Dealing with Complexity in Agent-Oriented Software Engineering: The Importance of Interactions

Joaquin Peña, Renato Levy, Mike Hinchey, and Antonio Ruiz-Cortés

1 Introduction

Complexity has been one of the main problems that science and industry has dealt with from the beginning of the industrial world. It prevents us from understanding and controlling reality, and as a result, significant effort of many scientists and practitioners have been expended on conquering it, with the aim of finally, understanding and controlling our world.

Complexity has been studied by researchers in many fields, ranging from the Social Sciences to Physics, and of course, Software Engineering. But in all of these fields, researchers agree that complexity is caused by the interaction between the parts that conform a system (see Sect. 2 for further details). Complex systems expose a behavior that cannot be predicted since it is the consequence of a long chain of cause-effects (interactions) where a small change in a component of the systems affects many others, thus amplifying its effect. This results in an overall behavior of the system—also called macro-level behavior—which cannot be explained by the behavior exposed by each of its component parts, its micro-level behavior. For example, an ant colony is able to feed and protect itself forming a sophisticated social system while the behavior of individual ants remains quite simple, most of the time consisting just of interacting with other ants by following trails of pheromones left by other individuals.

If these ants didn't interact by means of pheromones, the emergent behavior would not be possible, and thus, we would have removed complexity. So, in order to address complexity full-on, we must focus our efforts on studying the interactions, to subsequently understand how the composition of these interactions brings us to the macro-level behavior.

However complexity does not always appear at the same level. There are systems with higher levels of complexity and systems with lower ones, ranging from com-

J. Peña (✉)
University of Seville, Seville, Spain
e-mail: joaquinp@us.es

plicated systems, to complex systems, to chaotic systems (see Sect. 4 for further details). The level of complexity of a system, however, is not a property derived from the structure and behavior of the system and its parts, but it depends significantly on the tools that we use to study them. For instance, the *computational complexity* for many algorithms is smaller when using a multi-tape or inductive Turing machines than when using a Turing machines with a single tape.

The use of appropriate software engineering tools can open the possibility of understanding and controlling systems that are seen as complex with current tools, but can be designed as merely complicated with the proper ones [27].

Among these tools, the first must be to focus modelling efforts on the main source of complexity—interactions—and not on structural properties as is commonly done in the Software Engineering community, where UML class diagrams or component diagrams are the most commonly used modelling tools.

Hence, we must pay special attention to interactions when developing software systems, but especially so when addressing Multi-Agent Systems (MASs). This focus is possible based on the premise, accepted by researchers of human organizations [23], and later by researchers in the field of agent technology, that an organization can be observed from two different viewpoints: functional/interaction and structural. Roughly speaking, the model of the functional organization is composed of roles and interactions while the model of the structural organization is composed of agents and interactions. Despite their close relationship, both types of organization views can be modelled independently. This fact allows designers to model the interaction process while ignoring the organizational structure until it is clearly understood how it operates. This modelling process reduces the complexity of models to be managed at the early stages of the software process and eases the comprehension of complex behaviors (see Sect. 3 for further details on modelling in terms of interactions).

Once we can address the problem in terms of interactions, we find new problems. For example, the number of interactions to be designed may be huge, and the combination of them required until reaching the required macro-level behavior difficult. To address this, in Sects. 5, 6, 7, and 8 we present the main principles we can apply in this situation: abstraction, decomposition, composition, reuse, and automa-tion. All of these can be applied to deal with complexity, but we also need guidelines that help us to apply them systematically depending on the kind of system we face. These guidelines and the software process that can be followed to apply these principles systematically are presented in Sect. 9. In general, two approaches can be used: a top-down software process where we start at the desired macro-level and we systematically refine models by applying abstraction and decomposition principles; and bottom-up, where we start at the micro-level and we apply systematically composition and abstraction until we reach the desired macro-level behavior.

In addition, in order to show how to apply each principle and how they must be combined to systematically engineer a complex MAS, we employ a case study on what has traditionally been seen as a complex system: an Ant Colony. As a result, we end up showing how this system can be modelled, linking the macro-level behavior with the micro-level behavior systematically from a software engineering point-of-view.

2 Related Work on Focusing on Interactions as the Source of Complexity

Several authors agree that the complexity of MASs is a consequence of their interactions [17, 24]: *Complexity is caused by the collective behavior of many basic interacting agents.*

In fact, many authors point out that the complexity of MASs is the consequence of those interactions among agents, and that these interactions can vary at execution time, and cannot be predicted thoroughly at design time, viz., emergent behavior. The reasons for the emergence can be traced to two features present in MASs: self-adaptation, and self-organization [13, pp. 20–21], [17, 24]. It is important to observe that this capability of demonstrating emergent behavior is the key factor that drove us to implement MAS solutions in the first place, since this key capability is essential to address solutions to the targeted domains.

The importance of interactions has been already established in the agent literature as well as in several other fields. In the field of software engineering many authors have seen interactions as a source of complexity, and thus, many solutions has been proposed for dealing with it (see also Chaps. 1, 7, 8, and 15). For example, Larman et al. in [21] presents a set of principles proposed by other authors, from which, many of them focus on reducing the coupling, that is to say, the interactions, between different part of the system, namely Demeter's Law, Liskov's Principle, GRASPs Low Coupling, Indirection, Protected Variations, etc.

In addition, some advanced Object-Oriented Software Engineering approaches, e.g., [32, 35], even traditional sociology, already present a predominant role regarding structural features to interactions, to such a point that all the modelling process is focused on them.

OOram [32] is a good example of an Object-Oriented approach where the whole development cycle is focused on interactions. OOram's authors state that the main advantages of focusing on interactions is the improvement of reuse, traceability and the ability to cope with complexity [31].

Furthermore, in sociology, interactions have been also emphasized by important authors such as the German sociologist Max Weber. Weber in his concept of *ideal bureaucracy* emphasizes the form, or in other words, the interrelationships between the members of an organization. In 1988, Reenskaug, the author of OOram, stated in [31] that object-orientation was born at the hand of Weber. In this reference, Reenskaug concluded that object-oriented methodologies must focus on interactions.

In addition, this fact is also ratified by the research done in other mature fields: (i) in the component world, Szyperski and D'Souza also emphasized the importance of focusing on interactions instead of architecture (structure in MAS) in complex systems [34, p. 124], [6]; (ii) in the distributed field, several authors has also favored approaches that focus on interactions, i.e., Francez who highlight the importance of modelling complex interactions as a singleton and who also work on functional groups of interacting elements [12]; (iii) the latest version of UML also provides modelling artifacts to perform interaction-centered modelling, emphasizing and improving the role concept compared to previous versions.

3 The Main Tool for Dealing with Complexity: Modelling the Problem in Terms of Interactions

Given that interactions are the main source of complexity, we conclude that [27]:

> if we want to conquer complexity of MASs, we must focus the modelling process on them

In observing any MAS, we can say that no agent is an island, and thus, every MAS has the potential to become a "complex system". Since agents are limited to some environment and have limited abilities, complex problems are usually solved by a set of agents [4]. Hence, an organization represents a group of agents formed in the system in order to get benefits from one to another in a collaborative or competitive manner.

Therefore, a sub-organization emerges only when some kind of interaction between its participants exists, either through direct communication by means of speech acts or through the environment. The structure of an organization is underlined by the nature of their interactions; hence it is vital to clearly understand the interactions within a MAS system in order to determine its sub-organizations.

The Organization of the Agents in a MAS can be observed from two different points of views [3, 8, 36]:

The interaction point of view: it describes the organization by the set of interactions between its roles. The interaction view corresponds to the functional point of view.
The structural point of view: it describes the agents of the system and how they are distributed into sub-organizations, groups, and teams. In this view, agents are also organized into hierarchical structures showing the social architecture of the system.

The former is called the *Acquaintance Organization*, and the later is called the *Structural Organization*. Both views are intimately related, but they show the organization from radically different points of view.

Since any structural organization must include interactions between its agents in order to function, it is safe to say that the acquaintance organization is always contained in the structural organization. Therefore, if we determine first the acquaintance organization, and we define the constraints required for the structural organization, a natural map is formed between the acquaintance organization and the corresponding structural organization. This is the process of assigning roles to agents [36]. Thus, we can conclude that any acquaintance organization can be modelled orthogonally to its structural organization [20].

In Fig. 1, we present a simple version of the manufacturing pipeline example presented in [36, p. 10]. In this example, each stage is performed by an agent and the main requirement is that the speed of all stages is coordinated. A set of roles and interactions between them is implied in the Acquaintance Organization. In the structural organization, these roles can be structured to form several organizational structures. For example, as shown in Fig. 1, we can map the acquaintance organization into a plain structure, a hierarchical structure, and so on. In addition, starting the analysis with a certain organizational structure in mind (by means of agents), even if based on a real organization, will drive the deployment of the MAS. Consequently, the initial subdivision in interactions and roles may not be optimal.

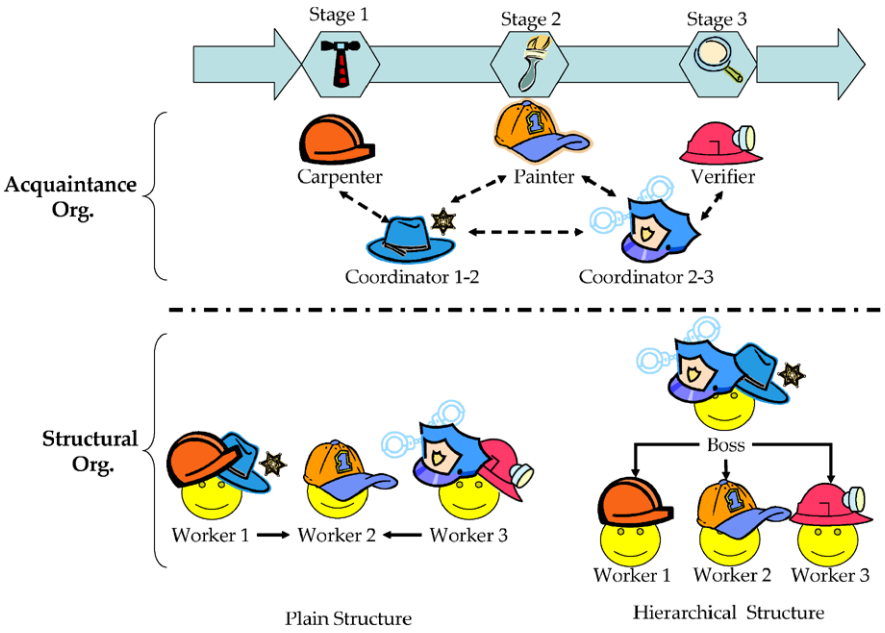

Fig. 1 Acquaintance vs. structural organization

Real life organizations are known to present less then optimal structures. The presence of such organizational mistakes has been well studied in economics [23], hence the field of operational research. Using the real life organization as the initial drive for the MAS system without further consideration will mimic its mistakes and may lead to some important misconstructions in terms of agent systems. Some of the common errors that can be induced are: agents coordinated by more than one agent, agents introduced to cover the relations between several sub-organizations, redundant agents with the same profile placed in different sub-organizations, etc.

As we show later, since interactions are the main source of complexity, we should not bother about organization structure at the initial analysis. This approach facilitates the process of understanding the complex behavior of a MAS and minimizes structural mistakes. Thus, when we consider the relationship between real organizations and their constraints in the system architecture, we must abstract the organization and let it be modelled by means of roles and interactions during the analysis phase. Later, these roles can be mapped into concrete agents and structured as the real organization trying to fit the real life organization and trying to minimize structural mistakes.

Interactions and role-to-role relationships are therefore the primary concept of the engineering process of MASs and structural organization arises because of them.

To exemplify these concepts and the tools for conquering complexity, we are going to use a typical case study of an emergent system: the ant colony. An ant colony is defined as a huge number of autonomous agents defined independently. There are

many definitions of ant colonies. We have selected the one given by StarLogo.[1] The ants in this implementation follow the set of rules below [24]:

1. Wander randomly.
2. If food is found, take a piece back to the colony and leave a trail of pheromones that evaporates over time; then go back to rule 1.
3. If a pheromone trail is found, follow it to the food and then go to rule 2.

In the emergent behavior that appears at the macro-level of an ant colony, the interactions between ants are the key concept. Notice that if ants move randomly without interacting among each other, no emergent behavior appears. Given this fact, if we were able to put together all ants in a colony and compose them to find out their interdependencies, we could provide a macro-level model of the colony.

To perform this model, where interactions are the main feature of interest, we must focus on the acquaintance organization. Ants are designed to pursue two goals: "search for food" and "carry food home". Considering both functional requirements, two kinds of ants appear, that is to say, ants playing two different roles: *explorer* and *carrier*. An *explorer* behaves as follows:

1. Wanders randomly.
2. If a pheromone trail is found, follows it, and go to rule 4.
3. If food is found directly, go to rule 4.
4. Becomes a *carrier* ant.

A *carrier* should behave as follows:

1. Takes a piece of food back to the colony.
2. Leaves a trail of pheromones that evaporates over time.
3. Becomes an *explorer* ant.

In addition, we can find two more roles representing the environment. In the ants' environment we can find the *ground* and the *anthill* as significant from the interaction point of view. Hence, we can divide the environment into two roles, one for each of them. We can observe that ants interact with these roles by means of pheromone trails, which are used to communicate the food position.

4 Characterizing Complexity

Although in Sect. 2, we show that interactions are seen as the main source of complexity in MASs, a large MAS is usually composed of many parts which do not present the same features. Some parts of a MAS could be fully predictable not presenting any emergent feature, while some other parts of the same MAS could be highly complex presenting a high-degree of self-adaptation and self-organization. In the field of enterprise organization, Snowden and Kurtz recognize this fact [33].

[1] The StarLogo definition is available here: http://education.mit.edu/starlogo/samples/ants.htm.

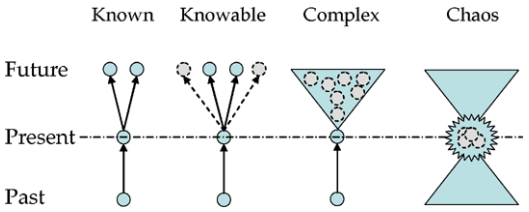

Fig. 2 Complexity and predictability

These authors divide an organization into the following domains whose main fea-tures are summarized in Fig. 2:

(1) *Ordered Domain*: Stable cause and effect relationships exist. In this domain, the sequence of events/actions of the organization can be established as a cause/effect chain. It represents the predictable part of the system. This domain is further divided into:
   (i) *Known Domain*: In this domain, every relationship between cause and effect is known. The part of a MAS in this domain is clearly predictable and can be easily modelled.
   (ii) *Knowable Domain*: This is the domain in which, while stable cause and effect relationships exist, they may not be fully known. In general, relation-ships are separated over time and space in chains that are difficult to fully understand. The key issue is whether or not we can afford the time and re-sources to move from the knowable to the known domain. In Fig. 2 this is represented by a higher number of future directions given a certain present state.

(2) *Un-ordered*: This domain presents unstable cause and effect relationships between interactions in the system. It represents the unpredictable part of the system. This domain is divided into:
   (i) *Complex Domain*: There are cause and effect relationships between the agents, but both the number of agents and the number of relationships defy categorization or analytic techniques. Relationships between cause and effect exist but they are not predictable. This domain presents retrospective coherence. That is to say, coherence can be only established by analyzing the past history of the system. Unfortunately, future directions, although coherent, cannot be predicted. In Fig. 2, the past events/actions can be understood as a single chain of cause/effects, but when we try to extrapolate and predict future changes the solution space is too wide to be analyzed.
   (ii) *Chaos Domain*: There are no perceivable relationships between cause and effect, and the system is turbulent; we do not have the response time to investigate change. Despite some previous work in this area, chaotic domains are still out of reach from the point of control theory. Agents systems have been used to model such domains, but strictly limited to simulation.

The Santa Fe Institute[2] define complexity as "*the condition of the universe that is integrated and yet too rich and varied for us to understand in simple common ways.*

[2]The Santa Fe Institute's webpage is here: http://www.santafe.edu/.

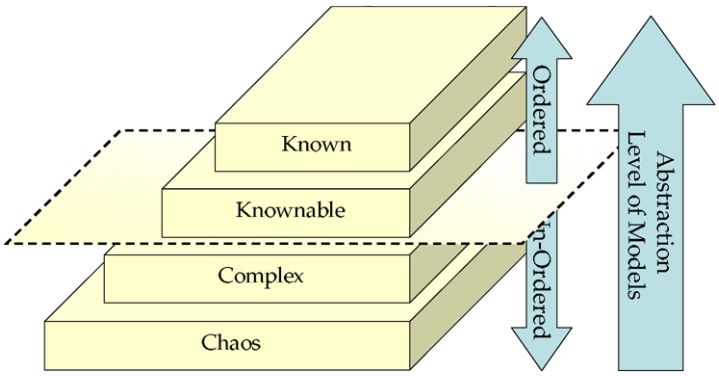

Fig. 3 Domain of a problem depending on the abstraction level of models

We can understand many parts of the universe in these ways, but the larger more intricately related phenomena can only be understood by principles and patterns; not in detail."

As the previous fact shows, problems in the complex or chaos domains can be only understood by principles and patterns that summarize their features and omit details; that is to say, that use abstract models. We do not have to know all the details of a problem, but the level of detail needed depends on our purpose. For example, the weather report can predict the temperatures, rain, and so forth, accurately enough for our daily life purposes: for example, to decide whether to pick up the umbrella or not. The model used to predict the weather can be classified in the known domain, but not the weather itself, which is so far chaotic. Depending on our purpose when studying a certain problem, we may need more or fewer details.

Consequently, we can introduce another dimension in the categorization of complexity done in the Cynefin framework: the level of abstraction of models as we have depicted in Fig. 3 [33]. Thus, depending on the level of abstraction with which we observe a MAS, each subpart of the model can be categorized in the known domain, using the highest level of abstraction, or even in the chaos domain, using the lowest level of abstraction.

4.1 Characterization of Interaction Complexity

Similarly, the complexity level of an interaction depends on the level of abstraction in which its features regarding emergence are observed. This principle can be visualized by the interaction categorization shown in Fig. 4.

The complexity of an interaction, or set of interactions, depends on their nature and on the effort taken in understanding its details, such as, their predictability and flexibility, and their level of abstraction. Our proposed interaction categorization is based in the space defined by these two axes. Figure 4 shows the classification of interactions in three categories: known, knowable, and complex interactions.

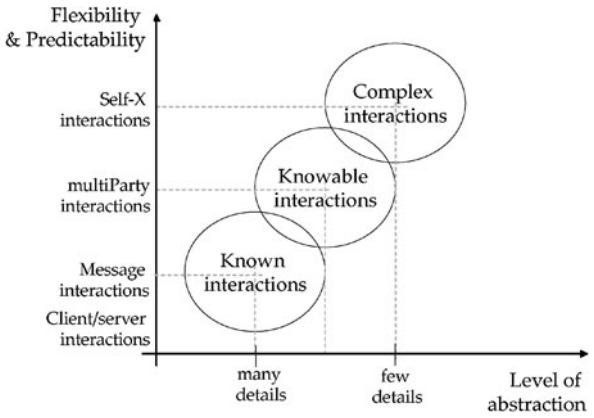

Fig. 4 Proposed interactions taxonomy regarding complexity

Known interactions are the least flexible; they do not present emergence, and all their details can be identified. Complex interactions present a higher degree of flexibility and can only be described with higher-level patterns emphasizing most important details. Knowable interactions represent a middle point between both of them.

In addition, agents undertaking complex interactions may present a high degree of autonomy, proactivity, reactivity, and social abilities. The further a subpart of an agent system moves from known into complex interactions the further its abilities, as described above, are intensified. We must observe that the need to describe (and generate) complex behavior from simpler constructs was the reason that drove us to agent based systems in the first place; therefore our goal, must be to describe the system as it is perceived (complex), and increase details until the desired behaviors can be synthesized.

5 Principles to Deal with Complexity

In [17], Jennings adapts to agency the three main principles for managing complexity proposed by Booch in the OO context [1]: Abstraction, Decomposition and Organization/Hierarchy[3]:

- *Abstraction*: is based on defining simplified models of the systems that emphasize some details while avoiding others. It is interesting since it limits the designer's scope of interest and the attention can be focused on the most important details at a given time.
- *Decomposition*: is based on the principle of "divide and conquer". It helps to limit the designer's scope to a portion of the problem.

[3]Notice that hereafter we call it *Composition* in order to differentiate it from the organization term in AOSE.

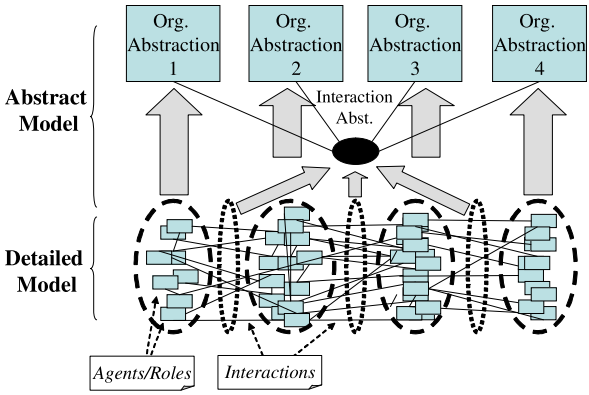

Fig. 5 Abstraction principle

- *Composition*: consists of identifying and managing the inter-relationships between the various subsystems in the problem. It makes it possible to group together various basic agents or organizations and treat them as higher-level units of analysis. It also provides means of describing the high-level relationships between several units.

In addition, automation and reuse have been presented as two important principles to overcome complexity [6, 19]:

- *Automation*: Automating the modelling process results in lower complexity of models and reduces effort and errors. Some procedures must definitely be carried out based on the judgment of the human modeller. However, some steps can be performed using automatic techniques to transform models which can be carried out by a software tool.
- *Reuse*: Reuse is based on using previous knowledge in designing MASs. It saves modellers from redesigning some parts of the system and avoids errors, thus achieving lower complexity of models. Reuse involves processes, modelling artifacts, techniques, guidelines, and models of previous projects.

However, these authors do not focus on managing the main source of complexity, as we do in this chapter. In the following, we detail each of the previous principles.

6 Abstraction

Abstraction consists of defining simplified models of the systems that emphasize some details, while avoiding others. The power of abstraction comes from limiting the designer's scope of interest, allowing the attention to be focused on the most important details. Abstraction can be applied to interactions that fall in the complex and knowable domains, enabling us to abstract from how emergence can be obtained until the designer is ready to address the issue.

As depicted in Fig. 5, we can apply abstraction to produce a simple model of a complex acquaintance organization where most relevant interaction patterns

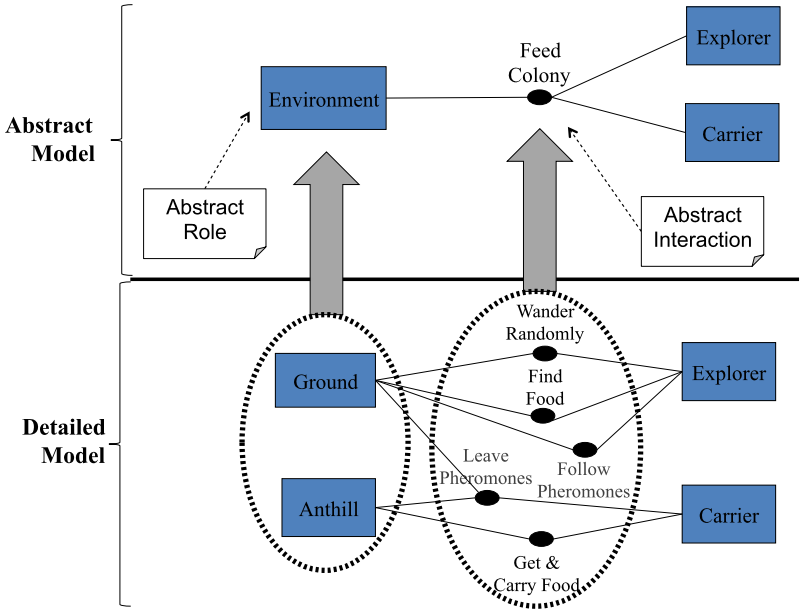

Fig. 6 Abstract model of the ant colony

and members of the organization can be abstracted until we bring the model to the known domain. In this model, most relevant patterns of interaction are abstractedly represented, while less important relationships, or even internal details of relevant interaction patterns, are omitted. Abstract interaction patterns, i.e., complex interactions, hide flexibility and emergence of interactions, which take place at lower levels of abstraction.

Consequently, there are two main modelling artifacts, abstractions that include the tools to perform simplified models [17, 19]: organization and interaction abstractions.

First, *organizational abstractions* represent how a system goal or several of them are achieved by a group of roles/agents. Many authors have worked on recursive definitions of agents and organizations, e.g., [2, 7, 9, 14–16, 26].

Secondly, but more importantly, *interaction abstractions* represent a set of interactions between any number of agents/roles. Many authors have proposed these abstractions, e.g., the protocols of Gaia [36], the interactions of MESSAGE [3], or joint intentions in the Belief-Desire-Joint-Intention Architecture [18].

If we consider our case study, the ant colony, we can derive a very abstract model, shown in Fig. 6. As shown in the figure, we provide an abstract model where we only consider the roles used by ants to interact, and just one interaction that abstracts all the relationships that takes part between the ground, the anthill, the explorers and the carriers. Given this model, we can observe just the amount of food available in the environment, the probability of finding food depending on the size of the ground, and the mean speed to find and carry food, all of them attributes of the roles involved,

to provide a model that ensures that our system operates within the requirements of the system at the macro-level.

Notice that this model is simpler than the one performed based on the structural organization where we should have modelled every ant in the colony.

7 Composition and Decomposition

Composition and Decomposition help us to merge or separate interactions and models in order to focus just on a part of the system in order to study it in isolation. In addition, when abstraction is applied to interactions, key for dealing with complexity, these principles help us also to decrease or increase the level of abstraction by dividing an interaction into several or by grouping several interactions into one. These tools are crucial for transiting from complex to knowable or known interactions, and thus understanding complexity.

7.1 Decomposition

Excessively large problems may become unmanageable. The decomposition principle helps us to divide large problems and their elements into smaller, more manageable chunks. Decomposition consists of the "divide and conquer" principle, helping to limit the designer's scope to a portion of the problem. Regarding interactions, it may help to decompose complex and knowable interactions into finer grain interactions. These finer grain interactions can be augmented with details, which cannot be applied when more abstract interactions are managed. Hence, using decomposition, the interactions obtained can be implemented with less effort.

Decomposition techniques can be applied to the main abstractions: interactions or organizational models, based on roles, organizations, or agents. On the one hand, as depicted in Fig. 7, interaction abstraction can be decomposed to observe them from a lower level of abstraction. The main approach to decompose interactions consists of providing an abstract modelling artifact that can be refined by means of finer grain interaction abstraction, or modelling artifacts designed to provide lower levels of details, such as AUML sequence diagrams where abstract interactions are decomposed into messages-based models [25].

On the other hand, an organizational model that becomes too large and complex can be also decomposed into several models. This allows each sub-problem to be studied in isolation, ignoring the complexity derived from the interactions between sub-problems. Notice also that agents can be indeed decomposed. The materialization of decomposition of agents can be found in the "Role" concept. As depicted in Fig. 7, when a complex organization, formed by agents, is decomposed to extract some functional aspect, their agents must also be decomposed to extract only the part that is related with the functionality we desire to observe. Each of these

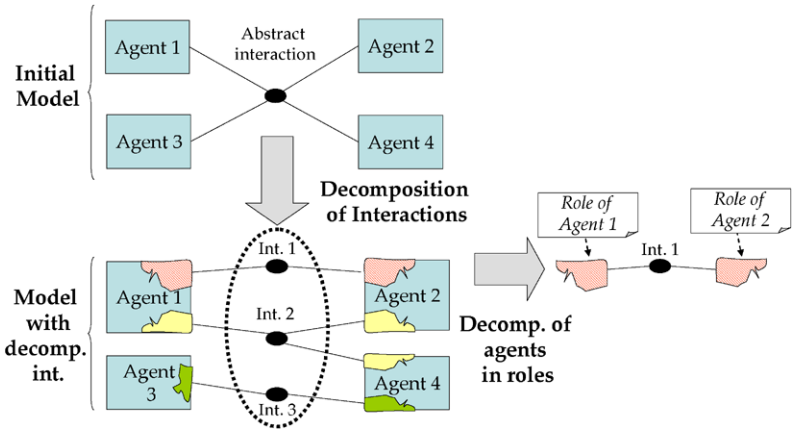

Fig. 7 Decomposition principle

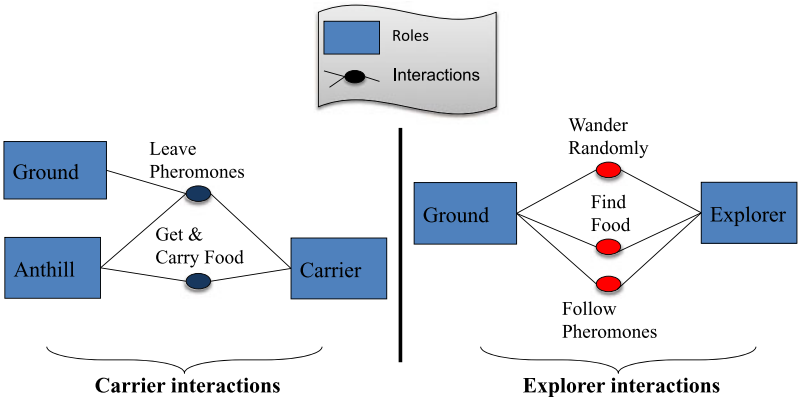

Fig. 8 Decomposed models of the ants colony

parts represents the role that the agent plays in the achievement of that functionality, permitting us to observe the acquaintance organization of the system.

For example, we can derive several models of our case study in order to study each of them separately. As shown in Fig. 8, we provide two models: one, on the left of the figure, where we can observe the interactions between the carrier and the environment, and another, on the right of the figure, where the interactions between the explorer and the environment are shown. The designer can focus, for example, on how the explorer may find food by means of wandering over the ground until finding it, or by finding a pheromone trail that can follow until reaching the food. In this way, the designer can focus on this problem not taking into account how carriers do their work.

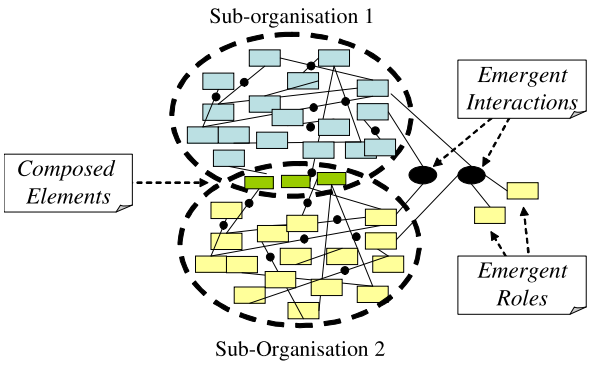

Fig. 9 Composition principle

7.2 Composition

Composition consists of identifying and managing the inter-relationships between the various subsystems in the problem. It makes it possible to group together various basic components and treat them as higher-level units of analysis. Composition makes it possible to describe the high-level interactions between several units. Composition helps to discover subtle interactions between several sub-organizations of the MAS.

In a sense, composition is the required mechanism in order to recreate the abstract complex interactions from their simpler components. In addition, the composition of acquaintance in a sub-organization can be used as the means to build the structural organization. As the roles of an agent are fused, we can draw a "black box" and overlook its internals based only on the interfaces (roles) that cross the boundaries of the box. This process will also help to view a group of agents as a single unit in itself, and help build the hierarchical structure of the organization.

Figure 9 shows that the emergent features that appear at the macro-level are a consequence of the interactions between agents and sub-organizations. Consequently, when several parts of the system are modelled in isolation, we are ignoring the interdependencies between them. That is to say, the whole is greater than the sum of its parts [24]. The lost elements may contain crucial features of the system. For example, the two models of our case study presented in Fig. 8 can be composed, but when doing so, we discover that the interactions *Follow Pheromones* and *Leave Pheromones* are related. This drives us to discover a new interaction that represents the fact that carriers communicate the path to find food to explorers. Figure 10 shows result of the composition of those models.

The advantage of modelling both problems in isolation abstracts these interactions and makes the modelling process easier. It also improves the reuse of models, since their interdependencies would limit the reuse of a combined solution only into systems where both conditions occur. The same principle applies to role composition. Roles are artifacts that can be combined. These artifacts may result in composed roles, or agents playing several roles. When agents are defined as a result of composition, the definition of a structural organization begins to be formed.

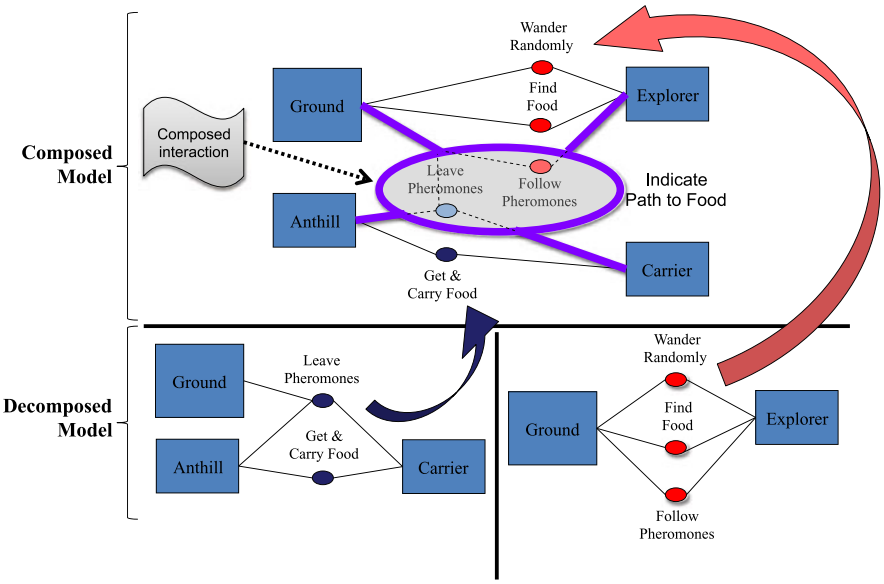

Fig. 10 Composition and decomposition of ants case study

7.3 Techniques for Decomposition and Composition

Decomposition and composition of the main modelling abstractions requires techniques and guidelines to determine feasible separations and to perform them. There are two main approaches to establish where to draw the limit.

- *Functional Decomposition/Goal-based Decomposition*: One of the most direct ways of determining the frontiers between separable/composable parts is through functional decomposition. As agents, and sub-organizations, are designed to achieve their design objectives, a functional subdivision of the system can be easily used. Functional decomposition, as Jennings argues, and Meyer in the OO field [22], is more intuitive and easier to produce than that based upon data and objects. Using this technique, we can analyze an interaction to observe which subgoals can be found on it, and determine which decomposed interactions can be found inside it or which interactions can be grouped to pursue a higher level goal. Notice that this can be also used to divide a big role model into several smaller problems or vice-versa. Notice that this kind of decomposition/composition is the one used in Fig. 10.
- *Dependency Composition/Decomposition*: The other main approach to decomposition/composition is that based on analyzing dependencies between modelling abstractions [3, 36]. Interactions between roles in a MAS are performed to solve small parts of the problem. Each of these interactions modifies the state of the roles participating in it, which is used later to perform further interactions. Thus, we can say that the results of an interaction are used by the rest of interactions.

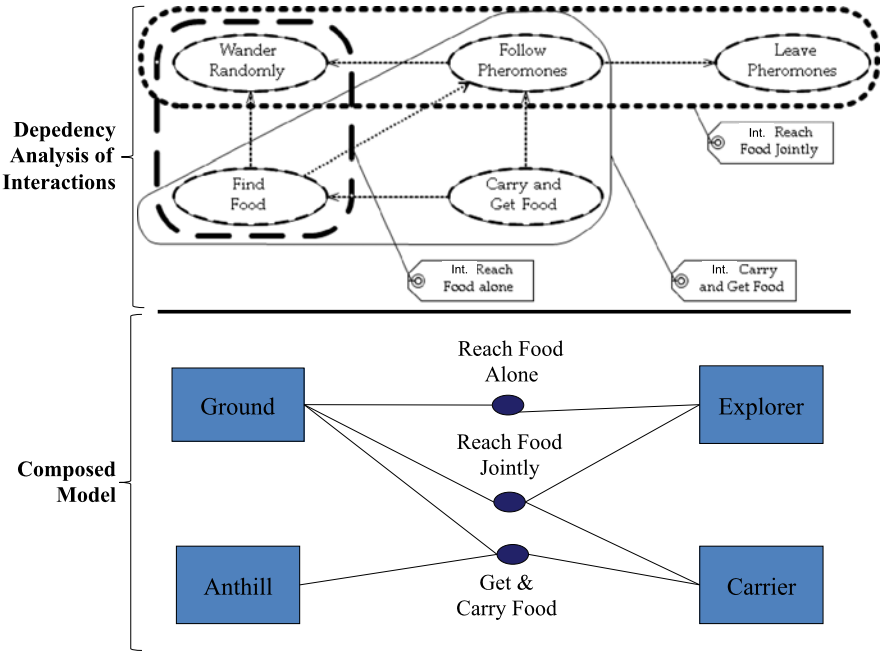

Fig. 11 Intermediate abstraction model of the ant colony by dependency analysis

Given that, we can analyze the dependencies between the state of each role and its interactions to find feasible decompositions or compositions. This kind of analysis has been studied in the distributed systems field where Francez et al. propose various techniques to decompose multiparty interactions [11, 12, 28].

This kind of analysis can also be applied to organizational abstractions. As agents/roles are designed to achieve their design objectives and are limited to a specific environment, sometimes the nature of the problem requires working with part of the environment and the capabilities of other agents/roles. Consequently, the achievement of certain goals is determined by dependencies with other agents [4, 5]. This shows how roles can be grouped/separated to form organizations and it is also useful to determine how they can be composed/decomposed.

As shown in Fig. 10, the model at the bottom is the one obtained directly from the problem description in terms of the behavior of each individual ant observed from an interaction point of view. Given that model, we can analyze the dependencies between interactions in the less abstract view, shown at the top of Fig. 11, to build a more abstract view of the same system obtained by means of interaction composition. From more detailed interactions, we obtain just three of them that abstract the interactions between the explorer and carrier ants with the environmental roles “anthill” and “ground”. This view is closer to the macro level of the system.

8 Reuse and Automation Principles

Automating the modelling process results in lower complexity of models and reduces effort and errors [19]. Some procedures must be carried out that rely on the judgment of the human modellers. However, in recent years the technique that better represents reuse is model driven engineering (MDE) whose goal is to automatically produce a system from requirements, analysis, and/or design models [10].

Reuse is based on using previous knowledge in designing MASs. It saves modellers from redesigning some parts of the system and avoids errors, thus achieving lower complexity of models [6, 19]. Catalysis and OOram are specially concerned about reuse and, as many authors in the agent field [20], present the role concept as the most appropriate tool to reuse functionality.

Reuse is strongly related to the bottom-up software process. When a set of already developed agents or roles are available, e.g., stored in a repository, they can be reused to cover some of the required aspects of the current project. In these situations, we have a highly detailed model of the micro-level of the part of the system implemented by reuse. From the interactions of these reused agents/roles, the required macro-level may or may not emerge. Using a bottom-up approach has proven to be appropriate to transit from the micro-level functionality of reused assets to the macro-functionality required, cf. *Assemble process* in [6, pp. 512–513].

Regarding reuse, the main techniques appearing in the literature are MAS Product Lines, which focus on massive reuse by analyzing common and variable features of MASs to produce a system with the desired features by reusing common features and adding, as automatically as possible, variable features [29].

9 Applying the Principles—Software Process

Although abstract models can provide us with a coherent and simple model, abstract models do not offer enough detail to reach a code model of the system. This problem has been solved in traditional software engineering by maintaining a set of system models structured in several abstraction layers. That is to say, a model of the same problem that is described using a different level of detail. Layered models are presented by Karageorgos as one of the main factors for reducing model complexity [19]. In a layered model, top layers show us abstract models that provide an overview of the system. On the other hand, bottom layers give us the means for detailing top layers, bringing our model nearer to a code model.

As modelling using several abstraction layers usually produces a large amount of models, traceability models are especially important to properly manage such an amount of models as D'Souza shows in [6].

9.1 Top-Down and Bottom-Up

In layered models, the completion of layers is usually done in an iterative way where abstract layers are refined to produce bottom layers and bottom layers are abstracted to produce top layers. That is to say, modelling in a top-down approach or in a bottom-up approach [30].

Top-down approaches correlate with reductionism, that is, designing by starting at the macro-level [13, 24]. Development starts with abstract models of the macro-level of the system. This model is refined until all details are discovered. This approach has some disadvantages. The interactions of systems studied with this approach should be fully known and fully predictable since, otherwise, we will not be able to discover all details. In addition, it misses the flexibility and change adaptation obtained in bottom-up designs.

Bottom-up approaches correlate with emergence, that is, designing by starting at the micro-level [13, 24]. In emergence, development starts at the micro-level defining a set of simple agents. Later, in subsequent layers, these agents are successively grouped into sub-organizations, and the latter into organizations, until reaching the macro-level of the system. This approach also has some advantages and disadvantages. It does not require modelling all interactions in the system since agents can be provided with the expertise necessary to decide their interactions with others at runtime, and thus, the macro-level need not be modelled. However, it requires tuning the macro-level behavior by changes in the micro-level. Bottom-up is also a crucial tool for reuse since reusing a set of agents to implement a new system requires reverse engineering to ensure that the goals of the system are met (cf. Sect. 8).

We cannot state categorically that one is better than the other; it depends on the requirements of the software that we intend to develop. Notice also that not all subparts of a system usually fall in the same Cynefin domain, but tend to spread out in all of them. The best choice in this situation is to apply both, and find a trade-off between them, as Pressman recommends in [30] and Karageorgos in [19].

Finally, note that decomposition and composition can be used to assist top-down and bottom-up approaches, respectively, as we show in the following sections.

9.2 Top-Down Refinement by Means of Decomposition

Decomposition is presented as a principle that supports reductionism, that is to say, a top-down software process. Abstraction mechanisms may result insufficient when we model large and complex MASs since abstract models provide us with an overview of the problem, but not the details. In these cases, we can decompose abstract models to obtain a set of simpler ones which can be easily refined [6]. Using decomposition in this way, we can maintain several layers of abstraction where higher-level layers abstractly represent complex problems and bottom layers store detailed descriptions of sub-parts of top layer models obtained by decomposition.

9.3 Bottom-Up Abstraction by Means of Composition

Composition is the principle that mainly supports emergence, that is to say, it supports a bottom-up software process. We can find two different ways of applying emergence in the literature.

On the one hand, in [17], Jennings follows an emergence approach. He presents bottom-up as a process that is automatically performed by agents at runtime. As shown previously, they draw MASs as highly decomposed structures where problems are "automatically" solved by agents or sub-organizations and where interactions between agents/organizations appear naturally at runtime. Thus, Jennings does not argue for engineers to apply a systematic bottom-up software process to model the system, but he leaves it to be accomplished by the system itself. However, this automation is not always possible since the degree of unpredictability exposed by this kind of designs may be not acceptable for some domain applications, e.g., real-time systems or critical business applications.

On the other hand, a set of models obtained by the decomposition of sub-parts of the system offers a tour of the system specification but does not offer the big picture of it, that is to say, the macro-level behavior [6]. The same problem occurs when the system is modelled as a set of autonomous, self-organizing agents, where the emergent behavior of the system is not explicitly modelled. Designers/implementers must be able to get an overview of it and, at least, have an approximation of the behavior of the system at the macro-level. Composition can be used to get this overview. We can compose finest-grain models to represent, in conjunction with the use of abstraction, the most relevant features in a simple higher-level model and to discover the emergent features that appear [6, 32]. Hence, model composition is an important tool to discover such elements when isolated problems have been properly studied. This reduces the complexity that we are concerned with. Usually, problems to be composed have been previously studied, making the construction of the composite model less complex, since when modelling it, we have to manage only the interrelationships between models and not the whole problem.

9.4 Guidelines for Deciding Between Top-Down and Bottom-Up

We can use three criteria to decide which approach must be applied: (i) the nature of the requirements of the system; (ii) the complexity domain in which each part of the system falls; and, (iii) the available set of reusable agents and models.

Firstly, requirements can be on the macro-level or on the micro-level. On the one hand, typical domain applications, where most requirements can be localized at the macro-level are information systems since requirements show how the overall system should work. On the other hand, typical systems where requirements deal with micro-level are simulation systems. In these systems, the requirements show

Table 1 Summary of criteria for applying top-down and bottom-up

Criteria		Requirements		
		Macro	Micro	Macro & Micro
Complexity Domain	Known	↓	↑	↕
	Knowable	↓	↑	↕
	Complex	↓ / ?	↑ / ?	↕

Criteria With reuse		Requirements		
		Macro	Micro	Macro & Micro
Complexity Domain	Known	↕	↑	↕
	Knowable	↕	↑	↕
	Complex	↕	↑/?	↕

Top-down	Both	Bottom-up	Uncertain Success
↓	↕	↑	?

us how individual agents must work, in order to later study the macro-level of the system.

Secondly, as we showed in Sect. 4, a system usually presents parts in several complexity domains. Depending on the domain that each part falls in, a different software process will fit better with its features.

Thirdly, the level of reuse in a certain project affects the software process since some agents or even organizations of agents and their respective models can be reused to reduce mistakes and time-to-market. When these assets fit with requirements, no extra work is needed.

In Table 1, we show a summary of these criteria. In the following, we show in which situations these criteria point to a top-down or a bottom-up approach, or to both at the same time:

- *Top-down:* must be applied in MASs where most requirements information is concerned with the macro-level. In addition, another reason to apply top-down is that the macro-level required is not usually clear in requirements documents and therefore must be refined to obtain a more accurate model.

  Whenever the requirements scope allows it, top-down should also be applied to such parts of the system that fall in the known or knowable domains since these parts can be fully analyzed by refining abstract descriptions.
- *Bottom-up:* can be used when most requirements information relates to the micro-level, since bottom-up helps us to discover how different micro-level models work together to produce the macro-level.

  Furthermore, if a repository of yet-to-be implemented agents and their models has been constructed because of previous projects, we must primarily apply bottom-up. In this kind of project, the micro-level has to be abstracted to ensure

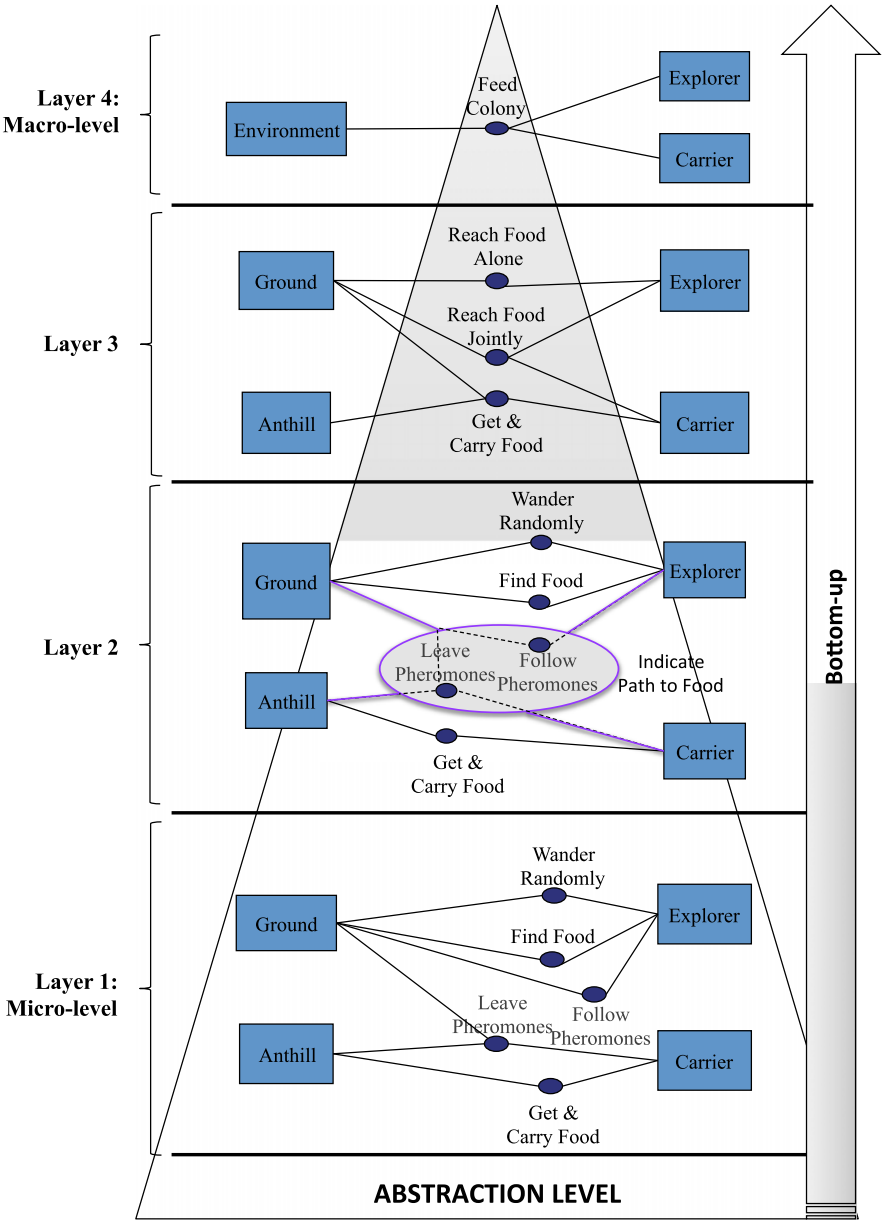

Fig. 12 From micro to macro-level of the ant colony by means of bottom-up

that we meet the desired macro-level and this can be done following a bottom-up approach. That is to say, models or code developed for other projects may not fit completely within the new system; thus, by means of bottom-up, we can

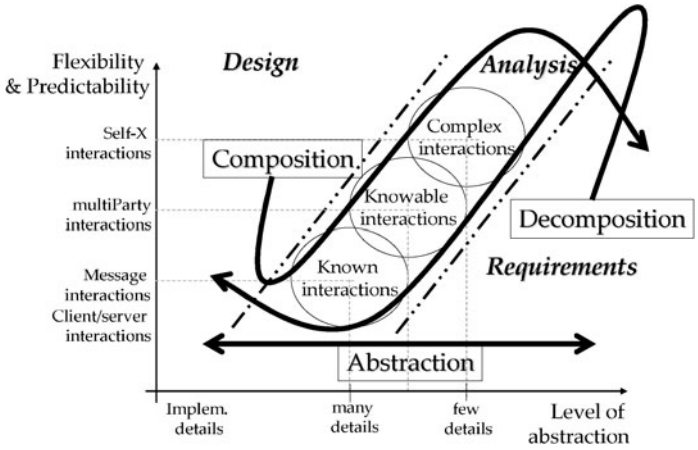

Fig. 13 Usage of conceptual tools to manage taxonomy complexity of interactions

integrate it and analyze to determine if the reused chunk produces the desired macro-behavior.

- *Bottom-up in conjunction with top-down:* must be applied when the system presents features that fit with both previous cases. In addition, it must be applied to the parts of the MAS that fall in the complex domain in order to bridge the gap between macro-level and micro-level. Following this process strategy, we can obtain two layered models of the MAS: one set of layered models for the macro-level and another for the micro-level. Thus, when the least abstract model of the macro-level and the most abstract model of the micro-level overlap, we bridge the gap between both levels.

Revisiting our case study, and taking into account previous guidelines, we can determine that the process that better fits with it is "bottom-up". The main reason for this is that we started with requirements at the micro-level of the system and that the macro-level behavior of it can be seen as complex using the classification provided in Sect. 4. In Fig. 12 we show all the models produced in previous sections by means of compositions of the previous layer until the macro-level behavior of the system is reached. As can be observed, the principles and techniques provided allow us to address a complex system systematically from an engineering point of view.

10 Conclusions

As shown, using the proper tools, namely the three principles to deal with complexity, and focusing on the source of complexity, namely interactions, a problem that can be seen as complex, such as an Ant Colony, can be analyzed systematically to perform engineering models that fall in the known domain.

We have shown the importance of interactions and we have outlined how complexity derived from interactions can be managed from an engineering perspective giving a set of guidelines. Given the findings shown in this chapter, we can summarize how these principles can be applied to transit between known, knowable, and complex interactions as shown in Fig. 13. As depicted, abstraction is lower for known interactions since their details can be easily modelled, while the level of abstraction required for complex interactions is higher, since they can be only understood when observed by their most important features. In addition, complex interactions modelled abstractly can be transformed into knowable and known interactions by means of decomposition. In the reverse process, known interactions, such as those found in code models, can be transformed into knowable and complex interactions by means of composition. The composition process will uncover emergent behaviors inherent to its internal components. The final resulting complex interaction can be further abstracted.

Acknowledgements This work has been partially supported by the European Commission (FEDER) and the Spanish Government under the CICYT project SETI (TIN2009-07366), and by the Andalusian Government under the projects ISABEL (P07-TIC-2533 and TIC-5906) and THEOS (TIC-5906).

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre (www.lero.ie)

References

1. Booch, G.: Object-Oriented Design with Applications. Benjamin/Cummings, Redwood City (1990)
2. Bürckert, H.-J., Fischer, K., Vierke, G.: Teletruck: a holonic fleet management system. In: 14th European Meeting on Cybernetics and Systems Research, pp. 695–700 (1998)
3. Caire, G., Coulier, W., Garijo, F.J., Gómez-Sanz, J.J., Pavón, J., Leal, F., Chainho, P., Kearney, P.E., Stark, J., Evans, R., Massonet, P.: Agent oriented analysis using MESSAGE/UML. In: Proceedings of Agent-Oriented Software Engineering (AOSE'01), Montreal, pp. 119–135 (2001)
4. Castelfranchi, C.: Founding agent's "autonomy" on dependence theory. In: 14th European Conference on Artificial Intelligence, pp. 353–357. IOS Press, Amsterdam (2000)
5. Castelfranchi, C., Miceli, M., Cesta, A.: Dependence relations among autonomous agents. In: Demazeau, I.Y., Werner, E. (eds.) Third European Workshop on Modeling Autonomous Agents in a Multi-agent World. Decentralized AI 3. Elsevier, Amsterdam (1992)
6. D'Souza, D.F., Wills, A.C.: Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison-Wesley, Reading (1999)
7. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Third International Conference on Multi-agent Systems (ICMAS'98), pp. 128–135. IEEE Comput. Soc., Los Alamitos (1998)
8. Ferber, J., Gutknecht, O., Michel:, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J. (eds.) IV International Workshop on Agent-Oriented Software Engineering (AOSE'03). LNCS, vol. 2935, pp. 214–230. Springer, Berlin (2003)
9. Fischer, K.: Agent-based design of holonic manufacturing systems. Robot. Auton. Syst. **27**(1–2), 3–13 (1999)
10. Fischer, K., Hahn, C., Madrigal-Mora, C.: Agent-oriented software engineering: a model-driven approach. Int. J. Agent-Oriented Softw. Eng. **1**, 334–369 (2007)

11. Francez, N., Forman, I.: Synchrony loosening transformations for interacting processes. In: Baeten, J., Klop, J. (eds.) Proceedings of Concurr'91: Theories of Concurrency—Unification and Extension. LNCS, vol. 527, pp. 27–30. Springer, Amsterdam (1991)
12. Francez, N., Forman, I.: Interacting Processes: A Multiparty Approach to Coordinated Distributed Programming. Addison-Wesley, Reading (1996)
13. Fromm, J.: The Emergence of Complexity. Kassel University Press, Kassel (2004)
14. Gerber, C., Siekmann, J., Vierke, G.: Flexible autonomy in holonic multi-agent systems. In: AAAI Spring Symposium on Agents with Adjustable Autonomy (1999)
15. Gerber, C., Siekmann, J., Vierke, G.: Holonic multi-agent systems. Technical report RR-99-03, DFKI, Kaiserslautern, Germany (1999)
16. Giret, A., Botti, V.: Towards an abstract recursive agent. Integr. Comput. Aided Eng. **11**(2) (2004)
17. Jennings, N.: An agent-based approach for building complex software systems. Commun. ACM **44**(4), 35–41 (2001)
18. Jennings, N.R.: Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. Int. J. Intell. Coop. Inf. Syst. **2**(3), 289–318 (1993)
19. Karageorgos, A., Mehandjiev, N.: A design complexity evaluation framework for agent-based system engineering methodologies. In: Omicini, A., Petta, P., Pitt, J. (eds.) Fourth International Workshop Engineering Societies in the Agents World. LNCS, vol. 3071, pp. 258–274. Springer, Berlin (2004)
20. Kendall, E.A.: Role modeling for agent system analysis, design, and implementation. IEEE Concurr. **8**(2), 34–41 (2000)
21. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd edn. Prentice Hall, Upper Saddle River (2001)
22. Meyer, B.: Object-Oriented Software Construction. Prentice Hall, Hertfordshire (1988)
23. Mintzberg, H.: The Structuring of Organizations. Prentice Hall, Upper Saddle River (1978)
24. Odell, J.: Agents and complex systems. J. Object Technol. **1**(2), 35–45 (2002)
25. Odell, J., Parunak, H.V.D., Bauer, B.: Representing agent interaction protocols in UML. In: Proceedings of the 1th Int. Workshop on Agent-Oriented Software Engineering (AOSE'00). LNCS, vol. 1957. Springer, Limerick (2000)
26. Parunak, H.V.D., Odell, J.: Representing social structures in UML. In: Müller, J.P., Andre, E., Sen, S., Frasson, C. (eds.) Proceedings of the Fifth International Conference on Autonomous Agents, pp. 100–101. ACM Press, Montreal (2001)
27. Peña, J.: On improving the modelling of complex acquaintance organisations of agents. A method fragment for the analysis phase. PhD thesis, University of Seville (2005)
28. Peña, J., Corchuelo, R., Ruiz-Cortés, A., Toro, M.: Towards an automatic method for detecting synchrony loosening anomalies in the context of multiparty interactions. In: Actas del II taller de trabajo sobre Desarrollo de Software Preciso. VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'01), Almagro (Ciudad Real, Spain) (2001)
29. Peña, J., Hinchey, M.G., Cortés, A.R.: Multi-agent system product lines: challenges and benefits. Commun. ACM **49**(12), 82–84 (2006)
30. Pressman, R.S.: Software Engineering: A Practitioner's Approach, 2nd edn. McGraw-Hill, New York (1986)
31. Reenskaug, T.: A methodology for the design and description of complex, object-oriented systems. Technical report, Center for Industrial Research, Oslo, Norway (November 1988)
32. Reenskaug, T.: Working with Objects: The OOram Software Engineering Method. Manning Publications, Greenwich (1996)
33. Snowden, D., Kurtz, C.: The new dynamics of strategy: sense-making in a complex and complicated world. IBM Syst. J. **42**(3), 35–45 (2003)
34. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley, Reading (2002)
35. Wirfs-Brock, R., McKean, A.: Object-Oriented Design: Roles, Responsibilities, and Collaborations. Addison-Wesley, Reading (1990)
36. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the GAIA methodology. ACM Trans. Softw. Eng. Methodol. **12**(3), 317–370 (2003)