Towards Intelligent Ensembles

Tomas Bures[1,2], Filip Krijt[1], Frantisek Plasil[1], Petr Hnetynka[1], Zbynek Jiracek[1]

[1]Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic

[2]Institute of Computer Science
Academy of Sciences of the Czech Republic
Prague, Czech Republic

{bures, krijt, plasil, hnetynka, jiracek}@d3s.mff.cuni.cz

ABSTRACT

Recently, several ensemble-based component models have been created to address the dynamicity and complexity of designing cyber-physical systems. Experience in applying these models to actual case studies has shown that there are still scenarios in distributed organization that are hard to capture by utilizing only the concepts of these component models. In this paper, we present a summary of issues encountered, based on the analysis of selected case studies. We propose new concepts that build on those contained in ensemble-based models. In particular, we introduce the ideas of ensemble nesting, dynamic role cardinalities and ensemble fitness. These concepts and their support in the runtime framework aim at serving as a bridge between high-level ensemble formation rules and low-level decentralized implementation. These concepts are illustrated on one of the case studies, demonstrating a domain specific language based on that used in the DEECo component model.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures, Patterns;* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents*.

General Terms

Algorithms, Design, Languages.

Keywords

Distributed coordination, architectural adaptation, ensemble-based component systems, component model, emergent architecture, component ensembles, autonomic systems.

1 INTRODUCTION

1.1 Motivation

Owing to technological and research advances, as well as the increasing availability of connectivity and cheap computer components on the market, a new kind of distributed systems has emerged in the recent years. These systems are collectively referred to as cyber-physical systems (CPS) and are a manifestation of the Internet-of-things (IoT) initiative. Cyber-physical systems usually consist of a set of autonomous components that, even though capable of working in isolation, achieve their full potential by using network communication to cooperate with each other to fulfill a common goal. As evidenced by various industry-driven use cases and research calls, CPS are also of interest to the industry. In recent years, this interest is especially growing in the area of "smart CPS" (sCPS), which are CPS with high-level of distribution, coordination, autonomy, and self-adaptivity.

While traditional software development practices, tools and architectures have been shown not to be very suitable for developing sCPS, several novel concepts and approaches have been proposed to better address the inherent dynamicity of the environment the sCPS usually operate in [1, 2, 3]. Out of these, a very promising one is the concept of autonomic component ensembles (i.e. dynamic cooperation groups of components), elaborated in the ASCENS project [4] and embodied in the ensemble-based component systems (EBCS) [5] – referred to as *contemporary EBCS* further in the text.

Nevertheless, experience with case studies of sCPS designed within the ASCENS project [4] and their realization in various ensemble-based languages and component systems of sCPS (jRESP [6] implementation of the SCEL [7] language, component models Helena [8] and DEECo [5, 9]) has shown that the expressive power of these novel approaches is limited. Specifically there has been lack of the ability to capture mutual awareness and complex coordination among components and various limits imposed by domain constraints (very often involving the limited ability and range of communication).

In addition, sCPS-related coordination problems are tackled in other fields as well: Multi-agent and robotics approaches such as the various coalition-based methods [10] often either disregard dynamicity (in terms of anytime component failure), or are tailored to particular problems (e.g. comprehensive solutions of the RoboCup Rescue disaster management competition [11]) and therefore lack generality and unified concepts. Promising solutions for smart spaces such as A-3 [12] and its extension SeSaMe [13] have emerged in the recent years, but depend heavily on stable network infrastructure that is often absent in the sCPS domain. All in all, these approaches suffer from similar problems we have faced in the case studies.

1.2 Goals and Structure of the Paper

The primary goal of this paper is to present the lessons learned from the ASCENS case studies and to propose novel concepts to address the issues encountered. We do so in the context of

ECSAW '15, September 07 - 11, 2015, Dubrovnik/Cavtat, Croatia
 ISBN 978-1-4503-3393-1/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2797433.2797450

autonomic component ensembles, which we consider as having a strong potential in architecting sCPS.

The goal is reflected in the structure of the paper as follows: Section 2 describes the ASCENS case studies and highlights which of their features are hard to capture using the contemporary EBCS concepts. Section 3 presents the key contribution of the paper – it proposes novel concepts and rules for ensemble formation; for this purpose it employs an extension of the DEECo DSL specification language. Finally, Section 4 concludes the paper.

2 LESSONS LEARNED FROM CASE STUDIES

2.1 Cloud Infrastructure Management

The Cloud Infrastructure Management case study is focused on load balancing in a cloud platform (physical machines vs. workloads). The goal is to distribute the workloads in such a way that the utilization of each machine is within given bounds while, at the same time, the number of machines running is kept as small as possible. In terms of contemporary EBCS concepts, it is natural to model both the machines and workloads as components and the assignment of a workload to a machine by participation in an ensemble. A natural requirement is that membership in such ensembles needs to be optimized in terms of the performance of the cloud. Moreover this optimized membership assignment has to be stable, since several factors negatively influence the performance; specifically: (i) Once the decision is to switch off a machine, it would require several minutes to boot up again if need be; (ii) fluctuation of the workload between machines also degrades the overall performance.

However, neither the optimized membership in the ensembles, nor the cost of taking a decision can be directly modelled via the contemporary EBCS concepts of component and ensemble. A key reason is that at least a partial mutual awareness of ensembles is required.

2.2 E-mobility

In this case study, the goal is to model smart cars attempting to find a parking place in the streets of a busy city. To be able to secure a parking slot, the cars have to book in advance. In order to do so, there has to be a coordinating entity for each street. In this case, it is assumed that a car already parked in a street serves as the local coordinator of parking bookings; this removes the need for deploying dedicated hardware components for this task. Mapping to EBCS concepts is straightforward: the cars as components, plus the cars in a specific street forming an ensemble. It is however hard to capture the coordination constraints related to the environment – e.g. permitting per street only a single ensemble representing the parked cars among which exactly one dynamically assumes the responsibility of parking reservation coordination.

2.3 Search and Rescue

The Search and Rescue case study considers a geographical area that has been afflicted by a disaster, such as an earthquake. It is necessary to secure all the buildings in this area using a group of robots. To make sure people in all buildings are found and rescued as soon as possible, it is required that the robots after being paradropped remain deployed to the buildings in a balanced way. Modelling this case study via contemporary EBCS concepts is straightforward at first sight – robots are represented as components. Nevertheless, as to ensembles, it would appear natural to group all robots in the area to allow coordination of the whole rescue mission and further sub-partition the group to form an ensemble per building (to facilitate coordination between robots in the same buildings) and optimize ensembles in such a way that all of them have "similar" number of robots. However, such "fair" distribution of robots does not map easily to any of the contemporary EBCS concepts.

2.4 Summary of Issues

As shown in Sections 2.1, 2.2, and 2.3, the component and ensemble concepts of contemporary EBCS (as featured by DEECo [5, 9]) are insufficient to explicitly model specifically the following:

a) To capture systems that are inherently hierarchically structured and/or organized (Sect. 2.3).

b) Associating the existing ensemble instances with domain entities (Sect. 2.2, 2.3), consequently limiting the number of the instances.

c) To dynamically select particular members in order to optimize functionality of the system in terms of achieving a fair, stable, and cost-reducing distribution of particular components among several ensembles (Sect. 2.1, 2.3).

d) To represent the cost of making a bad coordination decision (Sect. 2.1).

It is of course possible to find a way of getting around these insufficiencies in the application code and/or by introducing a specific component(s) maintaining the global knowledge required for coordinating activities of other components. However, formation of ensembles guided by complex application-specific constraints appears to be a very common requirement in sCPS. It would be therefore advantageous to define these constraints declaratively while designing an architectural model and let the runtime framework form ensembles based on these constraints. Any runtime implementation should, however, adhere to the principles already contained in contemporary EBCS – loose coupling, decentralized execution and opportunistic knowledge propagation. All in all, what is required is to enrich the contemporary EBCS concepts by high-level architectural constraints (such as contextually limiting the number of members of an ensemble and of instances of an ensemble type). Our goal can therefore be seen as an effort to bridge the gap between high-level declarative system design and decentralized sCPS-suitable implementation.

3 INTELLIGENT ENSEMBLES: CONCEPTS INTRODUCTION

In this section, we elaborate on the example stemming from the Search and Rescue case study (Sect. 2.3). We outline how a high-level architectural description and enriched semantics of ensembles (*intelligent ensembles* further in the text) can look like. Specifically, we outline a solution to the issues (a) – (c) of Section 2.4. We do so by presenting it in an extension of the EBCS DSL designed for the DEECo (Figure 1). The issue (d) along with exact formal semantics and low-level realization of the solution shapes a promising future work.

Below we focus on the example in Figure 1. Here every rescue robot is an instance of the component type `Robot`. It features the role `r_Searcher`, which defines the knowledge of the robot

obtained from or shared with other robots. The role thus acts as an interface to the component's knowledge.

All the `Robot` components have the same `areaId`, which identifies the deployment area and `s_buildings`, which is the set of buildings in the area to be explored. Further, every robot maintains information on its geographical position (represented in the knowledge of `Robot`).

Robots are grouped into ensembles, the instances of which are dynamically created by the runtime framework. The instance of the `e_Area` ensemble type represents the geographical area where a disaster took place and comprises all robots in the area. The `e_SearchGroup` ensemble type instances (instantiated by the runtime framework per building in the area) represent the groups of robots deployed to specific buildings in the area; assuming enough robots are available.

```
1  knowledge type Coord
2    lat: double
3    lon: double
4  end Coord
5
6  knowledge type Building
7    id: string
8    where: Coord
9  end Building
10
11 component role r_Searcher
12 // role/interface of a robot in the case study;
13   areaId : string
14   s_buildings : Building[]
15   position : Coord
16 end r_Searcher
17
18 component type Robot features r_Searcher
19 . . .
20 end Robot
21
22 ensemble type e_Area
23 // representing the local area where a disaster took
    place; single instance per local area
24   id areaId : string :=
                    one_of(rs_allSearchers.areaId)
25 // for collections, dot expresses projection
26   parent of es_groups [1..*]: e_SearchGroup
27   alias
28     buildings = one_of(rs_allSearchers.buildings)
29     N = count(es_groups)
30     M = sum(count(es_groups.rs_gsearchers))
31     ratio = floor(M/N)
32 // assuming all r_Searcher.buildings are equal
33
34   membership
35     roles
36       rs_allSearchers [1..*]: r_Searcher
37     constraints
38       allEqual(rs_allSearchers.areaId)
39     fitness
40       sum(es_groups.fitness)
41   knowledge exchange . . .
42 end e_Area
43
44 ensemble type e_SearchGroup
45 // represents the robot group assigned to a specific
    building; an instance per building
46   id structureId : e_Area.buildings.id
47   child of e_Area
48   alias
49     min_members = max(e_Area.ratio – 2, 1)
50     max_members = e_Area.ratio + 2
51   membership
52     roles
53       exclusive /* membership in at most one child
      ensemble permitted via a role in rs_gsearchers */
          rs_gsearchers [min_members .. max_members]
                              : e_Area.rs_allSearchers
54       fitness
55         -sum(distance(rs_gsearchers.position,
               e_Area.s_buildings.where))
56   knowledge exchange
57 // robots exchange information on their position
58 ...
59 end e_SearchGroup
```

Figure 1: Example of proposed concepts.

Robots in an ensemble are subject to knowledge exchange (lines 41 and 56) which is a form of component asynchronous communication done periodically [5].

Hierarchical structuring. To reflect the hierarchy area – buildings, we introduce hierarchy of ensembles (enhancing the flat ensemble architecture of contemporary EBCS). Specifically, an instance of `e_Area` contains (potentially) multiple instances of `e_SearchGroup`.

Syntactically, nesting is introduced in the ensemble type definition by the constructs starting with the keywords **`child of`** (line 47) and **`parent of`** (line 26). The entities of the parent ensemble are accessible in a child ensemble via composed names (dot notation, e.g. line 46).

Creating an instance of a parent ensemble type implies creation of instances of its child type(s), the number of which is determined by the domain of the child type identifier. For example, creating an instance of `e_Area` implies instantiating the set `es_groups`.

Associating ensembles with domain entities. Inherent to the example is the need to represent the communication of the robots deployed to a particular building (not to explicitly represent the building itself as a component). This means that we should model building as domain concept at the architectural level by associating it with an ensemble representing the communication.

To do so, we propose the **`id`** construct, which defines the domain of corresponding ensemble instances. For example, the specification on line 24 means that the identification of an instance of `e_Area` is a string and its actual valuation is determined as the `areaId` of the robots deployed to the area. Advantageously, the cardinality of the identifier domain (and range of its initial values) determines also the maximum number of instances of the ensemble type. While on the line 24 the valuation of the identifier is explicitly defined, on the line 46, the valuation is determined by the underlying framework for the domain `e_Area.buildings.id`. Again, this also determines the maximum number of instances of the ensemble type `e_SearchGroup`.

Optimized membership. An important issue of the Search and Rescue example is to deploy robots to buildings in an "optimal" manner, here based on geographical distance to a building. To address this issue, we reuse the concepts of membership constraint of contemporary EBCS and propose to combine it with the concepts of dynamic role cardinalities and ensemble fitness function. In this proposition, membership of a component in an ensemble is determined by a) its binding to a role of the same type offered by the ensemble, b) satisfaction of the ensemble constraint, and c) decision of the fitness function specified in the ensemble. These features of the ensemble are specified as follows:

Roles – As the rule of thumb, at most one component can be bound to a specific role of the ensemble. Syntactically, the roles an ensemble offers are specified as role sets. Such a specification is introduced by the keyword **`roles`** followed by role set definitions, each consisting of (i) identifier of the role set, (ii) role set cardinality, and (iii) type of role. For example on the line 36 the role set `rs_allSearchers` has the cardinality `[1..*]` (no

upper bound) and each role in the set is of type `r_Searcher`. More interestingly, the role set `rs_gsearchers` (line 53) has the cardinality `[min_members .. max_members]`, the bounds of which are dynamic as determined by the expressions on lines 49 and 50. Furthermore, the role type specification refers to the parent role set `e_Area.rs_allSearchers` – this indicates that a member component to be bound to the role in `rs_gsearchers` also has to be bound to a role in `e_Area.rs_allSearchers`, i.e. to be also a member of the parent ensemble (the type of the latter also defines the type of the former).

Constraints – A constraint is a predicate over elements of role set(s) expressing a necessary condition for a component to become a member of the ensemble. For example `allEqual(rs_allSearchers.areaId)` means that the `areaId` value of the components bound to the roles in `rs_allSearchers` have to be equal (line 38); i.e. all the robot components in this ensemble have to represent robots deployed to the same area.

Fitness Function – A fitness function captures a soft optimization rule, such as the requirement of a fair robot/building distribution. In other words it is a measure representing how well a system is optimized. Syntactically, it is denoted by the keyword **`fitness`** followed by a function definition. On the lines 54-55 the fitness function evaluates the distances of the robots from the buildings in an area; the smaller the result, the better. Importantly, since this is done dynamically and the number of available roles in a role set may be limited, a better component candidate can be found to replace a member component (the "worse" one) in the ensemble.

4 CONCLUSION

In this paper we have presented intelligent ensembles, the inception of which has been motivated by the difficulties encountered when modeling several case studies via the contemporary EBCS concepts. We have outlined how to address the issues (a)-(c). As for issue (d), this requires further extension of the specification to include the cost or risk implied by a wrong decision and change of the decision.

We are currently working on a prototype of a runtime framework supporting the new concepts (enhancing the JDEECo [14] implementation of DEECo). The primary challenge is finding a form of coordination between ensembles without violating the decentralized nature of DEECo that makes it viable for sCPS applications. A promising direction in this respect are the approaches taking advantage of the "continuous" nature of sCPS, in which a temporary flaw in ensemble forming does not typically have a critical impact on the system (nevertheless reducing its performance). Of course, this is closely related to the cost and risk of a wrong decision, and to the cost and latency of communication, which in sCPS (that often employ mobile ad-hoc networks) radically grows with geographical distance.

To conclude, we are of the opinion that the newly introduced concepts increase the expressive power of contemporary EBCS, and even though more case studies are needed and full implementation support has to be elaborated, this research direction, scientifically very promising, is of high importance.

5 ACKNOWLEDGEMENT

The work on this paper has been partially supported by the Charles University Grant Agency project No. 390615 and partially supported by Charles University institutional funding SVV-2015-260222.

6 REFERENCES

[1] Hölzl M., Rauschmayer A., Wirsing M. 2008. Software Engineering for Ensembles. In Software-Intensive Systems and New Computing Paradigms, pp. 45–63. LNCS 5380.

[2] Morin B., Fleurey F., Barais O. 2015. Taming Heterogeneity and Distribution in sCPS. Proceedings of SEsCPS 2015, ACM, Italy.

[3] Ruchkin I., Schmerl B., Garlan D. 2015. Architectural Abstractions for Hybrid Programs. Proceedings of CBSE 2015, ACM, pp. 65-74.

[4] Autonomic Service Component Ensembles (ASCENS). Project, Framework Programme 7. Homepage: http://ascens-ist.edu. Accessed on 2015/05/20.

[5] Bureš T., Gerostathopoulos I., Hnětynka P., Keznikl J., Kit M., Plášil F. 2013. DEECo - an Ensemble-Based Component System. Proceedings of CBSE 2013, ACM, pp. 81-90.

[6] Java Runtime Environment for SCEL Programs (jRESP). Project, developed as part of the ASCENS project. Homepage: http://jresp.sourceforge.net/. Accessed on 2015/06/02.

[7] De Nicola R., Ferrari G., Loreti M., Pugliese R. 2013. A Language-Based Approach to Autonomic Computing. In LNCS 7542. Springer Berlin Heidelberg.

[8] Hennicker R., Klarl A. 2014. Foundations for Ensemble Modeling – The Helena Approach. In Specification, Algebra, and Software, pp. 359–81. LNCS 8373. Springer Berlin Heidelberg.

[9] Keznikl J., Bureš T., Plášil F., Kit M. 2012. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. Proceedings of WICSA/ECSA 2012, Helsinki, Finland, pp. 249-252, IEEE CS.

[10] Vig, L., Adams, J. A. Multi-Robot Coalition Formation. 2006. IEEE Transactions on Robotics 22, no. 4, pp. 637–649.

[11] Parker J., Nunes E., Godoy J., Gini M. 2015. Exploiting Spatial Locality and Heterogeneity of Agents for Search and Rescue Teamwork. Journal of Field Robotics.

[12] Baresi, L., Guinea S. 2011. A-3: An Architectural Style for Coordinating Distributed Components. In 2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 161–70.

[13] Baresi L., Guinea S., Shahzada A. 2013. SeSaMe: Towards a Semantic Self Adaptive Middleware for Smart Spaces. In Engineering Multi-Agent Systems, pp. 1–18. LNCS 8245. Springer Berlin Heidelberg.

[14] Java Dependable Emergent Ensembles of Components (JDEECo). Project, developed as part of the ASCENS project. Homepage: https://github.com/d3scomp/JDEECo. Accessed on 2015/06/08