Engineering Agent Frameworks: An Application in Multi-Robot Systems

Jérôme Lacouture, Victor Noël, Jean-Paul Arcangeli, and Marie-Pierre Gleizes

Abstract. In this paper, we present a novel development process called SPEARAF (Species to Engineer Architectures for Agent Frameworks) and evaluate its relevance to ease the implementation of Multi-Agent Systems in the context of a multi-robot project for crisis management. SPEARAF allows to build component-based architectures for agents and their infrastructure. We show the advantages of using an architecture-based process to realise an application-specific agent framework adapted to the requirements of such a system. SPEARAF gives guidelines to enables the use of architecture-oriented practices for agent implementation.

1 Introduction

The Rosace (Robots and Embedded Self-Adaptive Communicating Systems) project[1] aims at developing means to specify, design, implement and deploy a set of mobile autonomous communicating and cooperating robots and personal devices. The designed system has to be safe, to enable self-healing, to achieve a set of missions and to self-adapt in a dynamic environment. The main case study considers a crisis management situation with a control center, Autonomous Ground Vehicles (AGVs), Autonomous Aerial Vehicles (AAVs) and human actors all carrying mobile communicating devices. Rosace promoted solution is the design of a Multi-Agent System (MAS) to manage the self-adaptation/self-management of the robots collective activities, where each actor (AAV, AGV, control center, human actor...) is represented by a software agent. The project intends to implement and compare the following multi-agent strategies for several scenarios: Adaptive Multi-Agent System (AMAS) [4], Bonnet-Torres and Tessier approach [4], and Gascuena *et al.* model [4]. In this paper, we focus on the AMAS strategy.

Jérôme Lacouture · Victor Noël · Jean-Paul Arcangeli · Marie-Pierre Gleizes
Université de Toulouse, Institut de Recherche en Informatique de Toulouse,
118, route de Narbonne, 31 062 Toulouse Cedex, France
e-mail: `firstname.lastname@irit.fr`

[1] `http://www.irit.fr/Rosace,737`

Because the development of such systems is complex and time-taking, we propose SpEArAF, a development process, to design the different entities (AGVs, AAVs...) using component-based software architectures. It enables us to ease the development, produce reusable artifacts and easily reuse them, as well as clearly design architectures supporting evolution while taking into account the requirements from the different stakeholders involved in the project.

In this paper, we focus on a simplified scenario in order to illustrate SpEArAF: AGVs with local perception have to rescue victims scattered in a forest on fire. Information about the existence and location of victims arrives dynamically and the rescue tasks must be self-allocated in a distributed manner by the fleet of AGVs to adapt to the disturbances (fire spreading, evolution of AGV tasks priorities, rescue team reorganisation, communication or material breakdowns, etc.). Based on these functional requirements, our objective is to build a system to implement and compare the strategies to solve this problem. We want to run simulations and take measures w.r.t. a set of metrics to test the feasibility, the consistency and the performance of the strategies in a virtual world before deploying it in real conditions (*i.e.* with real AGVs). A demo of the final system illustrating this scenario with 5 AGVs and 18 victims can be found on the website of the project.[2]

A constraint of the project is to use the provided simulator, called Morse[3], also built in the context of the Rosace project. It provides means to connect a program to a simulated AGV and control all its functions. We also have identified, with the stakeholders of the project, the following non-functional requirements for this system: 1. extensibility and reuse: build a prototype for one strategy that can be improved and extended with new strategies; 2. portability: minimise the effort to port the different strategies from the Morse simulator to real AGVs; 3. abstraction: provide abstract and high-level mechanisms to enable the developers of the strategies to focus on their expertise domain (functional concerns).

In the following, we discuss related works before presenting the process and applying it for our use case. Then we evaluate the process and conclude.

2 Related Works

In this section, we focus on existing approaches that can help the design of MAS with similar properties than the studied use case. Existing design methods [4] do provide guidance to develop MAS by identifying agents and environment as well as designing their behaviours and interactions. Their objectives are to design the functionality of the system and to realise the system itself. Other works such as [2] or Malaca [1] proposes to use component-based and aspect-oriented software engineering to build agent architectures and their crosscutting concerns. In the robotic community, several works tackle the building of robot architectures using components, such as YARP, OROCOS, Orca or more recently GenoM3.

[2] `http://www.irit.fr/Rosace,1196`
[3] `http://morse.openrobots.org`

In most of these approaches, we are missing a way to build specific types of agents for the specific problem and application we are building, without resting on a generic agent model such as the FIPA one. Moreover, we want to be able to build specific architectures depending on the chosen strategy instead of using a predefined one such as goal-oriented or subsumption architecture. The other approaches that enable reuse and extensibility doesn't give the possibility of abstracting specific types of agents: they at most focus on the component and mechanisms level, if not only the behaviour level.

3 Engineering Architectures for Species of Agents: SPEARAF

In this section, we present SPEARAF (Species to Engineer Architectures for Agent Frameworks), a development process based on [5].

To complete existing design methods that focus on designing the functionality of a MAS, SPEARAF promotes the engineering of application-specific frameworks for the development of multi-agent applications. It enables to take into account the non-functional requirements expressed by the developers of a MAS in order to help them to focus on the functionality of the system. Such frameworks provide what we call "species of agents": **species define sets of agents with common structural characteristics**. In the context of Rosace, there can be a species of agents representing AGVs and another one for those representing AAVs. Also, agents in the different strategies presented Sect. 1 differ from their individual behaviours but share the same structural elements such as GPS, camera or radio that makes them member of the species of AGVs. **A subspecies is a species that derives from a parent species after evolution** (*i.e.* refinement and possibly modification). By defining species, the idea is to provide specific types of agents that fit functional requirements. Developers can rely on species both when designing and implementing a MAS, they don't need to deal with operational concerns and can focus on the agent's functional behaviours.

Species of agents are realised by component-based software architectures and building them happens in two steps: 1. identifying a species of agent for the application and 2. assembling and reusing software components in architectures to create a framework for the species. Moreover, we differentiate two roles in the process: a) creation of the framework by the **framework developer** and b) use of this framework to develop the MAS by the **framework user**. Indeed, when programming the MAS, *hotspots* in the frameworks can be instantiated (possibly with sub-architectures) by the framework user to specify the behaviour of the agents using a set of agent-oriented and application-specific programming primitives defined by the framework developer. In practice, the architectures are defined using the MAKE AGENTS YOURSELF[4] tool that supports SPEARAF using model-driven engineering and editors while the frameworks are implemented with JAVA.

On top of the classical objectives of architecture and component-based software engineering such as modifiability, abstraction, testability or reuse of produced

[4] `http://www.irit.fr/MAY`

artifacts (architectures or components), SpEArAF focuses on concerns specific to MAS development by providing guidelines to build the frameworks. Moreover, it enables the use of software architectures for the development of the behaviours of a produced framework, thus giving all the advantages of such practices in all the development of the whole application.

4 Application: Engineering Architecture for Species of AGVs

In this section, we detail the SpEArAF development process by applying it to the Rosace use case and we show how it enables us to architecture, design and implement a solution to the requirements expressed previously. As we will show, SpEArAF proposes to build the architecture of the species of agents but it also enables to take care of the infrastructure to execute them, however, we will not address this second point since the infrastructure is managed by the Morse simulator.

4.1 Identification of the Species

Identifying the species of agents is twofold and is based on: 1) defining the agents of the species dynamics such as their lifecycle, the way they process information and take decisions; 2) defining the high-level abstract constructs the application developer will use to define the behaviour of the agents. Here, we are interested in the species of AGVs concerned with AGV dynamics and mechanisms.

Based on the species of AGVs, we differentiate the subspecies of AMAS AGVs, concerned with decision and interaction behaviour, by defining how it uses the mechanisms available from its parent species. In the scenario, the objective of the AMAS AGVs is to plan and allocate tasks (to move to a destination to "rescue" a victim) among them. For that, each AMAS AGV follows a perceive-decide-act lifecycle and exchange messages with other AMAS AGVs by following a cooperative task allocation protocol defined using the AMAS theory (see [3]). Tweaks of the behaviour are focused on the evaluation of tasks criticality and their acceptation.

4.2 Architecture of the Species of AGVs

Based on the species presented previously, Fig. 1 shows the architecture of AGVs. In particular, the `Behaviour` component is in charge of orchestrating the internal components functions to achieve a coherent global behaviour. It gathers elaborated information from the rest of the components, makes choices, orders execution of actions, monitors results, and sends control information to relevant components when necessary. `Behaviour` is considered as an abstract component where will be implemented the specific strategy we want to test (AMAS in this paper). Other components are considered generic for the species of AGVs (and thus for all strategies).

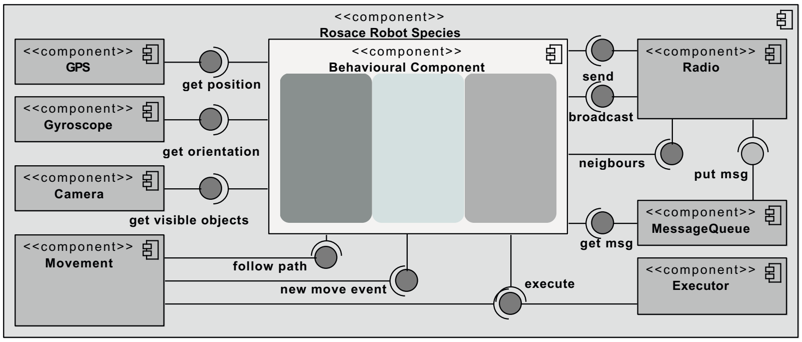
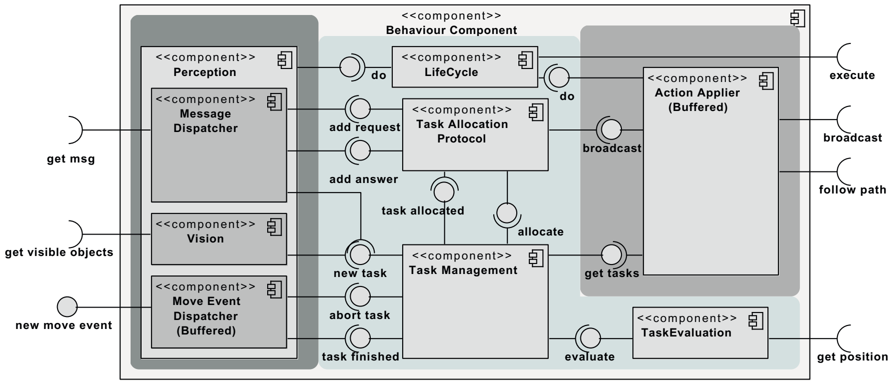


Fig. 1 Component view of the AGVs architecture (UML 2)



<<component>> Behaviour Component
<<component>> Perception
<<component>> Message Dispatcher
<<component>> Vision
<<component>> Move Event Dispatcher (Buffered)
<<component>> LifeCycle
<<component>> Task Allocation Protocol
<<component>> Task Management
<<component>> Action Applier (Buffered)
<<component>> TaskEvaluation
get msg
get visible objects
new move event
do
add request
add answer
task allocated
allocate
new task
abort task
task finished
broadcast
get tasks
evaluate
execute
follow path
get position

Fig. 2 Component view of the AMAS AGVs architecture (UML 2)

4.3 *Architecture of the Species of* AMAS AGVs

To match with the AMAS theory, the `Behaviour` component of the species of AGVs architecture is implemented with a set of components categorised into three main activities of the agent: perception, decision, action. A specification has been elicited from a description of the AMAS strategy for the victim rescue task allocation. From this specification, we were able to design the layered architecture of the subspecies of AMAS AGVs depicted Fig. 2.

At the perception level, AMAS AGVs use the components defined in the architecture of the species of AGVs in order to interpret messages and perception. It dispatches information to the decision level depending on their content. At the decision level, AMAS AGVs have to individually evaluate tasks and collectively decide to (re)allocate them. At the action level, AMAS AGVs execute tasks, broadcast messages (decisions) and forward victims location.

With `Behaviour` is seen as an architecture, the high-level abstract constructs for defining the behaviour of this subspecies are `Task Evaluation`, `Perception` or `Task Management`. This last point highlights the differences between user and developer of the framework: building the AMAS AGV subspecies is to use the AGV framework, but at the same time is to build a new specific framework.

5 Feedback and Evaluation

In this section, we briefly evaluate the produced species (architectures and framework) and SPEARAF in regards to the requirements detailed in Sect. 1.

From the AMAS specialist point of view, SPEARAF provides an easy way to implement the subspecies of AMAS AGVs. The developer doesn't need to know implementation details of the species of AGVs: indeed, specification of the interfaces and the dynamics is enough to use a specific species. In this sense, SPEARAF enable to implement MAS strategies with a common framework in order to factorise the effort and compare the different solutions. Moreover, responsibilities of components of the subspecies are clear enough to focus on improving the parameters (task evaluation and acceptation) of the task allocation protocol by a less expert developer.

Then, reuse and extensibility of species and components are clear advantages of SPEARAF. From the species of AGVs, it is easy to provide the other subspecies to implement the strategies presented Sect. 1. Consequently, the development itself produces reusable artifacts such as components (*e.g.* task allocation, vision) or species (by extension) for the development of other species, other scenarios and even other applications in robotics. An interesting example is reusing the species of AMAS AGVs to produce an species of AMAS AAVs with same goals: it doesn't need an important effort and would only consist in developing the AAVs specific components to build a species of AAVs. Moreover, using species enables to first prototype the implementation of an AGV then create a real subspecies by building a derived architecture from the parent species.

From the point of view of the framework developer, SPEARAF provides guidelines dedicated to architectures for agents by explicitly making the developer define precisely the agent dynamics, behaviour and mechanisms. Other advantages of SPEARAF for this role are not pointed up by this paper, in particular at the infrastructure level with the separation of agent concerns from environment concerns (MAS and runtime).

6 Conclusion

By relying on the SPEARAF development process, we built a system enabling the comparison in a simulator of different multi-agent strategies for multi-robot task allocation in a dynamic environment. As the evaluation shows, we were able to answer non-functional requirements needed by such a system. SPEARAF provides helps and guidelines for the development of MASs, in particular at the agent level and its programming. The concept of "species" encourages to explicitly build an architecture realising the dynamics of the agents, its interactions with its environment and enabling the definition of its behaviour through high-level abstract constructs.

Using such concepts enables the species user to build subspecies and profit from the advantages of software architecture practices till the end of the development.

References

1. Amor, M., Fuentes, L.: Malaca: A Component and Aspect-Oriented Agent Architecture. Information & Software Technology 51(6), 1052–1065 (2009)
2. Garcia, A., Lucena, C.: Taming Heterogeneous Agent Architectures with Aspects. Communications of the ACM 51(5), 75–81 (2008)
3. Georgé, J.P., Gleizes, M.P., Garijo, F., Noel, V., Arcangeli, J.P.: Self-adaptive Coordination for Robot Teams Accomplishing Critical Activities. In: Demazeau, Y., Dignum, F., Corchado, J.M., Bajo, J. (eds.) Advances in PAAMS. Advances in Intelligent and Soft Computing, vol. 70, pp. 145–150. Springer, Heidelberg (2010)
4. Gleizes, M.P., Camps, V., Georgé, J.P., Capera, D.: Engineering systems which generate emergent functionalities. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS (LNAI), vol. 5049, pp. 58–75. Springer, Heidelberg (2008)
5. Noël, V., Arcangeli, J.P., Gleizes, M.P.: Between Design and Implementation of MAS: A Component-Based Two-Step Process. In: EUMAS 2010 (2010)