A Conceptual Model of Self-Adaptive Systems based on Attribution Theory

Nianyu Li (li_nianyu@pku.edu.cn)
Key Laboratory of High Confidence, Peking University, China

Zhengyin Chen (chenzy512@pku.edu.cn)
Key Laboratory of High Confidence, Peking University, China

Zi-Long Li (zl.li@imt-atlantique.net)
IMT Atlantique, France

Wenpin Jiao (jwp@pku.edu.cn)
School of Electrical Engineering and Computer Science, Peking University, China

Abstract

The development of self-adaptive systems has attracted lots of attention as they can adapt themselves autonomously to environmental dynamics and maintain user satisfaction. However, there are still tremendous challenges remained. One major challenge is to guarantee the reusability of the system and extend the adaptability with the changing deployment environments. Another challenge is to ensure the adaptability coping with the open and complex environments with the existence of unknown. To solve these problems, we introduce a conceptual self-adaptive model, decoupling the environment with the system. This model is a two-layer structure, based on internal causes and external causes from attribution theory. The first layer, determining how the internal causes affect the adaptation behaviors, is independently designed and reusable; while the second layer, mapping the relationship between external causes with internal causes, is replaceable and dynamically bound to different deployment environments.

Keywords: Self-Adaptation; Attribution Theory; Reusability

Introduction

Current society extensively relies on software systems to achieve specific goals. However, achieving those required goals is a tremendous challenge (Cheng, de Lemos, & et al., 2009) since there are lots of uncertainties that developers have not considered or cannot fully understand during design time, and the changing environment leads to costly reconfiguration and time-consuming maintenance tasks (de Lemos, Giese, & et al., 2010). Therefore, there is a high demand for managing complexity reduction and achieving desired goals within a reasonable cost and timely manner. Self-adaptation is generally considered as one of the most promising approaches to manage the uncertainties of modern software systems since it enables a system to adapt itself autonomously to user requirements or environmental dynamics to continuously achieve system goals including performance, security, fault management, etc (Sawyer, Bencomo, & et al., 2010).

In the existing literatures, most of the adaptation behaviors are triggered by events in the environment (Salehie & Tahvildari, 2009; Shevtsov, Berekmeri, & et al., 2018; Filieri, D'Ippolito, & et al., 2017; Modoni, Trombetta, Veniero, Sacco, & Mourtzis, 2019). That is to say, the main adaptability of a self-adaptive software system is the internal response to the changes in the external environmental factors. Accordingly, the whole lifecycle of the adaptive system, including design time and run-time, is always associated with the environment where the system is deployed. In the design phase, system environment, as well as the mechanisms of perceiving and effecting environment are modeled and implemented. And the set of adaptation policies, tightly binding to this specific environment, are defined and customized. Then at run-time, adaptation behaviors could be achieved by implementing the activities of a well recognized feedback control loop called MAPE-K (Monitoring, Analysis, Planning, and Execution with Knowledge).

One of the disadvantages of current method is that these adaptation policies bound tightly to a specific environment will inevitably limit the adaptivity of the system to various deployment environments. Take a robot system as the example. In a wood floor environment, there could be policies describing how fast the robot should move forward to reach its destination as soon and as safe as possible; or how many angles it shall turn when encountering obstacles. However, the value of speed and angles will be very different in a more slippery tile floor or on a rough cement road. Therefore, for an adaptive system, in addition to being able to adapt in the specific deployment environment, it should have a wide range of applications (i.e., being deployed in a variety of environments). The other disadvantage is that current adaptive systems cannot cope with the increasing complexity and openness of the environment. It is basically impossible to pre customize a complete environment model since the developer cannot fully understand or have considered at design time. Inevitably, new environmental factors might exist and appear at run-time and the system is not reliable to recognize or predict those unforeseen. For example, when designing the adaptive strategies for a robot avoiding obstacles, it is necessary to know in advance what kind of obstacles it might encounter, and then to specify how to deal with obstacle A, obstacle B, etc. Obviously, obstacles in the environment could be infinite. New and unexpected obstacles will emerge constantly in the practical environment, which leads to the inadequate capacity of the existing system.

The fundamental reason for these disadvantages lies in the tight bound between the specific environment and the system. To deal with current challenges, this paper comes up with a novel approach based on attribution theory. Philosophically, the internal causes are the fundamental reasons for the change or the development of things while the external causes are merely the conditions and become operative through internal causes. In other words, it is when the external causes lead to the changes of internal causes that the adaptation behaviors could be triggered. Therefore, the basic idea behind our approach is to decouple the environment with the system both at two stages: independent design and run-time binding. In the design stage, how the adaptation behaviors of the system are determined by the internal causes is focused and emphasized, which makes the design and development of software are independent of the practical environment. In the run-time stage, the relationships between environmental factors (i.e., environmental events as external causes) and state of the system (i.e., internal causes) are dynamically established, thus binding the system to the specific application (i.e., deployment) and realizing the environmental-related adaptability.

The main contributions of our research is summarized as follows:

- We propose a new conceptual model of designing adaptive systems based on the attribution theory;
- We describe a two-layer structure in accordance with the conceptual model. The first layer is the independent design with decisive adaptation policies pertaining the relation between internal causes to adaptation behaviors; while the second layer is the dynamic bound with influential adaptation policies connecting the external causes to internal causes.

Approach Overview

So fundamental is the process of asking and answering "why" questions – trying to figure out what caused something else – that it has been characterized as a basic human activity, and a family of theories has been developed to illumine how and why things happen as they do. This set of theories, collectively called Attribution Theory initiated by Fritz Heider(Heider, 1958) and further advanced by Harold Kelley and Bernard Weiner(Kelley, 1967), attempts to describe and explain the processes involved in everyday explanations, most typical explanations of individual behaviors and events. An interesting example that someone is angry could be attached to the causes of bad-tempered characteristics or something bad happened.

There are a number of definitions for attributions, but a common way to define attributions is as the internal and external process of interpreting and understanding what is behind individual behaviors. External attribution, also called situational attribution, refers to interpreting the causes of behaviors to the situational or environment features outside a person's control. Internal attribution is the process of assigning the causes to some personality traits, rather than to outside forces.

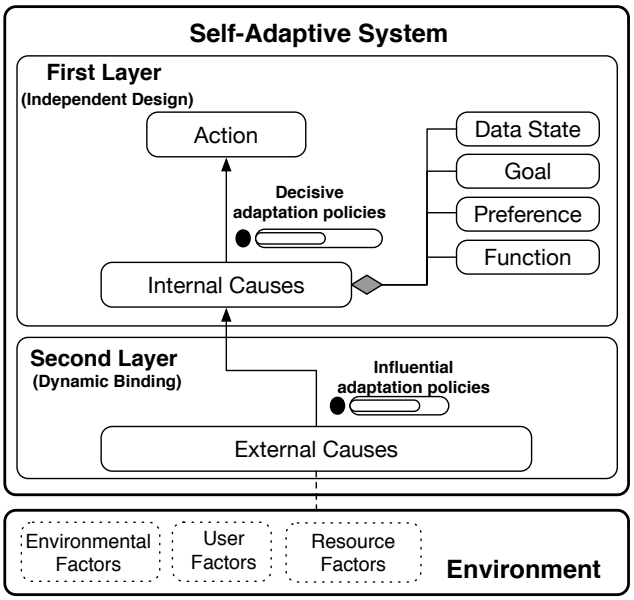

Figure 1: Conceptual Model of Attribution-Based Self-Adaptive System.

Similarly, adaptive behaviors of self-adaptive systems are the reactions to the external causes, i.e., changes. It is important to identify the reason for an adaptation: *why do we have to adapt?*. This is the central question influencing the reaction. In general, the reasons for an adaptation could be i)a change in the technical resources, e.g., the availability of an alternative network connection; ii)a change in the environmental variable, e.g., the workload for a website changed; and iii)a change regarding the user, e.g., a change in the user goal or the user preferences(Krupitzer, Roth, & et al., 2015). Users and operative technical resources could be regarded as a part of the environment and together with the environmental factors form the periphery of adaptive systems (Jiao & Sun, 2016). They are the external conditions of the existences and referred to as external causes.

Factors or events in the environment are not the necessary conditions that systems could execute reactive behaviors. For example, when the number of active users increases, some of the websites may saturate while others may not. In fact, whether an application deployed in the cloud is saturated and then allocated with more resources depends on whether the response latency (the time elapsed from sending the first byte of the request to receiving the last byte of the response) is long, and one will not do so if the latency is within a satisfactory range even if this application is with a huge number of users. In other words, the change of the user number does not determine the adaptive behaviors on an allocation of the resource; instead, the influence could take effect only when the change affects system internal states, which further im-

pair system goals. On the contrary, internal changes on the states of the system itself are the intrinsic reasons for adaptive behaviors.

Figure 1 provides a birds-eye view of our conceptual model of self-adaptive systems. An adaptive system can be divided into two layers, each corresponding to the internal and external attribution process. The first layer is composed of internal causes (including data state, function, preference, and goal) and adaptation behaviors (i.e., actions). Decisive adaptation policies, how to cope with changes in internal causes, determine the relations between internal causes and actions. This layer is independently designed and fixed even with the changing deployment or the extended environment. Note that the changing deployment is the change from one specific environment to another; the extended environment is the open environment with environmental factors from unknown to known. In the second layer, the relationship between external causes and internal causes is denoted in the influential adaptation policies. The second layer is a replaceable component and dynamical bound when the running environment is determined.

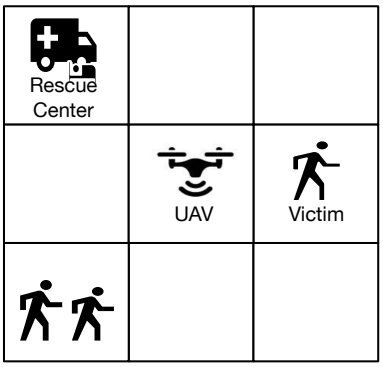

Figure 2: Running Example

Running Example. As a motivating example showcasing our approach, consider a *disaster relief* mission in adaptive system domains (Li, Jiao, & et al., 2018). In such a scenario, communication infrastructure is disabled in a city due to disasters; parts of the city may be unsafe. Figure 2 visualizes a possible configuration of a part of the city (i.e., a district). The rescue center is the safe zone in charge of the district's safety. The district then is divided into several blocks. The victims are spread in different blocks and have no idea of where the rescue center is. The unmanned aerial vehicle(abbr. as UAV) will be arranged to search and guide victims to the rescue center. In the process of searching for the victims, the UAV should not only guarantee the search and rescue task quick and thorough (search and rescue all victims in an acceptable time), but also ensure its own safety (no crash) and energy storage (avoid battery depletion). Furthermore, we expect that this UAV system can participate in missions in various disaster environments, such as fire, floods, earthquakes etc.

Formal Definition of the Conceptual Model

Inspired by the attribution theory, external causes take effects on the system through the internal causes, instead of directly affecting or determining the behaviors of the system. To this end, the conceptual model (M) of a self-adaptive system is defined as a tuple $\mathtt{M} = (\mathtt{IC}, \mathtt{DAP}, \mathtt{IAP})$, where:

- $\mathtt{IC}$ are the internal causes (i.e., intrinsic reasons for adaptation behaviors) which can be further specified as a tuple $\mathtt{IC} = (\mathtt{Data\ State}, \mathtt{Goal}, \mathtt{Preference}, \mathtt{Function})$;
- $\mathtt{DAP}$ are the decisive adaptation policies, which define how the internal causes of the system determine the adaptation behaviors and are generally expressed as rules of "internal causes – actions";
- $\mathtt{IAP}$ are influential adaptation policies, denoting how events in the environment affect the changes of internal causes with the form of "external causes – internal causes".

Internal Causes and System State

Data State The remembered information of the system determined by a set of attribute values. Let $(Attr = a1, ..., an)$ be the attribute set of the system, and $(Dom = dom(a1), ..., dom(an))$ be the set of domains of these attributes. Then the data state of the system is the mapping of these attributes to their values. In the motivation example, UAV system needs to maintain certain data, such as current location, residual power, flight height, searched blocks, unsearched blocks, hazard blocks, status(i.e., cruise or guidance). The data state of UAV is defined by the value of these kinds of information.

Goal the data state that the system expects to achieve or maintain. Generally, goals can be classified into three categories (Filieri et al., 2017). One type of goal is a reference value, called setpoint, to track. In this case, the objective is to keep a measurable quantity as close as possible to the setpoint. The second category of the goals is the variation of the classic setpoint-based goal where the goal resides in a specific range of interest with confidence intervals. The third category of goals concerns the minimization (or maximization) of a measurable quantity of the system. In substance, these goals can be regarded as functions of data states:

$$\mathtt{Goal} = \{\ \mathtt{g} \mid \mathtt{g} \in 2^{\mathtt{Data}} \wedge \mathtt{eval}(\mathtt{g}) = 1\}. \tag{1}$$

In general, the system is considered completely achieving a goal if it enters the target data state; however, in some cases, the system can only (infinitely) get close to the target but not reach. For example, it is impossible to require the UAV's flighting speed "to maintain exactly at 2 meters per second", then the system is said to be achieving the goal to some extent. Therefore, a goal is usually associated with an evaluation function which determines whether the goal has been achieved or how far it has been achieved. For

instance, in this mission scenario, should all the blocks had been searched and rescued, the evaluation result of this goal is one. If it is not, the evaluation function is $\texttt{eval}(\texttt{g}) = \texttt{num}(\texttt{searched blocks}) \div \texttt{num}(\texttt{total blocks})$, and pertains that goal satisfaction is directly proportional to the number of blocks cruised.

Preference The data state that the system is more interested in. Contrary to the goals that the system must be achieved or maintained, the preferences are not necessary must-to-do, but the performance of the system would be better if they are met. For example, the UAV is not only expected to complete the search and rescue goal but also with economical (less electricity consumption) and fast preference (all the victims should be searched as soon as possible). Similar to the goal, the preferences are associated with the utility functions that measure the satisfaction of preferences. For example, a utility function is present: $\texttt{u} = \texttt{residual power} \div \texttt{total energy storage}$, illustrating the tendencies of less energy consuming.

$$\texttt{Preference} = \{\, \texttt{p} \mid \texttt{p} \in 2^{\texttt{Data}} \wedge \texttt{util}(\texttt{p}) \in [0,1] \}. \quad (2)$$

Function The means or methods of achieving the goals. The functions are essentially changing or maintaining system status through manipulating controllable variables. For example, the UAV has the functions of take-off, flying, landing, direction change, etc.

$$\texttt{Function} = \{\texttt{f} \mid \texttt{f} : \texttt{Data} \rightarrow \texttt{Data}\}. \quad (3)$$

System State The state of the system is determined by the internal factors. In other words, data state, the available functions, and the satisfaction of goals and preferences together define the system state. Note that a function is not always valid (sometimes not working); the valid data of the system is the combination of attribute values and functions(i.e., $\texttt{Data} = \{\texttt{s} | \texttt{s} \in (\texttt{Attr} \times \texttt{Dom}) \cup 2^{\texttt{Function}}\}$)

Adaptation Policies

In the complex and uncertain environment, many kinds of environmental factors (discovered and to be discovered) exist, thus resulting in complicated influences on the internal causes. It is not necessarily true that all the changes of environmental factors will affect the system thereupon triggering adaptation behaviors. Only those leading to the changes of internal causes have an impact on the self-adaptive system.

Influential Adaptation Policies The IAP describe how the external causes especially environmental factors affect the internal causes of the system. These external causes directly influence the data state, which will further affect the functions, preferences and goals. In a fire scenario, the environmental factors for the motivation example could be the magnitude of the fire, which inevitably results in different data states for the UAV system. For example, the detection of a serious situation (i.e., high magnitude) for a block would render the UAV marking it as hazard and UAV will try to avoid this block cruise in a certain amount of time for its own safety. However, if in an earthquake scenario, the obstacles from the ground would probably not affect the mark of the block which is stored as an internal data since it is not a threat to high altitude flying UAV.

$$\texttt{IAP} = \{\texttt{p}_\texttt{i} \mid \texttt{p}_\texttt{i} : \texttt{ExtFactors} \rightarrow \texttt{Data State} \times \texttt{Function} \times \texttt{Preference} \times \texttt{Goal}\}. \quad (4)$$

Decisive Adaptation Policies The DAP characterize how the internal causes determine the self-adaptive actions of the system. An action is the operation of a function, which means that taking an action is to perform a function. $\texttt{Action} = \{\texttt{a} | \texttt{f} \in \texttt{Function}, \texttt{a} = \texttt{Do}(\texttt{f})\}$. The factors that determine the adaptive actions may involve system states, functions that the system possesses, preferences and goals. For the current location of UAV as shown in Figure 2, without detected victims to be guided to the rescue center, actions of four direction changes (North, South, East, West) are available if all corresponding blocks have not been cruised before and no hazard mark for the time-being.

$$\texttt{DAP} = \{\texttt{p}_\texttt{d} \mid \texttt{p}_\texttt{d} : \texttt{Data State} \times \texttt{Function} \times \texttt{Preference} \times \texttt{Goal} \rightarrow \texttt{Action}\}. \quad (5)$$

Through this conceptual model with a tuple structure, adaptation behaviors are achieved by explicitly defining internal causes and reasoning about the influences of external events on internal causes via IAP, upon which reactive actions are acquired by DAP. Concretely, for the motivation example, the UAV can infer the influences of environmental factors on its internal causes; and then the UAV can reason about its DAP to decide its adaptation behaviors. With the two layer structure, this conceptual model is supposed to be characteristic with applicability and reusability. Applicability entails the appropriateness of our attribution theory based conceptual model to design the self-adaptive systems and to capture dynamics of the environment, triggering IAP and DAP while maintaining satisfaction on system goals. Reusability describes usability of the DAP without modification, especially coordinating with various alterable dynamic binding IAP to different deployment environments and extended environments allowing continuously gaining knowledge.

Implementation Model

This section provides an implementation model of our attribution based approach. Adaptation builds on adaptation policies characterizing casual relationships between external and internal causes of a system, and between internal causes and actions in the knowledge. Adaptation behaviors are achieved by

implementing the activities of the MAPE (Monitoring, Analysis, Planning, Execution) loop (Kephart & Chess, 2003). Analysis and Planning are responsible for identifying possible requirements violations and generating an adaptation strategy, respectively, while Monitoring and Execution are responsible for enacting it at runtime.

Knowledge

To achieve self-adaptation, the system needs to be tailored, mainly regarding to the adaptation policies in the knowledge. This component Knowledge (K) is shared by all MAPE components. Ideally, all of the knowledge should be reusable across the same class of systems, i.e., these systems can adapt to all kinds of deployments and achieve user goals. However, the generality of this component comes at a cost:

- A significant amount of system-specific knowledge needs to be specified and maintained to apply the system to different deployment environments.

- Should the need for changing the K arises with the gain of information from the environment, the whole component shall be revised separately and deployed to aid (correctly) user expectation.

To support the extendability of new information and reusability of adaptation policies across different systems, we separate system-specific knowledge from the environment-specific part, echoing the two-layer structure in the conceptual model. System-specific knowledge, denoted as the DAP and instructing the behaviors to certain states of the system, is fixed and reusable between systems in similar functions. Meanwhile, environment-specific knowledge for defining how events in the deployment environment (external causes) affect system state in the form of IAP, is alterable as the deployment changes or discovery of additional environmental factors impacting system state. This is faithful to the principle of separation of concerns – the principle for separating a design into distinct sections, such that each section addresses a separate concern (Dijkstra & W, 1982).

MAPE Loop

For a specific deployment environment, adaptation behaviors are achieved by following the widely adopted mechanism of MAPE loop, which is shown in the implementation model in Figure 3.

Monitor Events generated in the environment indicating the execution of system actions or natural changes in the external factors are received. Component Monitor (M) gathers or synthesizes particular data through probes (or sensors) from the environment, and saves data in the knowledge in the form of external causes. For our example, events can indicate a serious fire detected by the cruising UAV.

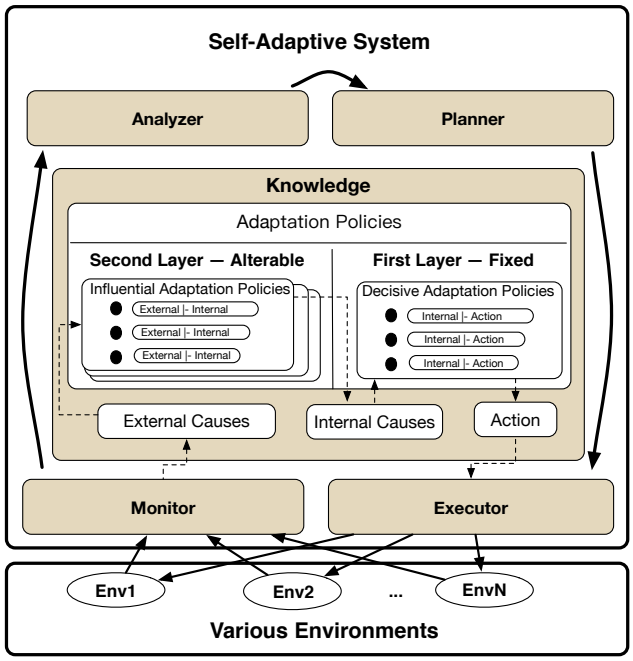

Figure 3: Implementation Model of Attribution-Based Self-Adaptive System.

Analyzer During speculative analysis, conditions of the environment representing violations of goals or better satisfaction of goals which can arise when active entities perform actions are identified. The component Analyzer (A), with the input of external causes from the Monitor, performs analysis by starting adaptation policies engine and reasoning about `IAP` to acquire the data state of the system. On this basis, analyzer further checks whether the goal is fulfilled; preference is satisfied; an adaptation is required. A typical example could be a new mark of hazard blocks resulted from the fire situation which might endanger its own safety.

Planner Component Planner (P) composes a workflow of adaptation actions aiming to counteract violations of system goals or better achieving goals. It consists of one or a set of actions to be enacted inferring from `DAP` in the adaptation policies engine receiving internal causes as input. For each situation, it identifies a policy if one exists, or prompts for a change in the design of the system if the violation cannot be handled and the system goal cannot be satisfied. Direction changing or safe landing could be feasible actions for UAV facing with a dangerous situation.

Executor During execution, the action from the `DAP` is enacted on the system by the component executor (E) through effectors (or actuators). This activity receives as input the current conditions of the environment from the monitoring activity, and identifies if a specific state in the adaptation policy is reached. If that is the case, it enacts specific action indicated in the adaptation strategy.

Discussion

Self-adaptation has been growing increasingly important. Though numerous excellent research efforts have been put into this area, self-adaptation as a field is still in its infancy, and existing knowledge and approaches are not adequate enough to address today's ever-expanding and ever-changing various environments. In this paper, we mainly focus on a novel conceptual model to design self-adaptive systems for various and open environments based on attribution theory, offering reusability engineering by decoupling the environment with the system. Accordingly, the related work will be classified into two categories. First, we look into the mechanisms of reusability in adaptive systems, positioning our work. Then, we discuss cross approaches among different disciplines that play an important role in the construction of self-adaptive systems.

Reusability has always been a concern in adaptive systems field. Generally, research has focused on providing frameworks for adaptation, such as rainbow (Garlan, Cheng, & et al., 2004) monitoring the executing system in the system-layer through an abstract model in the architecture layer which interacts with system layer through a translation layer, and Hogna, a platform for deploying self-managing web applications on cloud (Barna, Ghanbari, & et al., 2015), allowing developers to customize each phase of the feedback loop without having to implement the entire layer themselves. Autonomic Software Product Lines (ASPL) is a strategy for developing self-adaptive software systems with systematic reuse by integrating a domain-independent managing system into a domain-specific software product line (Abbas & Nadeem, 2018). Besides, different patterns that can be reused have been proposed facilitating the development of dynamic adaptive systems (Ramirez & Cheng, 2010); other techniques such as bidirectional transformations, a mechanism of synchronization, have been applied to ensure the correctness of reusability in adaptive systems (Colson, Dupuis, & et al., 2016). Though our approach divides the system framework into two levels like most of the reusable approaches, the basis for this division is the attribution to either environment or system itself facilitating the reusability in various deployment environments, not the structure to be reused in different systems with similar functions.

Adaptive system is an interdisciplinary research field. The concept of self-adaptation, derived from biology, is the characteristics of a creature changing its habits to adapt to a new environment (Longman, 1994). Biological approaches in computer science have emerged with the study of collective behavior in natural multi-agent systems by Parunak (Parunak, 1997). Other mechanisms in biology, such as flocking, nest building, molding (Mamei, Menezes, & et al., 2006) and human immune system (Hart, McEwan, & et al., 2011) has been adopted in self-organizing systems and can be transferred to self-adaptive systems. Besides that, it is important to learn and borrow from other fields of knowledge that are working or have been working in the development and study of similar systems, or have already contributed solutions that fit for the purpose of self-adaptive systems. Researches from chemical have been gradually applied. Viroli et al. propose a coordination model for self-organizing systems based on biochemical tuple spaces and chemical reactions (Viroli, Mirko, & et al., 2009). In the physical field, Weyns et al. employ field-based mechanisms for adaptive task assignment in multi-agent systems. Social area concentrates on market and auction mechanisms and as an example, coordination in multi-agent systems is based on social conventions (Salazar, Rodríguez-Aguilar, & et al., 2010). To the end, our approach is inspired by the research findings from psychology, emphasizing that the influence on adaptation behaviors comes from two aspects, the external environment and the internal system. It decouples the system with a specific environment and brings a new perspective in the construction of self-adaptive systems.

In our future research, we plan to further elaborate on the work presented in this paper by applying the method to practical scenarios to strengthen the applicability. In addition, the mapping relations between external factors and internal causes are complicated and changeable due to open and various environments. More efforts would be put into investigating the automatic acquisition of influential adaptation policies, such as machine learning in response to uncertain environmental changes and reinforcement learning method constantly adjusting to new environments.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work is partially sponsored by the National Basic Research Program of China (973) (2015CB352200), and the National Natural Science Foundation of China (61620106007).

References

Abbas, & Nadeem. (2018). *Designing self-adaptive software systems with reuse*. Unpublished doctoral dissertation, Linnaeus University Press.

Barna, C., Ghanbari, H., & et al. (2015). Hogna: A platform for self-adaptive applications in cloud environments. In *10th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems, SEAMS 2015, florence, italy, may 18-19, 2015.*

Cheng, B. H. C., de Lemos, R., & et al. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems [outcome of a dagstuhl seminar]* (pp. 1–26).

Colson, K., Dupuis, R., & et al. (2016). Reusable self-adaptation through bidirectional programming. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems, seams@icse 2016, austin, texas, usa, may 14-22, 2016.*

de Lemos, R., Giese, H., & et al. (2010). Software engineering for self-adaptive systems: A second research roadmap.

In *Software engineering for self-adaptive systems II - international seminar, dagstuhl castle, germany, october 24-29, 2010 revised selected and invited papers* (pp. 1–32).

Dijkstra, & W, E. (1982). On the role of scientific thought. In *Selected writings on computing: a personal perspective* (pp. 60–66). Springer.

Filieri, A., D'Ippolito, N., & et al. (2017). Control strategies for self-adaptive software systems. *TAAS*.

Garlan, D., Cheng, S., & et al. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*.

Hart, E., McEwan, C., & et al. (2011). Advances in artificial immune systems. *Evolutionary Intelligence*.

Heider, F. (1958). *The psychology of interpersonal relations*. New York: Wiley.

Jiao, W., & Sun, Y. (2016). Self-adaptation of multi-agent systems in dynamic environments based on experience exchanges. *Journal of Systems and Software*.

Kelley, H. H. (1967). Attribution theory in social psychology. *Nebraska Symposium on Motivation*, *15*, 192-238.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer*.

Krupitzer, C., Roth, F. M., & et al. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, *17*, 184–206.

Li, N., Jiao, W., & et al. (2018). *2018 IEEE international conference on software quality, reliability and security, QRS 2018, lisbon, portugal, july 16-20, 2018.* IEEE.

Longman, A.-W. (1994). Adaptive control. In *Publishing co., inc. boston, ma, usa.*

Mamei, M., Menezes, R., & et al. (2006). Case studies for self-organization in computer science. *Journal of Systems Architecture*.

Modoni, G. E., Trombetta, A., Veniero, M., Sacco, M., & Mourtzis, D. (2019). An event-driven integrative framework enabling information notification among manufacturing resources. *Int. J. Computer Integrated Manufacturing*, *32*(3), 241–252. Retrieved from https://doi.org/10.1080/0951192X.2019.1571232 doi: 10.1080/0951192X.2019.1571232

Parunak, H. V. D. (1997). "go to the ant": Engineering principles from natural multi-agent systems. *Annals OR*.

Ramirez, A. J., & Cheng, B. H. C. (2010). Design patterns for developing dynamically adaptive systems. In *2010 ICSE workshop on software engineering for adaptive and self-managing systems, SEAMS 2010, cape town, south africa, may 3-4, 2010.*

Salazar, N., Rodríguez-Aguilar, J. A., & et al. (2010). Robust coordination in large convention spaces. *AI Commun.*.

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *TAAS*, *4*(2), 14:1–14:42.

Sawyer, P., Bencomo, N., & et al. (2010). Requirements-aware systems: A research agenda for RE for self-adaptive systems. In *RE 2010, 18th IEEE international requirements engineering conference, sydney, new south wales, australia, september 27 - october 1, 2010* (pp. 95–103).

Shevtsov, S., Berekmeri, M., & et al. (2018). Control-theoretical software adaptation: A systematic literature review. *IEEE Trans. Software Eng.*, *44*(8), 784–810.

Viroli, Mirko, & et al. (2009). Biochemical tuple spaces for self-organising coordination. In *Coordination models and languages.* Springer Berlin Heidelberg.