Property-driven design for swarm robotics

Manuele Brambilla, Carlo Pinciroli, Mauro Birattari and Marco Dorigo
Université Libre de Bruxelles
Brussels, Belgium
{mbrambil,cpinciro,mbiro,mdorigo}@ulb.ac.be

ABSTRACT

In this paper, we propose a novel top-down design method for the development of collective behaviors of swarm robotics systems called *property-driven design*. Swarm robotics systems are usually designed and developed using a *code-and-fix* approach, that is, the developer devises, tests and modifies the individual robot behaviors until a desired collective behavior is obtained. The code-and-fix approach can be very time consuming and relies completely on the ingenuity and expertise of the designer. The idea of property-driven design is that a swarm robotics system can be described by specifying formally a set of desired properties. In an iterative process similar to test-driven development, the developer produces a model of the system that satisfies the desired properties. Subsequently, the system is implemented in simulation and using real robots. Property-driven design helps to minimize the risk of developing a system that does not satisfy the required properties, and to promote the reuse of hardware independent models. In this paper, we start by giving a general description of the method. We then present a possible way to apply it by using Discrete Time Markov Chains (DTMC) and Probabilistic Computation Tree Logic* (PCTL*). Finally, we conclude by presenting the application of the proposed method to the design and development of a swarm robotics system performing aggregation.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Robotics

General Terms

Design, Verification

Keywords

Swarm robotics, Swarm engineering, Top-down design, Aggregation

1. INTRODUCTION

Swarm robotics is a distributed approach to multi-robot systems in which, through local interactions, robots achieve a self-organized collective behavior. Swarm robotics systems have the potential to display interesting properties, such as robustness, scalability and flexibility [13].

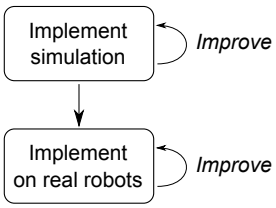

Figure 1: The most common method for the development of a swarm robotics system. The individual behavior is developed, tested and modified until the desired collective behavior of the swarm is obtained.

Swarm robotics systems are complex systems [9]. They exhibit dynamics at two different levels: the collective, or macroscopic, level, and the individual, or microscopic, level. The collective behavior is the result of the interactions of the individual robots with each other and with the environment. In order to obtain a desired collective behavior, the individual behaviors and the interactions of the robots must be carefully designed. However, this design process is usually non-trivial, as the dynamics of complex systems are very often difficult to predict [1].

Despite the increasing attention on swarm robotics systems in the past two decades [4], a top-down methodology for the design and development of this kind of systems has not been defined yet. Swarm robotics systems are usually designed and developed using a *code-and-fix* approach [7]. This means that, usually, the individual behavior is developed, tested and modified until the desired collective behavior is obtained. This process is often performed first in computer simulations and eventually on real robots (see Fig. 1).

We believe that, to improve the quality of swarm robotics systems and to reduce the effort for their development, it is necessary to create a new, specific branch of engineering that we call *swarm engineering*. We define swarm engineering to be the systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate and maintain an artificial swarm intelligence system.

Traditional system engineering approaches are not suited for swarm robotics systems. System engineering is mainly aimed towards centralized systems or, in general, systems in which the interactions of the components can be precisely

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

predicted. In this respect, swarm robotics systems, which can have hundreds of interacting robots, present unprecedented challenges [12].

In this paper, we propose a top-down design method, that we call *property-driven design*. Property-driven design provides ways to specify requirements, design, develop, verify and validate a swarm robotics system.

We believe that property-driven design has many advantages compared with code-and-fix development: it helps to formally specify the requirements of the system; to reduce the risk of developing the "wrong" system, that is, a system that does not satisfy the requirements; to develop a set of hardware independent models that can be reused for future applications; and to shift the focus of the development process from implementation to design, given that most of the developing effort happens in the modeling phase.

In Section 2, we present related work on top-down design methods and verification techniques for swarm robotics. In Section 3, we present property-driven design. In Section 4, we propose a possible way to specify properties and validate a model for a swarm robotics system. In Section 5, we present an example application of property-driven design to the design and development of a system able to perform aggregation. In Section 6, we discuss about the features and limits of property-driven design. In Section 7, we conclude this paper.

2. RELATED WORK

Design methods: Developing a top-down design method for complex systems is still an open challenge [31]. In the last years, the effort on this topic has been quite limited.

Bachrach et al. [2] proposed a scripting language called *Protoswarm*. This language enables the definition of a vector field on an abstract spatial machine. This vector field is then translated by a middleware into individual robot behaviors. Protoswarm allows the developer to focus mostly on the collective behavior, removing some of the effort necessary to develop the individual behaviors. However, Protoswarm is thought for situations in which the robots are covering the entire environment and keep constant network connectivity. Thus, it is more suited for sensor networks than for swarm robotics systems. Another thing to note is that Protoswarm is not a design method, but a scripting language. As such, it can be used only as a development tool, requiring an appropriate design method to guide the process.

Kazadi et al. [19] proposed *the Hamiltonian method*, a design method based on Hamiltonian vector fields. This method allows one to develop systems by specifying one or more numerical properties, such as the energy level of a particular state of the system. One limitation of this design method is that it is suited only for spatially-organizing behaviors. The goal of spatially-organizing behaviors is to achieve a specific robot distribution in the environment, such as pattern formation (e.g., [30]).

Another possible approach to the design of swarm robotics systems are automatic design methods. Work on automatic design methods for swarm robotics systems focuses mainly on evolutionary robotics [24] and reinforcement learning [26]. Automatic design methods can be considered top-down approaches because, in principle, the development process is driven by the desired collective-level goal behavior. However, a lot of domain knowledge is required to tackle medium to complex applications. Moreover, once a system is obtained, it is, in general, non-trivial to understand its behavior and it is often very difficult to verify its properties or adapt it to other applications, even if they are similar to the original one.

Property verification: The problem of property verification in swarm robotics systems has been tackled only in a limited way. Dixon et al. [14], used Linear Temporal Logic (LTL) to define properties of individual robots and of the swarm. This method is based on modeling the individual robot behavior with a Markov chain, and then considering the collective behavior as the result of the *and*-composition of these individual-level models. A limitation of this approach is that linear temporal logic deals only with binary values. This limits the possibility to analyze systems displaying stochastic properties, such as non-trivial swarm robotics systems. Furthermore, in this method, there is a possible scalability problem, because the number of states of the system grows exponentially with the number of robots: $\sim \Theta(k^n)$, where k is the number of states of the individual Markov chain and n is the number of robots.

Recently, Konur et al. [20] proposed an approach to verify formally the properties of a swarm behavior through *probabilistic computation tree logic* [16]. Their approach is able to overcome the limits of linear temporal logical while providing scalability.

3. PROPERTY-DRIVEN DESIGN

The idea behind property-driven design is that a swarm robotics system can be formally described through a series of properties. These properties are the distinguishing features of the system the developer wants to realize. They can be task specific, such as *the system eventually completes the task X*, or they can express more generic properties, such as *the system keeps working as long as there are at least N robots* or *the system will never enter state Y*.

A schema showing the different steps of property-driven design is presented in Figure 2.

Phase One: The first phase of property-driven design consists in formally specifying the requirements of the system by stating its desired properties. The clearer and more complete these properties are in this phase, the more the developed system will conform to the requirements.

Phase Two: In the second phase, a model of the system is created. At first, similarly to test-driven development [5], one cannot expect the system to satisfy all the desired properties. In an iterative process, the developer expands and improves the model, and checks whether the properties are verified. The outcome of this process is a model of the system that satisfies the stated properties. Note that the model must be complete just enough to capture all the important characteristics of the system, avoiding unnecessary complication. For example, to model failures, the developer could insert in the model only a general *failure* state, without specifying all the possible hardware problems if not necessary. Eventually, through this process, one obtains a model that satisfies all the required properties.

At the end of this phase, the developed model is robot independent. The developer can now identify a set of necessary sensors and actuators to select the proper robot platform to use in implementing the system.

Phase Three: In this phase, the developer can use the model to guide the process of implementing the swarm

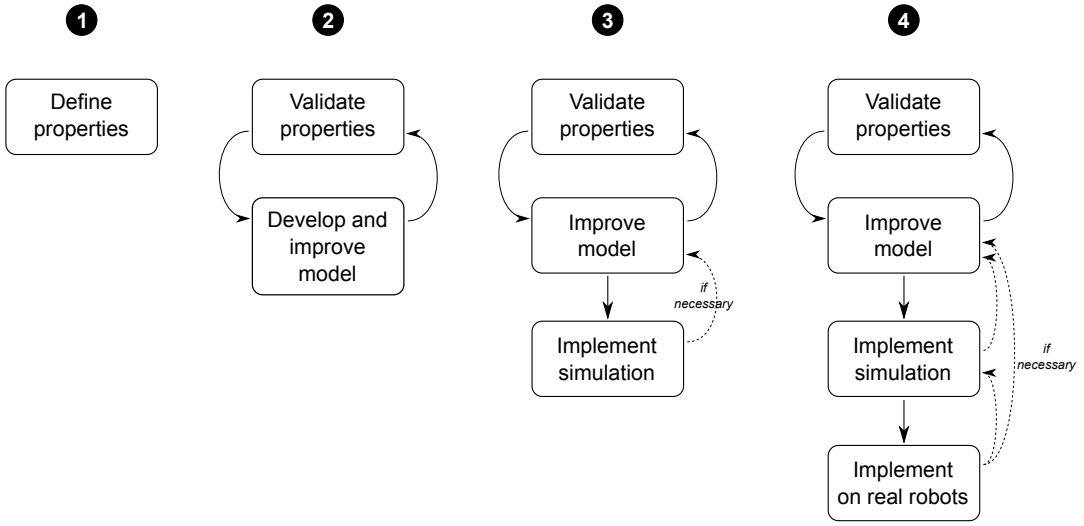

Figure 2: The four different phases of property-driven design to develop a swarm robotics system.

robotics system using, for instance, a physics-based computer simulation (henceforth simply simulation). As discussed in Section 1, this phase can be challenging, as moving from a macroscopic model to the microscopic implementation is a process that is guided mainly by the ingenuity and expertise of the developer. However, the defined model gives a clear picture of the system that can greatly help in its development.

It is possible that the simulation does not validate the model [22]. In this case the developer must go back to step 2, modify the model to include the results obtained from the simulation, and verify whether the required properties still hold true.

Phase Four: The last phase consists in deploying the system on real robots.

Similarly to the transition between the model and the simulation, if the implementation on real robots reveals that some assumptions made during the previous phases do not hold, it might be necessary to modify the simulated version or the model, in order to keep all levels consistent.

4. DTMC AND PCTL*

So far, we purposely did not mention how to model the system or how to specify its properties. There are several possible ways to perform this activity. Here, we do not discuss the different options available, as a review of the possible techniques for modeling swarm robotics system is out of the scope of this paper. The interested reader can refer to Lerman et al. [21].

Of all the various possibilities, the developer can choose the one that best fits the system to develop and its personal experience. In this section, we briefly introduce one possible way to model a swarm robotics system and specify its properties based on Deterministic Time Markov Chains (DTMC) and Probabilistic Computation Tree Logic* (PCTL*).

DTMC are often used to model swarm robotics systems [21]. One of the main advantages of DTMC is that, in many cases, the model comprises both the microscopic and the macroscopic levels. At the microscopic level, the model represents the behavior of a single robot. At the macroscopic level, each state can be used to count the number of robots in that particular state. For example, at the microscopic level, one can model the behavior of a single robot with a 3-state DTMC. The same DTMC can be augmented by associating a counter to each state. Each counter keeps track of the number of robots in the associated state. Another advantage of DTMC is that their use can ease property verification, especially if the properties are written through the use of logic predicates [11].

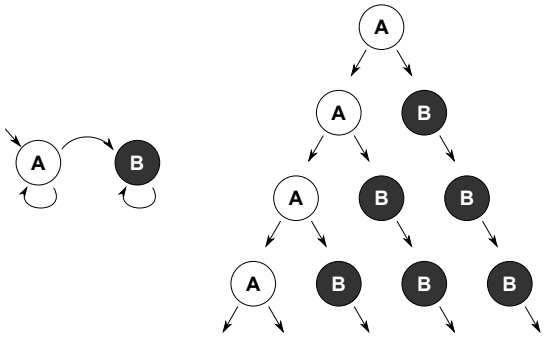

Figure 3: A simple Markov chain (on the left) and its computation tree (on the right).

Among the many formal logical systems, we consider Probabilistic Computation Tree Logic* (PCTL*). PCTL*, originally developed by Hansson and Jonsson [16], is an extension of CTL (Computation Tree Logic), a branching time logic. CTL is based on the idea that a Markov chain can be "expanded" in a computation tree. A computation tree is a potentially infinite rooted tree in which the root is the initial state of the corresponding Markov chain, and each node is a possible state of the system. The edges link a state with its next possible states. An example of a simple Markov chain and its computation tree is displayed in Figure 3.

Through CTL, one can express time-related properties

such as *property α will eventually become true* or *property α will hold true for at least 10 seconds*. PCTL* extends CTL by introducing probabilities. In this way, one can express properties such as *property α will eventually become true with probability 0.45* or *there is a 0.7 probability that α will hold true for 10 seconds*. PCTL* is well suited for swarm robotics systems as it can capture the time-related and stochastic aspects of this kind of systems. We do not discuss the details of the presented logics, we refer the interested reader to Ciesinski and Größer [10].

Our approach is based on model checking, a technique that allows to verify automatically and completely whether a set of formulae is satisfied by a given system. As model checker software we choose PRISM [17]. PRISM is a probabilistic model checker which supports DTMC and PCTL* among many other models and logics. With PRISM, it is possible not only to verify properties, but also to perform so called "experiments," in which the model checker computes the probability of the property being true against different parameter values. In this way, it is possible to find the parameter set that scores the best probability in verifying a property.

5. AN EXAMPLE APPLICATION: AGGREGATION

In order to show the characteristics of property-driven design, we present an example application: *aggregation*. In this application the robots have to cluster in an area of the environment. The robots have neither a map of the environment nor knowledge of the position of the other robots. We choose aggregation as a case study for four reasons:

- aggregation is a simple behavior: this allows us to focus on the development process without being hampered by the details of the system itself;
- aggregation is a common test-case behavior for the swarm robotics community, and it has been studied extensively in the past (see, for example, [3, 22, 29, 32, 6]);
- aggregation is a collective behavior that cannot be developed easily with Protoswarm [2], since the robots can often lose network connectivity; or using the Hamiltonian method [19], since no specific spatial distribution is required once the aggregate is formed;
- aggregation possesses many of the salient traits of a typical swarm robotics behavior. It is completely distributed, it is based on simple robot-to-robot interactions, and it is characterized by stochasticity and spatial requirements.

The collective behavior we study in this paper is similar to the one presented by Jeanson et al. [18]. We consider a dodecagonal environment with two black spots called *area A* and *area B*. We call *area C* the remaining white area. Each of the black spots is big enough to host all the robots. See Figure 7 for a screenshot of the environment. In the following, we will follow the 4-phase process explained in Section 3 using DTMC and PTCL* as modeling tools and PRISM as model checker.

Phase One: The property we focus on is *"eventually all the robots form an aggregate"*. We would like that the aggregate is formed as fast as possible (for example, in the first 1,000 seconds) but we do not differentiate between obtaining the aggregate in area A or area B. Using PRISM syntax, we can define this property as follows:

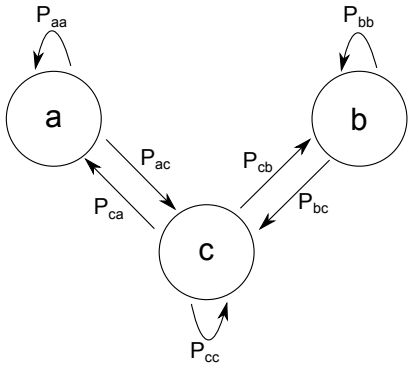

Figure 4: The DTMC model of the aggregation example. Each state is used to count the number of robots in that particular area.

```
P=? [F<=1000 (a=N_total)|(b=N_total)]        (1)
```

In less formal terms, we compute the probability (`P=?`) that, in the first thousand seconds (`F<=1000`), the number of robots in area A or in area B is equal to the total number of robots in the swarm (`(a=N_total)|(b=N_total)`). Since we want to maximize this probability, we do not specify a value for it.

Another property we want for the system is that the aggregate, once formed, is stable for a certain period. In this example we set such period to 10 seconds. We verify this property with probability greater or equal to $\frac{2}{3} \simeq 0.67$:

```
(a=N_total)|(b=N_total) =>
P>=0.67 [G>=10 (a=N_total)|(b=N_total)]        (2)
```

In natural language, Property 2 can be expressed in this way: from the aggregate state (`(a=N_total)|(b=N_total)`) is it true with probability greater or equal to 0.67 (`=> P>=0.67`) that the system stays for at least 10 seconds (`G>=10`) in the aggregate state?

Phase Two: Once the above desired properties have been specified we need to build the model. We start by setting the total number of robots in the system. In order to perform a scalability test, we selected three different group sizes: $N_t = 10, 20, 50$[1]. Then, we specify three states: state S_a, S_b and S_c. A robot in area A or B is in state S_a or S_b, respectively. Robots outside area A or B are in state S_c. Moreover, three counters `a`, `b` and `c` are associated to the respective states. These counters are used to keep track of the number of robots that are in state S_a, S_b and S_c, respectively. Note that `a+b+c`=N_t. See Figure 4 for the DTMC model of the system.

We design the following behavior for a robot: it performs random walk and when it finds a black area it stops with probability 1. The robot then decides whether to leave according to a certain probability.

[1]We tested the system also with 100 robots, but the probability of obtaining an aggregate in less than 1000 seconds was close to 0. We decided thus not to include these results in order to simplify the explanation.

In this initial stage of the definition of the model, we assume that our system can be effectively described by a *non-spatial* model, that is, a model in which the trajectories of the robots are ignored and a robot can move instantaneously from area C to area A or B, and vice versa. Moreover, we also ignore the effects of interferences between robots [21]. However, especially for larger group sizes, the performance of the system may be reduced by the fact that robots must avoid each other or that robots stopping in the black areas prevent other robots from entering it. In case these assumption proves to be not realistic and the results obtained with the model do not match those obtained in simulation or with real robots, we will modify them in the following phases, as explained in Section 3. Note that a model of a similar system is presented in O'Grady et al. [25].

Since our model is non-spatial and ignores interference, we consider only the geometric properties of the areas to compute p_{ca}. A robot in area C can either go in area A, go in area B or stay in area C. This means that a robot in area C has a probability of going from area C to area A equal to $p_{ca} = \frac{A_A}{A_{arena}}$, of going from area C to area B equal to $p_{cb} = \frac{A_B}{A_{arena}}$, and of staying in area C equal to $p_{cc} = \frac{A_C}{A_{arena}} = 1-(p_{ca}+p_{cb})$. In our scenarios we used three different arena sizes for the three different group sizes. In Table 1 it is possible to find the details about the parameters used for the experiments.

We need to define the remaining probabilities. The aggregate can be obtained in area A or area B, thus we set the probabilities of leaving these two areas to be equal: $p_{ac} = p_{bc}$. Since the two areas have the same size we set $p_{aa} = p_{bb}$. A robot in area A can only return to area C or stay in area A, thus $p_{aa} = 1 - p_{ac}$. The only independent probability remaining is p_{ac}. Initially, we set p_{ac} to a fixed value. Through model checking, we can find the value of p_{ac} that maximizes the probability of satisfying Property 1. The process consists in automatically testing the model for different p_{ac} values and find the best one.

The best values found with PRISM are $p_{ac} = 0.05, 0.04, 0.04$ when $N_t = 10, 20, 50$, respectively. With these values, the probabilities of satisfying Property 1 are 0.75, 0.15 and 8.8×10^{-5}. Property 2 is not satisfied for any of the three group sizes. The developed behavior, thus, obtains poor results and the system does not cope well with increasing group sizes.

It is thus necessary to improve the developed model by modifying the behavior of the robots. A fixed p_{ac} does not promote the formation of a single cluster. A better solution is to let a robot decide whether to leave according to the number of sensed robots around it [18]: with only few robots nearby, the probability to leave the aggregate p_{ac} is high and vice versa. We set $p_{ac} = p_{min-ac} * (N_s + 1)$, where p_{min-ac} is the minimum staying probability we want for a robot and N_s is the number of other robots sensed. We add 1 to the number of robots sensed, as we want to include also the robot that is choosing its next action. Subsequently, we use PRISM to find the best value of p_{min-ac} for the different group sizes. As reported in Table 1, results are better than before, both for Property 1 and Property 2.

With the current model we are also able to define requirements on the hardware capabilities of the robots: a ground sensor, to differentiate between the two black areas A and B and the white area C; a sensor to detect nearby robots; and wheels to move. An example of such a robot is the e-puck [23] robot which has a range and bearing board that allows it to perceive the presence of neighboring robots [15].

Table 1: A table that presents the obtained results. Column p_{min-ac} shows the best value of p_{min-ac} found using PRISM. Column Pr 1 shows the probability of satisfying Property 1, and column Pr 2 shows whether Property 2 is satisfied.

N_t	A_A	A_{arena}	p_{ca}	p_{min-ac}	Pr 1	Pr 2
10	$0.38m^2$	$4.91m^2$	0.0784	[0.19, 0.24]	0.95	✓
20	$0.78m^2$	$19.63m^2$	0.0625	0.12	0.79	✓
50	$3.14m^2$	$50.26m^2$	0.0625	0.10	0.25	✓

Phase Three: In this aggregation example, the model captures well the microscopic behavior of the single robots, thus it is quite easy to implement the system in simulation. However, several implementation details are not explicitly present in the model, such as how the robots perform random walk. These implementation details must be dealt with in such a way that they do not falsify the model.

We implemented the system using the ARGoS simulator [27]. Figure 6 shows a screenshot of the simulated system. We performed three different sets of experiments, one for each group size. To validate the model we measured the average time necessary to form a complete aggregate on 100 runs with different values of p_{min-ac}. The robots were deployed in a random position at the beginning of each experiment. Each experiment stopped when a complete aggregate was formed or after 10,000 seconds.

As reported in Figure 5, for all the three group sizes, the best results were obtained with the value p_{min-ac} predicted using the model. However, the results obtained for Property 1 with the simulated version of the system are usually worse than those predicted by the model, in particular with 20 and 50 robots. With 10 robots and $p_{min-ac} = 0.22$ the simulated system was able to form a complete aggregate before 1,000 seconds 100 times out of 100, in line with the model predictions. However, with 20 robots and $p_{min-ac} = 0.12$, Property 1 was satisfied only 53 times out of 100, whereas in the model it was satisfied with a probability of 0.79. With 50 robots and $p_{min-ac} = 0.10$ the difference is even more evident: only 2 runs out of 100 resulted in an aggregation time of under 1,000 seconds whereas the model predicted a probability of satisfying Property 1 of 0.25.

As explained in Section 3, since the results obtained from the model and from the simulation do not match, we need to modify the model in order to make them consistent. The discrepancy between the model and the simulated system is due to the fact that, as the number of robots grows, interference between robots reduces p_{ca}. This is because the robots spend more time avoiding collisions and because the robots stopping in the black areas prevent other robots from accessing them. Reducing p_{ca} in the model allows us to obtain results that are closer to those obtained in simulation. For 20 robots and $p_{ca} = 0.0475$, we observe that Property 1 is satisfied with probability 0.5275, which matches the results obtained in simulation. For 50 robots we set $p_{ca} = 0.041$, which gives a probability of satisfying Property 1 of 0.01.

We also tested Property 2. 100 runs of the simulated experiments were executed for 10,000 seconds with the three

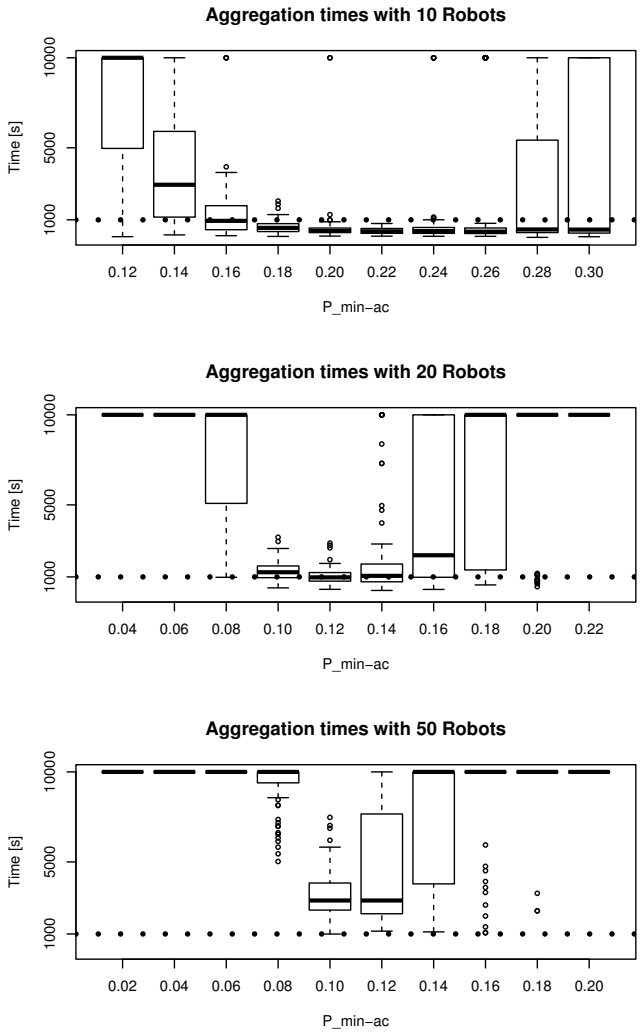

Figure 5: Some results obtained with the ARGoS simulator. The graphs show the time necessary to form the complete aggregate with different p_{min-ac} over 100 runs for 10, 20 and 50 robots.

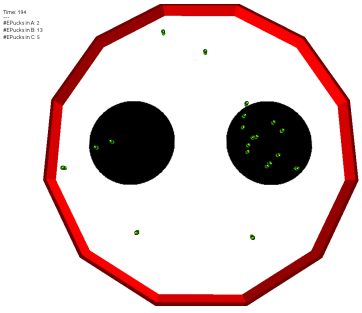

Figure 6: A screenshot of the simulated version of the system using 20 robots.

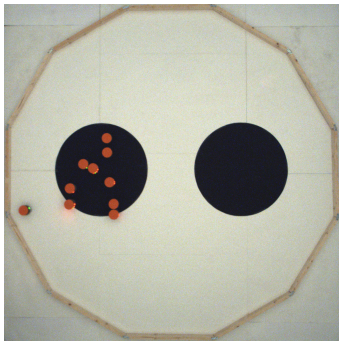

Figure 7: A screenshot of the experiment performed with the 10 e-puck robots.

group sizes. In the experiments, we measured whether the system satisfies Property 2, that is, whether a complete aggregate, once formed, lasts more than 10 seconds. In all the cases in which a complete aggregate was formed before 10,000 seconds, Property 2 was satisfied.

Videos of the simulated experiments are available in the supplementary pages [8].

Phase Four: In the last phase, we implement the system using real e-pucks. We performed 10 experiments with a group of 10 e-pucks in an arena identical to the simulated one. A picture of an experiment can be seen in Figure 7. Figure 8 shows a comparison between the results obtained with the real robots and in simulation. A video of a run is available in the supplementary pages [8].

In 10 runs out of 10, both Property 1 and Property 2 were satisfied. The results obtained with the real robots are in line with those obtained in simulation, even though the aggregation time is slightly longer. This is probably due to a higher wheel speed in the simulated experiments.

6. DISCUSSION

Property-driven design aims at supporting the development of swarm engineering, that is, a systematic application of scientific and technical knowledge to specify requirements, design, realize, verify, validate, operate and maintain a swarm intelligence system. The code-and-fix development method for swarm robotics systems relies completely on the ingenuity and experience of the developer. On the contrary, the proposed property-driven design offers several advantages.

First, property-driven design is an iterative process that guides and helps the designer in developing the system. The great majority of iterations occur when building the model. This allows the developer to focus only on the important aspects of the system because "whereas a simulation should include as much detail as possible, a good model should include as little as possible" [28],

Another advantage is that the risk of developing a system that does not satisfy the required properties is reduced, as these properties are evaluated at each step of the design and development phase. With the code-and-fix development these properties are either not verified or verified a posteriori, so the risk of developing the "wrong" system is high. Note that this holds only if the model is "good", meaning that it is able to faithfully represent the system. In this

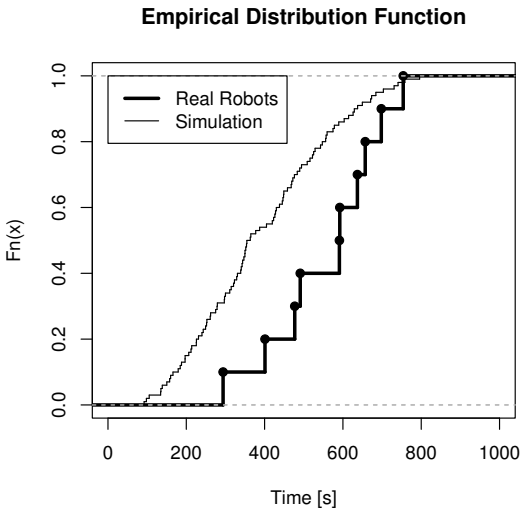

Figure 8: A graph showing the empirical cumulative distribution function ($Fn(x)$) of the results obtained with real robots (10 runs) and in simulation (100 runs). In both cases $p_{min-ac} = 0.22$.

paper, we did not discuss in details model validation. The interested reader can see, for example, Martinoli et al. [22].

Finally, since the developed model is hardware independent, it can be used to choose the robotic platform that fits best the characteristics of the system. Also, the model can be partially or completely reused for other applications, limiting the issue known as "reinventing the wheel". In the future, it is possible to imagine a set of publicly available models for swarm robotics applications that can be reused and modified by other developers.

While property-driven design has many advantages, it also has some limits. Being based on modeling, property-driven design inherits its advantages and limits. As with modeling, property-driven design can be applied to a large variety of swarm robotics systems. However, modeling a swarm robotics system is a hard task on its own. Many critical aspects of a swarm robotics system, such as robot-to-robot interaction or time and spatial aspects of the system are not always easy to capture in a model. Fortunately, many aspects of modeling a swarm robotics system have been studied extensively over the years (see, for instance, a review on modeling [21]).

Property-driven design can guide the developer in designing and developing a swarm robotics system. However, depending on the complexity of system to develop, implementing the model in simulation (phase 3) might be complicated. In these cases the ingenuity and expertise of the developer are still necessary.

7. CONCLUSION

In this paper, we presented property-driven design: a top-down design method based on the idea that a swarm robotics system can be described through a series of properties. Once these properties have been specified, it is possible to create a model of the system that satisfies them. In an iterative process, the model is improved until it correctly describes the system that the developer wants to design and satisfies the desired properties. The obtained model can then guide the development of a computer simulated version of the system.

Property-driven design is one of the first attempts towards the development of swarm engineering. Differently from code-and-fix development, property-driven design offers a systematic approach towards the development of a swarm robotics system. Among the advantages of property-driven design, compared with code-and-fix development, there are: a shift of the focus of the development process from implementation to design, given that most of the iterations happen at the design level; a formal way to specify the requirements of the system; a reduced risk of developing the "wrong" system, that is, a system that does not satisfy the requirements; and the possibility to develop a set of hardware independent models that can be reused for future applications.

In the future we plan to apply property-driven design to more complex applications, possibly using different modeling approaches. One problem to tackle is deriving the individual behavior of the robots starting from a collective behavior. Several possibilities can be studied, such as the integration of property-driven design with spatial computing or artificial evolution.

8. ACKNOWLEDGMENTS

This work was partially supported by the European Union through the ERC Advanced Grant "E-SWARM: Engineering Swarm Intelligence Systems" (contract 246939) and by the Future and Emerging Technologies project "ASCENS" (contract 257414).

Manuele Brambilla, Mauro Birattari and Marco Dorigo acknowledge support from the F.R.S.-FNRS of Belgium's Wallonia-Brussels Federation, of which they are a F.R.I.A. Research Fellow, a Research Associate and a Research Director, respectively.

9. REFERENCES

[1] R. Abbott. Emergence explained. *Complexity*, 12(1):13–26, 2006.

[2] J. Bachrach, J. Beal, and J. McLurkin. Composable continuous-space programs for robotic swarms. *Neural Computation & Applications*, 19:825–847, 2010.

[3] E. Bahçeci and E. Şahin. Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *Proceedings of the 2005 Swarm Intelligence Symposium - (SIS 2005)*, pages 333–340, Piscataway, NJ, 2005. IEEE Press.

[4] L. Bayindir and E. Şahin. A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering*, 15(2):115–147, 2007.

[5] K. Beck. *Test-driven Development: By Example*. Addison-Wesley, Boston, MA, 2003.

[6] S. Berman, A. Halasz, M. Hsieh, and V. Kumar. Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics*, 25(4):927–937, 2009.

[7] B. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[8] M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo. Property-driven design for swarm robotics:

Complete data, 2011. Supplementary information page at http://iridia.ulb.ac.be/supp/IridiaSupp2011-018/.

[9] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, 2001.

[10] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, number 2925 in Lecture Notes in Computer Science, pages 333–355. Springer, Berlin, Germany, 2004.

[11] E. Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science*, number 1346 in Lecture Notes in Computer Science, pages 54–56. Springer, Berlin, Heidelberg, 1997.

[12] D. Cleary. Perspectives on complex-system engineering. *Collaborations*, 3(2):1–4, 2005.

[13] E. Şahin. Swarm robotics: from sources of inspiration to domains of application. In *Swarm Robotics*, volume 3342 of *Lecture notes in computer science*, pages 10–20. Springer, Berlin, Heidelberg, 2005.

[14] C. Dixon, A. Winfield, and M. Fisher. Towards temporal verification of emergent behaviours in swarm robotic systems. In *Towards Autonomous Robotic Systems*, volume 6856 of *Lecture Notes in Computer Science*, pages 336–347. Springer, Berlin, Heidelberg, 2011.

[15] A. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena. Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In *IEEE International Conference on Robotics and Automation – ICRA 2009*, pages 3111–3116. IEEE Press, Piscataway, NJ, 2009.

[16] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[17] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, number 3920 in Lecture Notes in Computer Science, pages 441–444. Springer, Berlin, Germany, 2006.

[18] R. Jeanson, C. Rivault, J.-L. Deneubourg, S. Blanco, R. Fournier, C. Jost, and G. Theraulaz. Self-organized aggregation in cockroaches. *Animal Behaviour*, 69(1):169–180, 2005.

[19] S. Kazadi, J. R. Lee, and J. Lee. Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics, Special Issue on Swarm Robotics*, 2(4):672–694, 2009.

[20] S. Konur and C. Dixon. Formal verification of probabilistic swarm behaviours. In *Swarm Intelligence, 7th International Conference, ANTS 2010*, volume 6234 of *Lecture Notes in Computer Science*, pages 572–573. Springer, Berlin, Germany, 2010.

[21] K. Lerman, A. Martinoli, and A. Galstyan. A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 143–152. Springer, Berlin, Heidelberg, 2005.

[22] A. Martinoli, A. J. Ijspeert, and F. Mondada. Understanding collective aggregation mechanisms: from probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 29(1):51–63, 1999.

[23] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65, Portugal, 2009. IPCB: Instituto Politécnico de Castelo Branco.

[24] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[25] R. O'Grady, C. Pinciroli, A. L. Christensen, and M. Dorigo. Supervised group size regulation in a heterogeneous robotic swarm. In *Proceedings of ROBOTICA 2009 - 9th International Conference on Autonomous Robot Systems and Competitions*, pages 113–119. IPCB, Castelo Branco, Portugal, 2009.

[26] L. Panait and S. Luke. Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[27] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, T. Stirling, A. Gutiérrez, L. M. Gambardella, and M. Dorigo. ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11)*, pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA, 2011.

[28] J. M. Smith. *Models in ecology*. Cambridge University Press, Cambridge, MA, 1978.

[29] O. Soysal and E. Şahin. A macroscopic model for self-organized aggregation in swarm robotic systems. In *Swarm Robotics*, volume 4433 of *Lecture Notes in Computer Science*, pages 27–42. Springer, Berlin, Heidelberg, 2007.

[30] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2–3):137–162, 2004.

[31] S. Stepney, F. Polack, and H. R. Turner. Engineering emergence. In *11th IEEE International Conference on Engineering of Complex Computer Systems, 2006. ICECCS 2006*, pages 89–97. IEEE Press, Piscataway, NJ, 2006.

[32] V. Trianni, R. Groß, T. H. Labella, E. Şahin, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Proceedings of the Seventh European Conference on Artificial Life (ECAL 2003)*, volume 2801 of *Lecture Notes in Computer Science*, pages 865–874. Springer, Berlin, Heidelberg, 2003.