Using self-awareness in decentralized computing systems

Florian Maier
Faculty of Computer Science and Mathematics
University of Passau
Passau, Germany
maier162@stud.uni-passau.de

***Abstract* — The term self-awareness in technological systems has been discussed many years now. There is no commonly agreed definition of the term self-aware in biological or psychological meaning therefore there are many different definitions all stating different aspects and levels of what we call self-aware in the biological world. In addition the system should be characterized by properties such as: robustness, decentralization, flexibility and self-adapting. In the past this was often achieved by designing good and robust but also complex algorithms which often lead to unnecessary overhead and hard to fix runtime bugs. Using self-aware components can turn such algorithmic systems into organic computing systems, offering better scalability, more robustness of the global state and less unnecessary overhead in the communication between different components in the decentralized system. On the counterpart such a system might work in a way that cannot be fully understood by humans in a reasonable time leading to other problems such as trust issues in the system or unwanted behavior in the global state of the system. The goal of this article is to state out how decentralized computing systems can benefit from self-aware approaches.**

Keywords — self-awareness, decentralized computing systems

I. Introduction

In our modern world the demand of large and complex computer systems rises, including high decentralization, local applications with local sights and a high network usage. Implementing and servicing these systems can be time and cost intensive if one uses traditional algorithmic approaches. A more modern approach is to use independent small systems with local sight linked together. Such systems can be declared as self-aware, as will be discussed later.

This article will discuss the question how to benefit from self-aware components in decentralized computing systems, meaning sort of artificial intelligence on small individual components, including challenges to face.

Section II. will discuss the term "self-aware" in computer systems, stating out different approaches to define the term and finally looks detailed into a specific definition, which will be used as a reference definition for the rest of the article.

Section III. than looks into decentralized technical systems, what they mean and how they have been implemented in the past using algorithmic approaches. Than it will take a look at how self-aware components can be and also are used in such systems.

How such components can be applied to decentralized computing systems and where the differences, opportunities and challenges lie compared to classical algorithmic approaches will be discussed in Section IV. Afterwards there will be looked at how organic computing links to such systems.

Section V. concludes this paper.

II. Background: Self-Awareness

There is no commonly agreed definition of the term self-awareness in biological means. Therefore, there are many different approaches to define "self-aware" for technological systems. Some of them that stand out as significant contribution to the definition of self-awareness in computing systems will be stated in this section.

A. *The term "self-aware" in technological systems*

One of the higher levels of self-awareness can be called meta-self-awareness or meta-cognition. This sort of self-awareness is defined as an organism that is able to construct and reason about an abstract symbolic representation of itself [1] or can also be called knowing about knowing [2]. With this definition Cox [3] argues that being aware of itself is being able to use possessed information to achieve goals, which can also include that the information might be modified. In addition he also states that meta-cognition is similar to the algorithm selection problem, which task is to choose the most efficient algorithm from a set of possibilities.

Agarwal et al. [4] state that the design philosophy should move from a procedural design, meaning the behavior of a computing system is pre-programmed and defined during the design time, onto self-aware systems where pre-programming is not necessary and the system can adapt to its context automatically at runtime. One goal of this is to let the system handle the availability of resources on its own, resulting in less design effort and a higher possible performance at runtime. They also state that there should be five design properties considered while designing self-aware systems:

- introspective, i.e. to perceive and optimize their behavior,
- adaptive, i.e. being able to adapt to changing needs of other applications relying on them,
- self-healing, i.e. being able to make corrections if errors occur,
- goal oriented, i.e. trying to meet user application needs, and
- approximate, i.e. the system is able to automatically choose how precise a task must be executed.

Pervasive computing is primarily about mobile systems. Those systems are forced to monitor their environment for changes and adapt to those changes to meet the needs of human interaction. I.e. such a system must be able to detect in what situation the human is currently in and therefore adapt the behavior of the program automatically. Lewis et al. [5] describe this sort of behavior as self-aware pervasive computing systems because they monitor their environment and self-adapt automatically. Because this sort of system is context-aware, at least a lower self-awareness is implemented there.

As discussed above, self-awareness can be located in one single entity or system, that is aware of its environment or itself. But self-awareness can also be applied to larger systems with many entities that are self-aware together but every single entity on its own can't be described as fully self-aware. This is called emergent self-awareness [6]. This sort of distributed system helps the global state to stay robust if errors occur in local states. This is achieved by the ability of local entities to be aware of disturbances in the global state. If every entity collects local information, it is possible to get enough statistical information in the whole system so that recovery mechanisms can be applied and the global state can be stabilized again. With this sort of self-awareness an artificial intelligent system can be decentralized.

There are also more formal approaches to reach self-awareness, especially in complex systems. Newer approaches to represent knowledge within those systems in formal deterministic and probabilistic ways are discussed by Vassev and Hinchey [7]. According to them this can lead to better self-awareness because it is easier to analyze current local states and goals of parts of the global system.

B. *Peter R. Lewis' et al. definition of "self-aware"*

In the above section several different approaches to define the term self-aware have been discussed. All of them have their right to exist; they tackle different problems with sometimes different, sometimes the same solutions. To be able to discuss the application of self-aware computing systems in decentralized systems, I will use the comprehensive definition of Lewis et al [5]. They discussed all above mentioned definitions and tried to sum up the general concept of self-aware computing system definitions. With this definition it is easier to discuss general, not application specific algorithms.

Their definition is based on a conceptual component called a self-aware node. Such a node does not need to be existent in physical or software meanings but it defines a concept of what is meant by local within a global system; even more than that it defines the concept of what is seen as "self" in a self-aware system. Because those conceptual self-aware nodes are distributed components per definition, this particular approach to define the term self-aware is hugely useful for the task of tackling distributed systems problems with self-awareness. The definition is as follows.

To be self-aware a node must:

- Possess information about its internal state (private self-awareness).
- Possess sufficient knowledge of its environment to determine how it is perceived by other parts of the system (public self-awareness).

Optionally, it might also:

- Possess knowledge of its role or importance within the wider system.
- Possess knowledge about the likely effect of potential future actions / decisions.
- Possess historical knowledge.
- Select what is relevant knowledge and what is not.

With this definition it is clear that a node must both have private and public self-awareness. If it had only private self-awareness, it would have knowledge about itself, e.g. its current state, its behavior or history knowledge. It would not have any information about even the existence of other nodes in the system, leading it to not communicate with the other nodes and therefore it would not be able to react to changing needs of other parts of the system. On the other hand if it had only public self-awareness, it would only have access to knowledge of other nodes or more precisely the information other nodes are sharing with the node. The node would not have information about itself and therefore would not know what it currently executes.

If a node has access to both private and public self-awareness it is able to combine all information of both sources, which creates a meaningful context for the node and therefore makes it possible for the node to adapt and change its behavior perfectly to the needs of the global system. On top this knowledge allows to support simple reactive behavior of the node as well as way more complex tasks like self-learning and prediction making.

III. Challenge: Decentralized technical systems

Currently the term self-awareness has been discussed in detail. For the goal of this paper, researching how self-awareness can contribute to decentralized systems, we first have to take a look at what those systems are and how they function traditionally.*

Decentralized technical systems are a subset of distributed systems. According to Coulouris et al. a distributed system is a system in which hardware and software components located on network computers communicate and coordinate their actions only by passing messages [8]. While distributed systems have several parts that are spread across several units, they still can have one master unit that combines all information and is responsible for action choices, decentralized systems don't have such an entity. Every unit in those systems takes individual choices and only the sum of all those actions is what the system as a whole is doing. The absence of a central master unit makes such systems much more stable against local disturbances.

A. *Algorithmic approaches*

Many decentralized systems rely on algorithmic approaches to achieve their goals.

First of all, architectural modeling is used to define the details of the organization of the components within the decentralized system. One commonly known approach here is peer-to-peer. In a (pure) peer-to-peer approach, all participants of the system are equally rated and have no master. Everyone can communicate with everyone else and the system has no hierarchical system. There are also some hybrids that are based on peer-to-peer but have hierarchical

* Details in this chapter are based on knowledge learned by Florian Maier in the course Distributed Systems held by Michael McMahon at Waterford Institute of Technology, 2016.

structures like a super-peer.

To make it possible for units within the decentralized system to communicate with each other, one approach is Remote Method Invocation or RMI, which is Java exclusive. In this approach, the network itself is seen as a computer, which makes it possible to use classic system designs like object orientating across several units in the system. RMI enables one unit to make method calls on another unit within the system and therefore share and gather information and call actions on other parts of the system. Similar approaches to RMI are Common Object Request Broker Architecture, also named CORBA and the Microsoft exclusive .NET.

The big problems of decentralized systems are performance including scalability, reliability, security and adaptability. Those problems will first be discussed in detail and in the next chapter addressed and shown how to solve them with self-aware nodes.

In an algorithmic approach the designer has to think of how many resources the system needs and how it can scale up if necessary. If he chooses too little resources, the system might get very slow, if he chooses too many, resources are wasted because they are held for the system even if they are not needed. Hitting the sweet spot can be a tough challenge, especially considered that the demands of such a system can change over time.

A distributed system has to work reliable, which is often a problem if an unreliable network is used to communicate between several nodes. A programmer of such a system has to preconsider the failure of network components or complete units, design and implement backup plans, e.g. using other routes through the network or doubling key units etc. Taking all possible failures into account can be a hard task if the system should not be overscaled.

Large decentralized systems also have to take security into account. The larger and more network based a system is, the easier is it for an attacker to get into the system. On top, if there is no central master overviewing the whole system, an attacker could also implement a node that links in the network, impersonating as a normal part of the system but disturbing the correct function of the global state. An algorithmic approach additionally suffers from predictable behavior, making it easier for an intruder who knows the response of all or many components in the system to infiltrate it.

Another big problem of algorithmic approaches is adaptability. It is pretty common that requirements in the real world change, thus the requirements for decentralized systems also change. Adapting the new needs into a system can be time consuming in an algorithmic approach because either a new element of the system has to be designed and implemented or an existing one has to be updated. Both of these methods involve at least one person to make actively changes to implement the needed changes and therefore to adapt the system to the new requirements.

B. *Self-aware components in decentralized computing systems*

Mitchell [9] researched self-awareness in decentralized biological systems – namely the immune system and ant colonies – and abstracts four principles to adapt to Decentralized Computing Systems.

- Global information is encoded as statistics and dynamics of patterns over the system's components
- Randomness and probabilities are essential
- The system carries out a fine-grained, parallel search of possibilities
- The system exhibits a continual interplay of bottom-up and top-down processes

The task of interpreting the statistic information is done by all units locally with local statistical information, what leads to a system adaptation that fits the current needs. Randomness is needed to be able to detect and react to much more events in the environment. Having a large number of relatively small units helps under these circumstances for a stable functionality of the whole system. On top it is crucial to obtain a good balance between bottom-up and top-down processes, including shifts of how they have to be weighted to be balanced, thus this balance shifts over time.

C. *Current state of the art systems*

Esterle et al. [10] used self-aware agents to operate a distributed smart camera system. They had the goal to track moving objects within their field of view. To achieve that, every camera on its own communicates with its neighbors, making intelligent decisions about what camera is responsible for what objects and who communicates with whom. The result is that the cameras have a well balancing between tracking performance and communication overhead. On top the cameras did not need to know anything about their environment or the system structure in design, programming or installation time.

Decentralized systems with self-aware components can also be of a much simpler scale. One example is cognitive radio devices [11]. They monitor and control their own abilities and communicate that with other radio devices. Doing this improves the efficiency of communication because the devices can adapt to parameter changes easily on their own.

A third example is the SWARM-BOTS project [12]. They developed so called s-bots. Their individual abilities are strongly limited but they can communicate with local neighbors. Through this they can assemble into a larger structure called Swarm-bot. Doing this allows them to achieve goals a single s-bot could not achieve like navigating over difficult terrain or the transportation of large objects.

IV. Application of Self-Awareness in Decentralized Systems

Now that we have discussed what self-awareness means, what definition will be used in this paper and what decentralized technical systems are, the following section will look at the opportunities that self-aware components give against algorithmic approaches and what challenges come with it. Also a brief look at organic computing will be made.

A. *Opportunities of self-aware approaches opposed to algorithmic ones*

Section III discussed four major problems with decentralized technical systems. Self-aware components can help solving these problems.

Self-aware components in decentralized technical systems are able to observe their environment and adapt to changing needs on the fly. With this it is pretty easy for such a system to scale the needed resources up and down depending on current runtime needs. Therefore, the resources do not have to be predefined in the design phase.

Reliability in decentralized technical systems can be improved by using self-aware components, as they are able to operate on a local scale to detect failures in the system, react and adapt to them and therefore solving the problem during runtime, creating a stable global state again. Therefore, a programmer does not have to put as much effort into error correction as he or she would have to do in an algorithmic approach.

As self-aware components are much less predictable as algorithmic ones, it is way harder to foretell their actions, making it harder for hackers to attack the system. Also, one dysfunctional component within the system will have much less effect on the global state because all other self-aware nodes will detect the failing behavior and therefore adapt their actions in such a way that a stable global state is established again.

The last mentioned challenge is adaptability. As discussed above, the biggest strength of self-aware components is that they can adapt their behavior very good during runtime depending on environmental changes or new goals to achieve. So with a self-aware approach adaptability is no challenge at all for a decentralized technical system.

B. *Challenges with self-aware components*

Self-aware and self-adapting components in decentralized technical systems might behave in an unpredicted way. It also can be very hard to track down why such components behave the way they do. This makes it very difficult to find and fix bugs. On top it might be possible that all nodes in such a system might do unforeseen things, resulting in an unwanted behavior of the global state of the system. Therefore, it might be hard for designers and programmers to implement a system that behaves exactly as wanted and for the users to fully trust such a system.

C. *Relation between self-aware decentralized systems and organic computing*

According to Tomforde, Sick and Müller-Schloer [13] an organic computing system is a technical system equipped with a potentially large set of sensors and actuators. This highly overlaps with Lewis' et al. definition of self-awareness discussed in Section II. There, several nodes (actuators) have to receive knowledge about themselves and their environment (sensors). Tomforde, Sick and Müller-Schloer also argue that an organic system has to adapt to changing needs on its own, as has been discussed for self-aware components in decentralized computing systems before. Tomforde et al. [14] also defined self-organization, self-configuration, self-repair and adaptation as parts of organic computing, where all these attributes are also core components of self-aware distributed systems as discussed above.

V. Conclusion

In this paper several approaches to the term "self-awareness" has been discussed. Synthesizing on that the more general and comprising definition of Lewis et al. was introduced. Afterwards the term decentralized technical system has been defined and several tasks that are hard to fulfill with algorithmic approaches were discussed. Self-aware components can contribute to those kinds of systems in a way that those components can solve formerly hard tasks on their own with little effort of the designer and programmer. On the counterpart there are new challenges to face with those components. Having organic computing in mind, this sort of new way of designing and programming decentralized systems will be the way to choose in the future nonetheless.

References

[1] A. Morin and J. Everett, "Conscience de soi et langage interieur: Quelques speculations. [self-awareness and inner speech: Some speculations]," Philosophiques, vol. XVII, no. 2, pp. 169–188, 1990.

[2] J. Metcalfe and A. P. Shumamura, Eds., Metacognition: Knowing about knowing. Cam, MA, USA: MIT Press, 1994.

[3] M. Cox, "Metacognition in computation: A selected research review," Art. Int., vol. 169, no. 2, pp. 104–141, 2005.

[4] A. Agarwal, J. Miller, J. Eastep, D. Wentziaff, and H. Kasture, "Self-aware computing," MIT, Tech. Rep. AFRL-RI-RS-TR- 2009-161, 2009.

[5] P. R. Lewis, A. Chandray, S. Parsons, E. Robinson, K. Glettey, R. Bahsoon, J. Torreseny and X. Yao, "A Survey of Self-Awareness and Its Application in Computing Systems," in Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on IEEE, 2011. pp. 102-107.

[6] M. Mitchell, "Self-awareness and control in decentralized systems," in Metacognition in Computation, 2005, pp. 80–85.

[7] E. Vassev and M. Hinchey, "Knowledge representation and awareness in autonomic service-component ensembles – state of the art," in 14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing. IEEE Computer Society, March 2011, pp. 110–119.

[8] G. Coulouris, J. Dollimore, T. Kindberg, "Distributed Systems – Concepts and Design", 3rd Edition, Addison-Wesley, ISBN 0-201-62433-8, 2000

[9] M. Mitchell, "Self-awareness and control in decentralized systems", in AAAI Spring Symposium, Metacognition in Computation, 2005, pp. 80-85.

[10] L. Esterle, P. R. Lewis, M. Bogdanski, B. Rinner, and X. Yao, "A socio-economic approach to online vision graph generation and handover in distributed smart camera networks," in 13th International Conference on Distributed Smart Cameras (ICDSC 2011), Ghent, Belgium, 2011, in press.

[11] B. Fette, "Cognitive radio technology", Academic Press, 2009.

[12] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo, "Swarm-bot: A new distributed robotic concept," Autonomous Robots, vol. 17, pp. 193–221, 2004.

[13] S. Tomforde, B. Sick, C. Müller-Schloer, "Organic Computing in the Spotlight", in arXiv preprint, arXiv:1701.08125, 2017

[14] S. Tomforde et al., "Observation and Control of Organic Systems", in Organic Computing - {A} Paradigm Shift for Complex Systems, 2011, pp. 325-338.