarchical and networked structures

Table 2. Guidelines for Work Process Use Case Analysis

1. Identify the relevant ***Use Cases*** you want to design a system for.
   - Therefore, analyze all ***relevant loops*** and ***meshes***, in which your `WProc` is involved and interacts with other `WProcs`, the `WEnv`, and the `WO`.
   - Identify all ***possible specifications*** of the `WObj`, the `Env`, the `Sup`, the `Inf`, and the `WPout`.
   - Make sure not to drift into ***design discussions*** on how your `WProc` shall be implemented now.
2. ***Collect all inputs and outputs*** of your `WProc` from the various use cases.
   - This will be rather straightforward, given the use cases have been well described.
   - However, this process may cause you to ***rethink*** some of the use cases.
   - As a result, you will get a good ***user requirements specification*** for the system you want to design.
3. Describe the ***life cycle phases*** your `WProc` is supposed to go through.
   - Describe the ***dynamics*** of your `WProc` by the most important or frequent use cases as ***subsequent, parallel, and/or nested sub-processes*** (e.g., “take off”, “transit”, “operation”).
   - ***Determine*** necessary subsequent and/or parallel ***tasks*** to be performed (e.g., “hold flight altitude”, “shoot photo”, “get clearance”) for each of the found sub-processes. Caution, do not pre-determine any human-automation functional allocation or any design yet!
   - Again, this can provide lengthy debates. ***Postpone*** those ***discussions*** for when you have more time and money to spend.

2.2 The Work System

Now we are ready to open the black box. From now on, we look at the physical system that runs the `WProc` described so far (i.e., the system we want to design). We will name this a Work System (`WSys`), which is our first ***design pattern***. It is important to note that the `WSys` inherits the complete definition of the corresponding `WProc` we described before. Within the box, in principle, there are two essential roles to be taken to run the `WProc`: the `Worker` and the `Tools`. Consequently, the `WSys` is composed of two components, each taking one of the roles (see Fig. 3).

The main characteristic of a `Worker` is to ***know, understand, and pursue*** the `WObj` ***by own initiative***. Without this initiative, the `WProc` would not be carried out. Therefore, a `WSys` cannot exist without a ***human*** `Worker`, by definition. Otherwise, we would not speak of a `WSys`, but rather of a mere technical artifact, i.e. a `Tool`. Only `Tools` would not make a `WSys`, nor perform a `WProc`, due to the lack of purpose. The `Worker` is the only instance and responsible for breaking down the

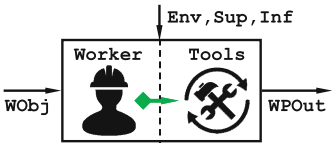

Fig. 3. Design pattern WSys as physical instance of the corresponding WProc comprising the roles of the Worker and the Tools being in a Hierarchical Relationship (HiR, green arrow) (Color figure online)

WObj into relevant tasks. The Tools, on the other hand, will receive tasks from the Worker and will only perform them when told to do so. Hence, the Worker and the Tools are always in a Hierarchical Relationship (HiR, green arrow in Fig. 3) that may be characterized by more detailed design patterns.

We would like to mention that in an earlier article [22], we defined "autonomy" by use of the WSys as the authorization of the Worker to self-define the WObj. Only the ***human*** Worker shall exercise this authority, for ethical and other reasons.

Table 3. Guidelines for initial Work System Design

1. Transition from **WProc** view to **WSys** view.
   - The starting point is the **WProc** or network of **WProcs** you designed according to Table 1. ***Each*** **WProc** ***will become a*** **WSys** keeping exactly the same specifications and periphery.
   - ***Open*** only the ***black box of your*** **WSys**, i.e. the **WSys** you want to design.
   - Keep all other **WProcs** (and also **WOs**, if needed) as black boxes. If you feel the urge to also open one of them, consider a ***re-design*** of the network of **WProcs**.
2. Initial **WSys** design.
   - Populate the **WSys** with the necessary physical entities (e.g., humans, vehicles, control stations).
   - Start with the ***human*** **Worker** and the ***conventional*** **Tools** (i.e., machines and automation not necessarily carrying attributes like "intelligent" or "autonomous") to develop a first product vision.
   - Take advantage of any available ***market solutions***.
3. Human-Agent/Automation Task Allocation.
   - Now ***allocate the tasks*** you found in your task analysis to the human **Worker** and/or automated functions. Some, unfortunately only very few, allocations are quite obvious due to their physical (e.g., aircraft flies) or legal/ethical (e.g., human decides on weapon deployment) requirements. However, many others are not!
   - If a task is undoable (by humans or system/automation), consider further breakdown of that task into sub-tasks.
   - The cognitive tasks, which turn out to be allocated to humans and automation (by sharing or trading) are the most interesting ones. They might require a ***cognitive agent*** (cf. chapter 3).

Traditionally, only a human or a human team represents the Worker. Machinery and automation would constitute the Tools. Thereby, a conventional human-machine system is created, involving manual control, and in presence of automation, also human

supervisory control [13]. However, the notion of the WSys provides, additional information concerning the WObj, i.e., the purpose of work, and the system boundary, including the definition of the interfaces to the environment. Finally, the Tools shall never contain a full WSys, or humans. From our experience, nested WSys are not an option. As an alternative, we recommend modeling the structure as a hierarchy of individual WProcs. Table 3 provides some guidelines from our experience for an initial WSys design.

3 Introduction of the Cognitive Agent into the Work System

With the advent of more advanced methods to provide higher cognitive capabilities on behalf of automated functions, the introduction of Cognitive Agents (CogA, little 'R2D2' in Fig. 4) into the WSys becomes an option. In the past, the focus in this field was predominantly on the provision of suitable information processing methods and algorithms (e.g., artificial intelligence, computer vision, soft computing; cf. e.g. [23]). Two trends have been followed in the past two decades concerning the role such an agent could potentially take in system design. Firstly, so-called autonomous systems, i.e. systems that aimed at performing user-given tasks, as much as possible independent from human intervention; and secondly, decision support, assistant, or associate systems, acknowledging that a human predominantly performs the work, while supported by a machine agent [24].

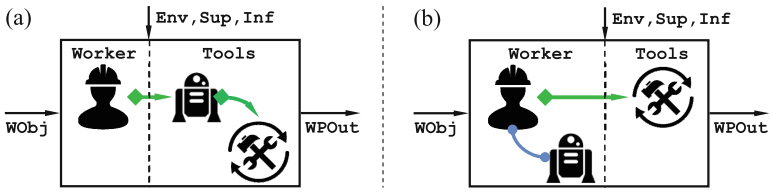

Fig. 4. (a) Design pattern WSys with CogA as Tool in HiR; (b) Design pattern WSys with CogA as Worker in HeR (Color figure online)

We want to acknowledge these trends by introducing two new elementary design patterns. Figure 4(a) shows a design pattern, where there exists a HiR (green arrow) between the Worker and the CogA being part of the Tools. Within the Tools, a HiR (green arrow) between the CogA and other automated Tools exist. Figure 4(b) shows a design pattern, where in addition to the HiR between the human Worker and the Tools (cf. Fig. 3), there exists a Heterarchical Relationship (HeR, blue connector) between the human Worker and the CogA being part of the Worker in this case. Concerning this HeR, we would like to introduce one restriction, i.e. the CogA shall not be given the authority to define or even question the WObj. The human Worker shall always have the final authority do decide.

Figure 5 shows some examples of existing setups we constructed using the elements human Worker, Tools, CogA, HiR, and HeR. Figure 5(a) depicts a regular

non-HAT system, in which two human operators cooperate while using technical equipment (e.g., a two-pilot flight-deck crew operating an aircraft). Figure 5(b) might be useful to either reduce the crew size (e.g., single pilot operations [25]), or to increase the span of control (e.g., larger "autonomy" of UVs [16]; single agent operation of multiple UVs [26]). In this case, the CogA is delegated certain tasks which otherwise a human crewmember would execute. In both examples, the effect could be mostly attributed to a reduction of the taskload of the human Worker. Elements of adaptive functional allocation might also be involved [27]. Figure 5(c) is an extension to (b) where there is more than one agent tasked by a human operator, each controlling its individual system (e.g., task-based guidance of multiple UVs [28]). Again, the challenge here is the increase of the span of control by means of spreading the taskload. Figure 5(d) goes even further down that road. Here, the human user controls a cooperating team of multiple agents, each of which operating its own equipment (e.g., pilot controlling a cooperating team of multiple UVs [29]; multi-agent system controlling multiple UVs [30]).

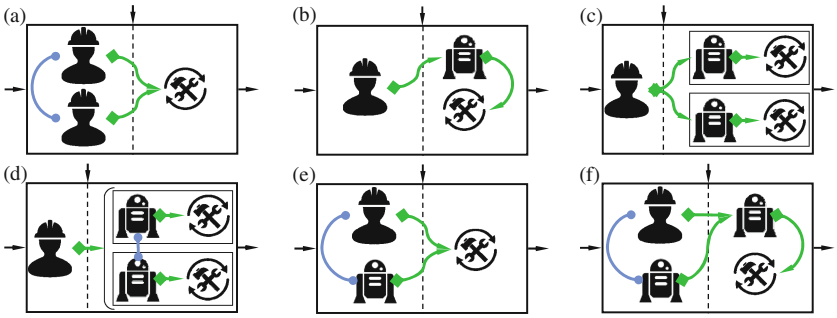

Fig. 5. Examples for WSys setups constructed from the elements human Worker, Tools, CogA, HiR, and HeR

However, an increase of automation complexity and span of control, as exercised in (b)–(d), may also result in automation-induced shortcomings. These effects have been reported by many researchers (e.g., [31] for a classical source). To counteract suchlike problems, a number of scientists suggested approaches as in Fig. 5(e). Here, the agent works in cooperation with the human operator being an element of the Worker (e.g., pilot assistance or associate systems [32]). Core elements here are the agent's initiative in achieving the WObj, and the ability of the agent to decode the human's mental states as a basis for cooperation. Means of assistance can be attention allocation, mixed-initiative operation, and adaptive automation techniques [19]. Finally, Fig. 5(f) shows a setup, in which a human is controlling a system via agent delegation, while at the same time being assisted by an agent being part of the Worker (e.g., assisted guidance of a single UV operator [33]; assisted multi-UV mission management [34]). In the two latter cases (e) and (f), where a CogA is part of the Worker, the CogA inherits the required attributes of the role of a Worker as claimed in Sect. 2.2, i.e., to know, understand and pursue the WObj by its own initiative [22, 24].

At this stage, it is obvious that there can be constructed many more configurations, especially when we look at distributed multi-user, multi-agent systems with complex HiR/HeR structures.

4 Actor-Relationship-Actor Tuples

As became clear during the discussion of Fig. 5 in the previous chapter, there are possible very many WSys configurations only by combining one or few of the symbols provided (i.e., human Worker, CogA, Tools, HiR, and HeR). Thereby, researchers and practitioners can depict their individual solutions described in literature, and hence, make them comparable. In this chapter, we now want to look at the possible {Actor-Relationship-Actor}-tuples, which can occur in all possible WSys. Figure 6 gives an annotated overview of all possible tuples.

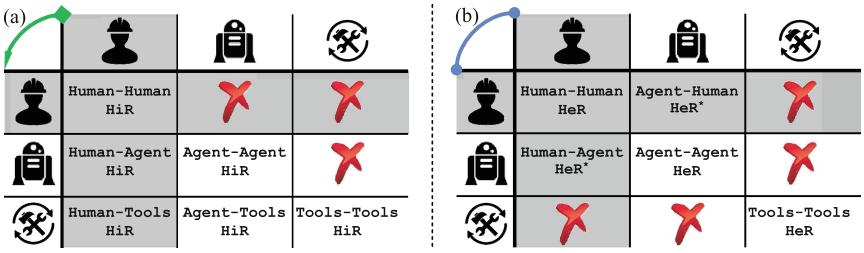

Fig. 6. (a) Hierarchical, (b) heterarchical{Actor-Relationship-Actor}-tuples; (shaded: human involved; *: equal configurations; cross: invalid option)

A hierarchy of an agent or a tool over a human, or a tool over an agent we do not want further to consider. The same applies to a heterarchy of a tool with either a human or an agent, for obvious reasons. Tuples, which do not involve humans, may not directly be interesting for HAT systems. However, they certainly can influence the behavior of the automation "under the hood", and therefore, be worthwhile to look at, at least from a pure engineering stance. Also, the pure human-human relationships, either hierarchical or heterarchical, may not directly be relevant for HAT systems, except of course for WSys with more than one human. Apart from that, they may serve as valuable source for design metaphors. Finally, we do not want to allow a Human-Agent HeR, where the agent is part of the Tools, since per definition there is always a HiR between Worker and Tools.

Within the scope of the modeled WSys, the {Actor-Relationship-Actor}-tuples describe the binary relationships between two entities. The possible combinations include tuples with well-established design patterns.

The tuple {Human-HiR-Tools} describes the basic setting of human supervisory control [13]. By the time when Sheridan established his LoA-scale, automation used to be mostly rather clear-cut control automation following relatively simple rules or algorithms. In most of these systems, the human exclusively owned higher cognitive

capabilities. It was acknowledged in later works [14] that automation could also take over higher cognitive tasks. New scales for LoA have been developed (e.g. [14, 15]), which, to a certain degree, reflect the situation that automation became capable of assuming functions of information acquisition and analysis. These scales are applicable to the tuple {Human-HiR-Agent}. Since then, many works have been conducted, supporting the relevance of this tuple and provide valuable design patterns 7(e.g., [26, 28, 35]). Van Breda and coauthors also provide a good overview [36].

Finally, the {Human-HeR-Agent}-tuple has become particularly interesting in situations where automation-induced human erroneous action should be prevented. We already mentioned a few citations in this context [22, 24, 32–34]. Also further international approaches to adaptive associate systems should be mentioned that are representative to many others [37, 38].

5 Conclusions

In this article, we describe a method for documenting human-autonomy teaming (HAT) design patterns. Therefore, we followed a strict top-down procedure, inspired by systems engineering and cognitive ergonomics. Coming from a general human-systems view, we ended up at the level of frequently recurring actor-relationship-actor-tuples, which may serve as containers for similar or competing design patterns to be found, described, and discussed. However, the benefit of this approach heavily depends on the researchers' and practitioners' efforts to describe their solutions to HAT problems using the offered method. In doing so, we could avail of a great opportunity for rational discussions on future highly automated systems.

References

1. Klein, G., Woods, D.D., Bradshaw, J.M., Hoffman, R.R., Feltovich, P.J.: Ten challenges for making automation a "team player" in joint human-agent activity. IEEE Intell. Syst. **6**, 91–95 (2004)
2. Bradshaw, J.M., Hoffman, R.R., Woods, D.D., Johnson, M.: The seven deadly myths of "Autonomous Systems". IEEE Intell. Syst. **3**, 54–61 (2013)
3. Alexander, C.: The Timeless Way of Building. Oxford University Press, New York (1979)
4. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York (1977)
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
6. Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Menlo Park (1996)
7. Pree, W.: Design Patterns for Object-Oriented Software Development. Addison-Wesley, Reading (1994)
8. Brown, W., Malveau, R., McCormick, H., Mowbray, T.: Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley, New York (1998)

9. Coplien, J.O.: Pattern Languages of Program Design. Addison-Wesley Professional, Reading (1995)
10. Borchers, J.O.: A pattern approach to interaction design. AI Soc. **15**(4), 359–376 (2001)
11. Kruschitz, C., Hitz, M.: Human-computer interaction design patterns: structure, methods, and tools. Int. J. Adv. Softw. 3(1&2) (2010)
12. Kahn, P.H., et al.: Design patterns for sociality in human-robot interaction. In: 3rd ACM/IEEE International Conference on Human Robot Interaction (2008)
13. Sheridan, T.B.: Telerobotics, Automation, and Human Supervisory Control. MIT, Cambridge (1992)
14. Parasuraman, R., Sheridan, T.B., Wickens, C.D.: A model for types and levels of human interaction with automation. IEEE SMC Trans. **30**(3), 286–297 (2000)
15. Miller, C.A., Parasuraman, R.: Beyond levels of automation: an architecture for more flexible human-automation collaboration. HFES **47**(1), 182–186 (2003)
16. Clough, B.T.: Unmanned aerial vehicles: autonomous control challenges, a researcher's perspective. In: Murphey, R., Pardalos, P.M. (eds.) Cooperative Control and Optimization, pp. 35–52. Springer, New York (2002)
17. Huang, H.M.: Autonomy levels for unmanned systems (ALFUS) framework. Volume I: Terminology, version 2.0. NIST special publication, pp. 1–47 (2008)
18. Juziuk, J.: Design Patterns for Multi-agent Systems. Linnaeus University, Sweden (2012)
19. Scerbo, M.: Adaptive automation. Neuroergonomics, pp. 239–252. Oxford University Press, New York (2006)
20. Hollnagel, E., Woods, D.D.: Joint Cognitive Systems: Foundations of Cognitive Systems Engineering. CRC Press (2005)
21. Roth, E.M., Woods, D.D.: Cognitive task analysis: an approach to knowledge acquisition for intelligent system design. In: Topics in Expert System Design, pp. 233–264 (1989)
22. Schulte, A., Meitinger, C., Onken, R.: Human factors in the guidance of uninhabited vehicles: oxymoron or tautology? Cogn. Technol. Work **11**(1), 71–86 (2009)
23. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs (1995)
24. Onken, R., Schulte, A.: System-Ergonomic Design of Cognitive Automation. Studies in Computational Intelligence, vol. 235. Springer, Heidelberg (2010)
25. Shively, R.J., Brandt, S.L., Lachter, J.: Application of Human Automation Teaming (HAT) patterns to Reduced Crew Operations (RCO). In: HCII Conference (2016)
26. Miller, C.A., Funk, H.B., Dorneich, M., Whitlow, S.D.: A playbook interface for mixed initiative control of multiple unmanned vehicle teams. In: IEEE DASC (2002)
27. Miller, C.A., Parasuraman, R.: Designing for flexible interaction between humans and automation: delegation interfaces for supervisory control. Hum. Factors **49**(1), 57–75 (2007)
28. Uhrmann, J., Schulte, A.: Concept, design and evaluation of cognitive task-based UAV guidance. Int. J. Adv. Intell. Syst. (2012)
29. Schulte, A., Meitinger, C.: Introducing cognitive and co-operative automation into UAV guidance work systems. In: Human-Robot Interaction in Future Military Operations. Ashgate. Series Human Factors in Defence, pp. 145–170 (2010)
30. Baxter, J.W., Horn, G.S., Leivers, D.P.: Fly-by-agent: controlling a pool of UAVs via a multi-agent system. Knowl. Based Syst. **21**(3), 232–237 (2008)
31. Bainbridge, L.: Ironies of automation. Automatica **19**(6), 775–779 (1983)
32. Onken, R., Walsdorf, A.: Assistant systems for aircraft guidance: cognitive man-machine cooperation. Aerosp. Sci. Technol. **5**(8), 511–520 (2001)
33. Theißing, N., Schulte, A.: Designing a support system to mitigate pilot error while minimizing out-of-the-loop-effects. In: HCII Conference (2016)

34. Strenzke, R., Uhrmann, J., Benzler, A., Maiwald, F., Rauschert, A., Schulte, A.: Managing cockpit crew excess task load in military manned-unmanned teaming missions by dual-mode cognitive automation approaches. In: AIAA GNC Conference (2011)
35. Lange, D.S., Gutzwiller, R.S.: Human-autonomy teaming patterns in the command and control of teams of autonomous systems. In: HCII Conference (2016)
36. Van Breda, L., et al.: Supervisory control of multiple uninhabited systems: methodologies and enabling human-robot interface technologies. NATO RTO HFM-170 Report (2012)
37. Taylor, R.M., Howells, H., Watson, D.: The cognitive cockpit: operational requirement and technical challenge. contemporary ergonomics, pp. 55–59 (2000)
38. Miller, C.A., Hannen, M.D.: The Rotorcraft Pilot's Associate: design and evaluation of an intelligent user interface for cockpit information management. Knowl. Based Syst. **12**(8), 443–456 (1999)