Towards an Approach for Engineering Complex Systems: Agents and Agility

Massimo Cossentino, Luca Sabatucci
Istituto di Calcolo e Reti ad Alte Prestazioni
CNR, Palermo, Italy
massimo.cossentino, luca.sabatucci@icar.cnr.it

Valeria Seidita
Dip. dell'Innovazione Industriale e Digitale
Università degli Studi di Palermo, Italy
valeria.seidita@unipa.it

***Abstract*—The way in which we use and conceive modern software systems is changing. Humans/users are becoming more and more immersed in today complex systems operation, systems interact in a dynamic fashion with the users and with changing and dynamic environments. New design paradigms are necessary. In this paper we propose a first insight to engineering complex physical systems by employing agility and a framework for self-adaptive service composition.**

I. Introduction

Due to the advancements in computer technologies, we are now drawing near a new era where each of us owns more than one computer (or smart or wearable device and so on). Computers and devices are more and more interconnected and we are getting used to work and live with them in an active fashion.

Our life is shared and in partnership with the technology. This brings to the necessity of communication, collaboration, cooperation and understanding among different devices and among users and devices. The everyday life style is deeply changing and consequently a new way of considering the creation and the interconnection of modern software systems is arising. Indeed, modern computer and devices must be more than simple responders; they have to be able to sense and anticipate users' needs in order to support them in the right way. Instructions we give to software systems are not passive and plan of actions no more decided at design time.

The scenario from now on is: humans work in tandem with their devices (and software systems). At the same time software systems have to interact in a proactive fashion in order to guide and support users on the base of their preferences, habits, constraints and contexts they live.

Such a kind of situations is common, for instance, in the field of smart cities, e-health, e-commerce, etc. and the open challenges are still a lot, from design approaches to standards, from interoperability to trust and so on.

Software systems where users are included and are part of the system require a high level of adaptivity and self-organization. Classic design approaches are not suitable for analyzing, designing and developing software systems supporting users in the cited scenarios where, in a few words, main features are: continuous changing operational context and changing environment. We have to change the way in which we look at software development.

Promising approaches to engineering complex physical systems like these span from Agile approaches to a new current where design time and runtime approach are plunging into each other.

In this paper, we propose the use of the framework MUSA supported by an agile approach that sees users and agents highly immersed in the design process. Agility let realize the user inclusion.

The paper is structured as follows: section II outlines motivations and challenges of the work from a design point of view; section III describes the MUSA framework and how it is engaged in including the user in the design loop; section IV illustrates Agile approaches features and how they may be useful in the complex systems domain; section V focuses on the use of MUSA and agility detailing how the use of MUSA realize agility; finally in section VI some conclusions are drawn.

II. Motivation

Engineering and developing complex physical systems where humans (or users) work together with the system and they both are immersed and continuously interact with a very dynamic environment, requires to consider several factors that may be summarized in: changing operational context and changing environment. Complex software systems supporting digital economy have to intrinsically exhibit adaptation.

A complex (self-adaptive) physical system is a system able to modify its behavior in order to respond or to face changes occurring in the environment it is working on or inside the system itself. Adaptation depends on all the actors that interact with the system, the environment whose changes are affected by and affect the system. The system behavior itself is a source of changes and adaptation. Adaptive behavior is prone to three types of dependency: actor-dependency, system-dependency and environment-dependency. These factors are taken into account during the requirement engineering phase of complex physical (self-adaptive) systems.

Different kinds of approaches for engineering self adaptive systems exist, they span from control theory to service oriented and from agent-based approaches to nature inspired ones. They all map to the four MAPE activities - well known in the field of Autonomic Computing - Monitoring, Analyzing, Planning and Executing.

A promising approach to manage complexity in runtime environments and to implement MAPE activities is to develop adaptation mechanisms that involves software models. This is referred to as *models@run.time* [3]. The idea is to extend the model produced using MDE approaches to runtime environment. The authors of [3] emphasize the importance that software models (artifacts) may play at runtime stating that if ....."a system changes, the representation of the system (the models) should all change, and viceversa". In so doing, researchers in this field stop at artifact levels; they wish artifact produced were tied to the process used for creating them. However at the best of our knowledge this is still a vision, an idea and nothing has been really realized for really putting the design phases at runtime.

Baresi et al. [2] introduce the need of bringing near the design time to the runtime: "The clear separation between development-time and run-time is blurring and may disappear in the future for relevant classes of applications". This allows some changing activities to be shifted from design and development to run time and some changing responsibilities to be assigned to the system itself instead of to the analysts or designers. Thus realizing and really implementing adaptation [1], [6].

Hence, requirements engineering has to deal with requirements that change at run time, for instance as the result of changing in the environment. Uncertainty and incompleteness are at the base of requirements engineering for complex (self-adaptive) physical systems [11], [7]. Some researchers investigated the use of a goal model for specifying behavior and requirements [14], [10] and for supporting the modeling of adaptation mechanism instead of implementing adaptation at run time.

The life cycle of a complex (self-adaptive) physical system, or of one of its components, starts with its design and does not terminate with its deployment and testing. The life-cycle continues with some monitoring phases aiming at identifying and handling new or emergent requirements and/or needs from users.

Classic heavyweight methodologies cannot be used anymore for engineering and developing such a kind of systems in fact, normally, they prescribe a very disciplined process that follows a well specific life-cycle; the main aim is to make the software development as more predictable as possible. In these cases all the requirements have to be identified and analyzed in the very early activities of the design process and transformed into code through whatever life cycle the designer considers more useful (waterfall, iterative and incremental and so on). This way of working has been well established for years for all those systems that do not require particular changes. What we mean is: normally, a software system is the solution to a problem, regardless the level of complexity of the problem and the software, and the level of adaptability to changing requirements is managed at design time using ad-hoc life cycles or process models.

Due to the features of complex (self-adaptive) physical systems and the fact that, nowadays, systems are more interconnected and various than before, designers have not the right means to anticipate and design interactions among different components, and interactions among users and the system. Indeed, (self-adaptive) software system properties are effectively known only when all the relationships among the software components and between the software and the environment have been expressed and have been made explicit. Such issues have to be dealt with at runtime; modeling and monitoring users and the environment is the key for enabling software to be adaptive [11], [7]. Self-adaptation deals with requirements that vary at run time. Therefore it is important that requirements lend themselves to be dynamically observed, i.e., during execution.

We are moving into a new era of research for engineering complex (self-adaptive) physical systems. A new design paradigm is necessary for enabling the design and development of systems that adapt their behavior at runtime with little, or better, no human intervention.

This paper attempts at framing some characteristics related to engineering complex physical systems by merging Agile approach to design at runtime. This is a first experiment to engineering complex systems and exploit the lessons learnt on the use of self-adaptive composition of services in dynamic and distributed environment by means of MUSA (Middleware for User-Driven Service Adaptation).

III. A Middleware for User-driven Self-Adaptation (MUSA)

The Middleware for User-driven Self-Adaptation (MUSA) has arisen for managing evolution and adaptivity of dynamic workflows [19]. Compared to traditional languages (e.g. BPMN and BPEL) MUSA provides a wider space of solution that improve the freedom of action when it is necessary to overcome exceptional events such as failures or resource unavailability. The authors adopted the solution of relaxing the high constrained workflow specification to increase the flexibility in dynamically generating alternative solutions.

The key concept is a clear separation between 'what the system has to address' and 'how it will operate for addressing it'. The enablers of the MUSA vision are: a) representing the two dimensions, *what* and *how*, as a couple of run-time artifacts (respectively goals and capabilities); b) implementing a reasoning system for dynamic binding of capabilities to goals; c) representing goals and capabilities with some formalism, based on a common grounding semantic.

The result is a multi-agent system, implemented in the Jason [4] agent-oriented programming language.

A. What: a Declarative Specification of Goals

The characteristics of being autonomous and proactive make the agents able to explore a solution space, even when this space dynamically changes or contains uncertainty.

Indeed, MUSA accepts, at run-time, requirements as a set of goals to be addressed [21].

The main language for goal injection is GoalSPEC, based on the natural language, and specifically conceived to be attractive

for a business audience. This language allows for describing workflows as a set of the user's goals that are delegated to the management system.

GoalSPEC supports Adaptivity because it is intended to make the business goals explicit in the process. Goals do not specify how to operate, but they rather define the expected results.

GoalSPEC also supports Evolution because business processes change during the time. This could happen because of the introduction of new business goals (laws to be respected or new desired functionalities). Whereas, traditionally, all the revisions imply to check inter-dependencies among process's tasks, with a consequent hard work of maintenance, MUSA accepts run time variations in the set of injected goals.

In the last years, MUSA and GoalSPEC have been used in many different application domains, from document management systems, smart travel planning, to cloud application mashup. The design activity necessary for instantiating the system in a specific domain includes the definition of an ontology/conceptual diagram.

B. How: a Declarative Specification of Capabilities

The concept of capability is a mixture of planning actions [8] and services (or micro-services [16]). MUSA respects this dual nature by separating the abstract capability – a symbolic description of the effect of the action – and the concrete capability – a small, independent, composable unit of computation that produces some concrete result.

The capability supports self-adaptation because it is not specific for a given goal, but it can be reused for addressing several ones. It also is implemented for explicit fault isolation.

Moreover, we focus on the idea that capabilities make it easier to deploy new versions of the software frequently. Indeed, providing capabilities as run-time entities constitute the basis for continuous data exchange between human and agents and therefore system evolution. Moreover, each capability is relatively small, and therefore easier for a developer to implement. It can be deployed independently of other capabilities. By separating the overall functionality through capabilities, it is easier to organize the overall development effort around multiple teams.

C. Self-Configuring a Solution

MUSA provides an automatic reasoner that, at run-time, configures the workflow by associating available capabilities to the injected goals.

The approach is possible because both goals and capabilities are first-class entities to be used within agent deliberation. An agent requiring to address an unanticipated goal must decide which capability (or combination of capabilities) have to execute.

The automatic reasoner selects and associates capabilities according to the goals that are injected [18]. The basic idea is that of exploring a space of solutions, where goals represent points of the space that must be reached, and the abstract capabilities provide evolution functions that allow moving through this space. Therefore self-configuration is defined as a space search problem.

Actually, the automatic reasoner is not a property of a single agent, but rather it is a social ability of all the agents of the system. Indeed, the algorithm is distributed and decentralized (for details see [20]).

D. The Self-Adaptation property

The main advantage of using MUSA for executing workflows is the ability to self-adapt the running instance when something unexpected happens. The self-adaptation property is based on the feedback loop [5].

In MUSA, the feedback loop is composed of two nested loops. The inner loop is a traditional MAPE-K loop [7] where the activities are: *Monitor*, *Analyze*, *Plan*, and *Act*. This loop is responsible for managing the periods of stability of the system. During this time the system tries to run a specific workflow configuration. Local task replacement may be enacted for ensuring the continuity of service.

However, it is possible that some severe malfunctioning occurs and that the monitor reveals a violation of requirements forces the system to stop working. The system becomes unstable and some fixing is necessary. These events are captured by the outer adaptation loop. In this case the system blocks any capability concerning the current execution and executes a new self-configuration phase. The result may be a totally new workflow instance to be executed for replacing the previous one.

The outer loop also monitors the goal injection, retrieving cases in which the running workflow is obsolete with respect to the new injected goals. Also in these case the system's objective is that of switching from a workflow configuration to another one that is able of addressing the new set of goals.

IV. Agility and Complex (self-adaptive) Systems

Agile is not strictly a methodology, it is more a way of conceiving working in teams. Agile movements were born and are operating for exploring new alternatives to traditional software development. Proponents of Agile approaches think that software development process can adapt to changing and dynamic operational conditions, imposed by variable and unpredictable environment, by putting emphasis to actual working code [13]. In this way Agile allows people to act in response to uncertainty and unpredictability through iterations, increments and feedbacks.

Agile is an efficacious alternative to traditional software sequential development. Agile development paradigm is well suited to the design of systems where requirements continuously change. Requirements changing is, in the case of the system we are considering here, due to uncertain and dynamic development environments resulting from: evolving technologies, changing customer requirements and other changes related to the analysis and design phase. All the existing Agile approaches face changes with incremental and small software releases, with highly adaptation to sudden or

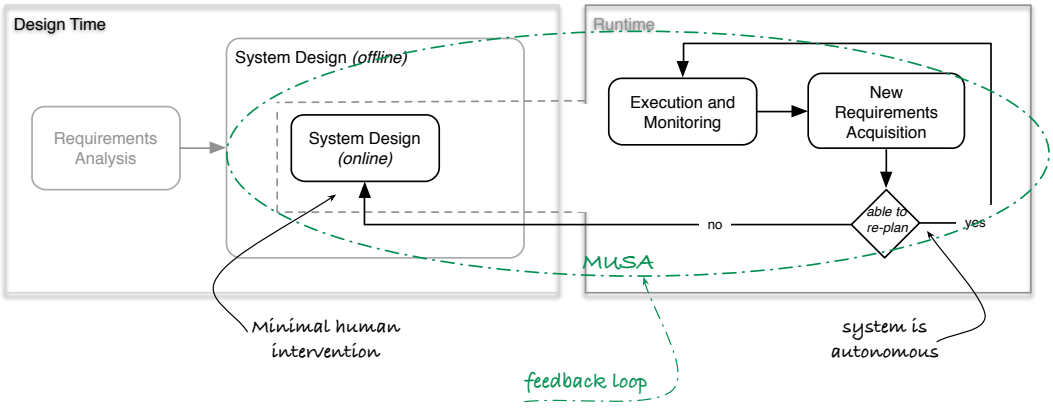

Fig. 1. Designing complex self-adaptive systems employing MUSA and Agility

last moment changes and with the production of the amount of documentation necessary only for learning and modifying.

So how and why talking about agility in the context of complex (self-adaptive) physical systems and agents? In [9] the author states that lightweight methodologies, including XP and Agile, "include some concepts and principles of natural complex systems". The main principles of natural complex systems (see [17] for a literature review) are:

- open systems and inner interactions - complex adaptive physical systems are open systems exchanging information in open and dynamic environments and components of the systems interact dynamically each other. Interactions affect the overall behavior of the system.
- feedback loop and emergent behavior - changes in some parts or components of the system result in changes in other parts or in the whole system.
- distributed control - control is distributed through the system.
- emergent behavior - features of the system and its behavior in its whole cannot be described and understood only looking at the single components, a holistic vision is necessary, interactions among components let the behavior arise.

These principles perfectly fit agile practices, the most important one is that Agile greatly focuses attention on the continuous open interaction among all the involved stakeholder of the system. So, our hypothesis is: complex physical system involves software components and users both at run time and design time. If we consider the complex adaptive physical system and its design process as it were a whole, as if one were an integral part of the other then we may apply the Agile principles. Agile principles assure us to use one of the best approaches for engineering self-adaptation [12], [15]; agents, in some moments of the software development, are "agilists". Users are included in the design process with the support of the multi-agent system from which MUSA is made. In the following we give an overview of the MUSA framework in order to let understand how it (by means of agents) support in the developing and enacting a software application and to bring to the illustration on how its use is agile.

V. Designing Complex Self-Adaptive Systems Employing MUSA and Agility

Using the MUSA framework allows to include the user in the design process by employing the following agents: discovery agent, specification manager, negotiator, worker e case manager; moreover it partially realizes the design time at run time [18].

Fig. 1 is intended to explain which are the main activities to be performed when developing an application with MUSA. The process is divided in two parts, the first one (the left block in the figure) includes the standard design activities (Requirements Analysis and System Design) for developing a first release of an application. During the very first step, the system design activity is performed "offline", hence requirements are captured and analyzed in a quite common and standard fashion and then they are converted in the design of the multi-agent system underpinning MUSA. During the first execution at runtime (the first specific workflow configuration) the system starts to monitor the environment and each time a change occurs, a new requirement or a need, then the system tries to re-plan. If re-planning is possible then the system goes on in its work and continues the executing and monitoring phase otherwise the system searches for a new solution. An "online" system design is performed, users and agents in MUSA collaborate in order to find a new solution at run time as explained in the previous section. The design phase is immersed in the run time one, agents and users work in team. During the "online" System Design the presence of agents lets users apply a minimal intervention thus enhancing adaptation. Moreover, the team formed by agents and users, during the implementation of the feedback loop, apply an agile approach that, as said in section IV, gives a means and is one of the most promising approaches for designing and developing complex (self-adaptive) systems.

In the following we analyze the Manifesto for Agile Software Development[1] with respect to the rationale underpinning

[1]http://agilemanifesto.org

TABLE I
MUSA FEATURES SUPPORTING AGILITY

Agile Manifesto	**MUSA general Features**
Early and continuous delivery of valuable software.	The Proactive Means-End-Reasoning for producing plans allows to create soon a working software and to continually update it by (re)starting from a set of *goals* and *capabilities*. (See also the following principle).
Deliver working software frequently with a preference to the shorter timescale.	Once a new set of goals is specified, MUSA, by means of the injection technique, allows to produce new portions of working software. (See also the previous principle).
Business people and developers must work together daily throughout the project.	Regardless of the role played by agents or humans in the MUSA loop, the use of *goals* allows to specify the business logic of how services have to be composed thus realizing the continuous collaboration among business people and developers.
Give projects the environment and support they need, and trust the job will be done.	Capabilities and goal specifications are compliant with the conceptualization of the environment.
Working software is the primary measure of progress.	MUSA, by means of the *solution explorer* agent, checks in the space of solutions if the new set of goal has been reached.
Agile processes promote sustainable (all the involved stakeholder maintain a constant pace) development. Simplicity–the art of maximizing the amount of work not done–is essential.	The use of MUSA as a middle layer, for realizing design process at runtime, allows of some part the process to be made automatic thus reducing the amount of work to be done and documentation to be produced.
The best architectures, requirements, and designs emerge from self-organizing teams.	The core of MUSA collaboration issues is guaranteed by the holonic architecture. Holons allow self organization among, distributed coordination and knowledge sharing.
Frequently tuning and adjusting team's behavior for becoming more effective.	With regard to the team composed both by agents, holons continuously know the state of the environment and so they are able to reach all the new goals by organizing and self adapting their behavior.

the use of MUSA:

Individuals and interactions over process and tools
MUSA interacts with its environment. There is no need to follow a rigid design process and the presence of MUSA guarantees that the user is the center of design/runtime as well as interaction between users and software.
Working software over comprehensive documentation
Using MUSA does not imply to produce documentation; the only useful documentation is the one for describing the environment through the ontology and for the newly introduced capabilities and goals.
Customer collaboration over contract negotiation
Considering the runtime phase as part of the design process, the user and the customer are at the center of MUSA activities; the customer collaboration starts with the definition of the requirements and continues during the execution with advising new goals. Negotiation happens at run time with a pay per use criterion, it is automatic and has a minor importance with respect to collaboration.
Responding to change over following a plan
No plans are pre-defined but the system is made able to configure itself at runtime each time a change occurs.

In Table I, we show how the principles behind the Agile Manifesto[2] are met by using MUSA, both during the self-adaptive system development and its running. In the left column the principles of Agile Manifest are summarized, the right column shows which MUSA's feature helps in putting into practice agility.

[2]http://agilemanifesto.org/principles.html

VI. CONCLUSION

Actually, complex physical (self-adaptive) systems are required to be much more than simply an ensemble of interacting components; components are not only intertwined but they form a unique ensemble of persons, technologies, organizations and environmental contexts. Environment, moreover, could involve laws and rules regulating the social interactions and the interaction of people with the technology. Humans are immersed together with devices in a highly dynamic environment where a great amount of data and services are constantly available. It is unthinkable that a human user were able to act in such a kind of environment and to consciously handle all the offered possibilities.

Standard analysis and design of systems with these extended features are more challenging than ever, mainly because the behavior of system components depends and have to be specified in terms of their interactions and communications with the physical environment. Moreover, interactions and communications are affected by environment dynamics at runtime.

Inspired by models@runtime [3] research line and by Baresi et al. [2] that suggest to reduce the boundary gap between design time and run time, we propose the use of MUSA along with some Agile design activities that see users and agents highly included and immersed in the design process. Agility allows to realize user inclusion and to provide a means for establishing a design approach that adapt and evolve at run time while new requirements arise from the continuous interaction among users and dynamic environment.

What we illustrate here is a first experiment towards engineering complex (self-adaptive) physical systems; it was

helpfulfor gaining useful insights on design activities in order to complete the development of a design methodology for complex systems we are working on.

REFERENCES

[1] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. Software engineering processes for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 51–75. Springer, 2013.

[2] Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 17–22. ACM, 2010.

[3] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. *Computer*, 42(10):22–27, Oct 2009.

[4] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.

[5] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[6] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[7] Betty HC Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, pages 1–26. Springer, 2009.

[8] Michael Gelfond and Vladimir Lifschitz. Action languages. *Computer and Information Science*, 3(16), 1998.

[9] M Gerber. Keynote speech: Lightweight methods and their foundations in chaos theory. In *6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland*, 2002.

[10] Heather J Goldsby, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Danny Hughes. Goal-based modeling of dynamically adaptive system requirements. In *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pages 36–45. IEEE, 2008.

[11] Paola Inverardi. Software of the future is the future of software? In *International Symposium on Trustworthy Global Computing*, pages 69–85. Springer, 2006.

[12] Radhika Jain and Peter Meso. Theory of complex adaptive systems and agile software development. *AMCIS 2004 Proceedings*, page 197, 2004.

[13] Philippe Kruchten. Agility with the rup. *Cutter IT journal*, 14(12):27–33, 2001.

[14] Sotirios Liaskos, Alexei Lapouchnian, Yiqiao Wang, Yijun Yu, and Steve Easterbrook. Configuring common personal software: a requirements-driven approach. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 9–18. IEEE, 2005.

[15] Peter Meso and Radhika Jain. Agile software development: Adaptive systems principles and best practices. *Information Systems Management*, 23(3):19–30, 2006.

[16] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.

[17] Mark EJ Newman. Complex systems: A survey. *arXiv preprint arXiv:1112.1440*, 2011.

[18] Luca Sabatucci and Massimo Cossentino. From Means-End Analysis to Proactive Means-End Reasoning. In *Proceedings of 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 18-19 2015, Florence, Italy.

[19] Luca Sabatucci, Carmelo Lodato, Salvatore Lopes, and Massimo Cossentino. Towards self-adaptation and evolution in business process. In *AIBP@ AI* IA*, pages 1–10. Citeseer, 2013.

[20] Luca Sabatucci, Salvatore Lopes, and Massimo Cossentino. Self-configuring cloud application mashup with goals and capabilities. *Cluster Computing - The Journal of Networks Software Tools and Applications*, 2017.

[21] Luca Sabatucci, Patrizia Ribino, Carmelo Lodato, Salvatore Lopes, and Massimo Cossentino. Goalspec: A goal specification language supporting adaptivity and evolution. In *International Workshop on Engineering Multi-Agent Systems*, pages 235–254. Springer, 2013.