An Illustrative Example of the JADEL Programming Language

Eleonora Iotti[1], Federico Bergenti[2] and Agostino Poggi[1]
[1]*Dipartimento di Ingegneria e Architettura, Università degli Studi di Parma, 43124 Parma, Italy*
[2]*Dipartimento di Scienze Matematiche, Fisiche e Informatiche, Università degli Studi di Parma, 43124 Parma, Italy*

Keywords: Agent-oriented Programming Languages, Agent-oriented Software Engineering, JADE.

Abstract: This paper presents a case study intended to investigate the features of JADEL, an agent-oriented programming language designed to ease the development of JADE agents and multi-agent systems. The paper first motivates the need for JADEL, and it briefly shows the main features of the language. Then, a well-known problem originally designed to assess the features of actor-based programming languages is recalled, and a possible solution implemented in JADEL is presented. The proposed solution is intended to validate the features of the language that concern concurrency and distribution, and it can be used as a guideline to use JADEL to target problems expressed in terms of agents that cooperate to bring about joint goals.

1 INTRODUCTION

JADE (*Java Agent DEvelopment framework*) (Bellifemine et al., 2005) is an agent platform that enables the development of distributed multi-agent systems by means of a specific Java development framework, which is used to access a dedicated runtime support. Nowadays, JADE is one of the most popular frameworks to develop and deploy real-world multi-agent systems, even if its first release dates back to the early 2000s. JADE is widely employed both for industrial and academic purposes, and it is generally recognised as a solid and reliable tool (Kravari and Bassiliades, 2015) that can follow programmers from early prototyping to mission critical deployments, as discussed, for example, in (Bergenti et al., 2003) and (Bergenti et al., 2015).

Many factors contributed to the success of JADE, both in academia and in the industry (Kravari and Bassiliades, 2015). First, it offers a mature and well-documented development framework fully written in Java. In the early days of JADE, Java was a novel and promising technology, and developers wanted to use it for their applications. Moreover, Java was particularly relevant in the scope of Web technologies, and it contributed significantly to the rapid growth of the Web. For these reasons, Java seemed to be a perfect solution for a novel framework like JADE. Then, JADE is currently developed and maintained, and a number of significant extensions have been proposed in the last fifteen years. The two main project related to JADE from the core group of researchers that originally proposed JADE are *WADE* (*Workflows and Agents Development Environment*) (Bergenti et al., 2012) and *AMUSE* (*Agent-based Multi-User Social Environment*) (Bergenti et al., 2013; Bergenti et al., 2014). WADE allows JADE to execute agents designed according to the workflow metaphor, and it provides additional features regarding fault tolerance and load balancing. AMUSE offers tools to use JADE to support multi-user interactions in applications like, for example, multi-player games. Finally, it is worth recalling that one of the factors that contributed to the success of JADE is that it complies with *FIPA* (*Foundation for Intelligent Physical Agents*) specifications, which promoted the adoption of JADE in the industry.

Besides the constant appreciation from a large community, JADE has been facing in the last few years a slow decay of popularity among developers of multi-agent systems. One of the most important reasons for such a loss of interest is that today the 100% *pure Java* approach is less appealing than it was in the past. This is mainly motivated by the growth of valid alternatives to Java that are becoming popular among programmers, like scripting languages and *DSLs* (*Domain-Specific Languages*) (Fowler, 2010). Another important reason for the loss of interest in JADE is that the *AOP* (*Agent-Oriented Programming*) paradigm is inherently different from the *OOP* (*Object-Oriented Programming*) paradigm, and such a difference is reflected in some controversial aspects of the JADE development framework. In parti-

Iotti, E., Bergenti, F. and Poggi, A.
An Illustrative Example of the JADEL Programming Language.
DOI: 10.5220/0006659402820289
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 1*, pages 282-289
ISBN: 978-989-758-275-2

cular, the difference between the AOP and the OOP programming paradigms forces AOP programmers to deal with low-level details that they do not perceive as important, while it forces OOP programmers to adopt programming idioms that fail to meet their expectations. In addition, the complexity of the JADE development framework has been consistently increasing because of the continuous growth of the JADE community and of related project, and such a growth of complexity is immediately perceived by programmers. Finally, the loss of interest in FIPA caused by the shift of industrial focus to other technologies that enable the effective development and deployment of interoperable systems contributed significantly to the loss of interest in JADE.

All mentioned problems have been discussed and analysed within the JADE community and among JADE contributors, and several solutions have been proposed. Among such proposals, *JADEL*, which stands for *JADE Language* (Bergenti, 2014; Bergenti et al., 2017), was proposed primarily to try to bridge the gap between the AOP-inspired approach that JADE promotes, and its current OOP incarnation. JADEL is a DSL for JADE agents and multi-agent systems that aims at simplifying the use of the framework with no loss of effectiveness and applicability. JADEL reinterprets the JADE agent model in terms of a simplified language that features selected AOP abstractions, like agents and behaviours, together with a dedicated support for messaging and interaction protocols. All such features contribute to lift the level of abstraction that the programmer perceives, and they make JADEL a tool to program at a higher level of abstraction with respect to the level of ordinary OOP programming. The programmer is no longer requested to master the details of JADE because such details are securely embedded in the language.

The core features of JADEL were presented in (Bergenti et al., 2016c; Bergenti et al., 2017), and the introduction of interaction protocols in JADEL was discussed in (Bergenti et al., 2016a; Bergenti et al., 2016b). JADEL has been recently used (Monica and Bergenti, 2015) to support specific experiments on the use of agents for location-aware services and applications (Monica and Ferrari, 2014a), and its current runtime support provides a module for robust localization of agents (Monica and Ferrari, 2013a; Monica and Ferrari, 2013c; Monica and Ferrari, 2014c; Monica and Ferrari, 2015a; Monica and Ferrari, 2016) in known environments using ordinary Wi-Fi infrastructures (Bergenti and Monica, 2016; Monica and Bergenti, 2016a; Monica and Bergenti, 2016b; Monica and Bergenti, 2017; Bergenti and Monica, 2017b) or specific infrastructures for accurate and robust localization based on *UWB* (*Ultra-Wide Band*) technology (Monica and Ferrari, 2013b; Monica and Ferrari, 2014b; Monica and Ferrari, 2015b).

In order to assess the actual capabilities of JADEL, it is important to validate it on complex problems and to analyse its features in terms of readability, usability and performance. In this paper, JADEL is used to develop a multi-agent system that solves a well-known coordination problem (Trono, 1994) that was stated to assess the characteristics of actor-based programming languages. The addressed problem is sufficiently complex to represent a valid case study of the features of JADEL in the management of concurrency and distribution, but it is also sufficiently simple to have it discussed and commented in the short space of a single paper. In addition, the study of a classic concurrency problem that is commonly solved using actors can help to evaluate the advantages provided by the agent-oriented approach in the context of distributed and decentralized coordination.

This paper is organized as follows. Section 2 provides a short overview of the state of the art of agent-oriented programming languages. Section 3 briefly explains the syntax of JADEL and its informal semantics. Section 4 describes the addressed case study and it shows the proposed implementation in JADEL. Finally, a concise summary of the major results of a quantitative analysis of the proposed implementation concludes the paper.

2 RELATED WORK

Agent-Oriented Software Engineering (*AOSE*) (Bergenti et al., 2004) is the discipline that studies methodologies and tools involved in the development of agent-oriented applications and services. *Agent-Oriented Programming* (*AOP*) is the programming paradigm first introduced in (Shoham, 1997) that is meant to reinterpret the development of multi-agent systems as a programming task. In this scope, *agent programming languages* are specific programming languages that turned out to be especially convenient to develop complex multi-agent systems, in contrast to more traditional, but lower-level, languages. Due to the long history of the research on agents and multi-agent systems, which lasts various decades, several agent programming languages have been designed and implemented. *AGENT-0* (Shoham, 1991) was the first example of the application of the AOP paradigm. The work of Shoham is very important because it affected most of the subsequent developments of agent programming languages, thus opening a wide and promising research area. PLACA (Tho-

mas, 1995) can be considered the direct descendant of AGENT-0. Just like AGENT-0, PLACA has an experimental nature and it was not meant for practical use. Successively, *Concurrent MetateM* (Finger et al., 1993) was proposed to use temporal logics to deal with concurrency in multi-agent systems. Similarly, *AgentSpeak(L)* (Rao, 1996) was introduced to support formal reasoning on multi-agent systems based on *BDI* (*Belief-Desire-Intention*) agents. Only recently, *Jason* (Bordini et al., 2007) was proposed to provide practical support for AgentSpeak(L). Many other languages were developed in the scope of specific contexts, such as the Semantic Web, with the *SEA_L* (*Semantic web-Enabled Agent Language*) (Demirkol et al., 2012) language, or BDI-based autonomous robots, with the *PROFETA* (*Python RObotic Framework for dEsigning sTrAtegies*) (Fichera et al., 2017) language. One of the latest entries in the plethora of agent programming languages is *SARL* (Rodriguez et al., 2014), which is a modern general-purpose programming language. SARL is very similar to JADEL in terms of syntax, but they are very different in terms of purpose and usage, they have been developed independently, and they target different agent platforms.

3 OVERVIEW OF JADEL

JADEL is a programming language intended to support effective development and deployment of JADE multi-agent systems. JADE provides a number of agent-oriented abstractions for the construction of multi-agent systems, but JADEL selects only a few primary abstractions among them to offer to the developer a clear vision of multi-agent systems. In detail, only four main abstractions that JADE implements were chosen, namely: agents, behaviours, (communication) ontologies, and interaction protocols. Note that in the case study described in next section, only agents, behaviours and ontologies are used.

Communication ontologies are formal means to support the semantics of agent communication languages for specific problems. An ontology provides a dictionary of terms and schemas, which can be organised in a hierarchical structure. Such terms can be atomic or composite concepts, atomic propositions, or predicates. With respect to first-order logics, atomic concepts are constant terms, composite concepts represent function symbols, propositions are well-formed formulas, and predicates are first-order logic predicates. Schemas describe terms at runtime, and they are used to validate inbound and outbound messages. A JADEL ontology with a proposition, a concept, and three predicates is shown below.

```
ontology ChameneosOntology {
  concept Color(string description)
  predicate MeetingCount(integer count)
  predicate Meet(aid chameneos,
    Color color)
  predicate Change(Color color)
  proposition Terminated
}
```

Behaviours can be considered as agent tasks. Each behaviour encapsulates an action, and it manages a list of inbound events. JADEL provides a specific syntax to declare, activate and deactivate behaviours, and it also offers specific linguistic constructs to manage actions and events. An example of a behaviour that handles the reception of messages and that sends replies is shown below.

```
cyclic behaviour ChangeColor
for ChameneosAgent {
  on message msg
  when {
    content is Change
  } do {
    extract msgcontent as Change

    theAgent.color = msgcontent.color

    send message {
      performative is INFORM
      receivers are #[msg.sender]
      content is new Meet(theAgent.AID,
          theAgent.color)
      ontology is ChameneosOntology
    }
  }
}
```

Agents in JADEL use ontologies and behaviours, and they take roles in interaction protocols, as discussed, for example, in (Bergenti et al., 2003). JADEL aims at making the declarations of agents clearer than semantically equivalent declarations made with JADE by means of a lighter syntax, and by means of a focus on the connections of the agent with other abstractions. The declaration of an agent is allowed to extend the declaration of another agent, with the usual semantics of inheritance, and two event handlers are provided to support initialization and take-down phases. Behaviours are normally activated in initialization and take-down phases to ensure that agents can react to interesting events, like the reception of a message. An agent that uses the `ChameneosOntology` shown previously and that activates two behaviours upon creation is shown below.

```
agent ChameneosAgent(
  String description)
uses ontology ChameneosOntology {
  var Color color

  on create {
    color = new Color(description)

    activate behaviour
      ChangeColor(this)
    activate behaviour
      WaitForMeeting(this)
  }
}
```

A description of the expressions that JADEL offers to the programmer for the construction of program statements is omitted because of space constraints. Next section provides examples of relevant expressions that should be self-exaplanatory.

4 THE SANTA CLAUS COORDINATION PROBLEM

This section describes a JADEL implementation of a multi-agent system that solves the Santa Claus coordination problem, which is a classic problem first introduced in (Trono, 1994). In its simplest form, the problem can be expressed as follows. Santa Claus has nine reindeer and ten elves. He sleeps while he waits for a group made of all reindeers or of three of his elves. When the group of reindeer is ready and Santa Claus is awake, they work together to deliver toys. On the contrary, when three elves are ready and Santa Claus is awake, they discuss about building new toys. Note that when both groups are ready at the same time, Santa Claus gives priority to the group of reindeers. In addition, the problem could be extended by incrementing the number of Santa Clauses, elves and reindeers, thus opening a wide variety of other problems. For example, reindeers and elves must help one Santa Claus at a time, and Santa Clauses cannot wait for too long for a group to get ready, so that they could decide to release reindeers and elves when groups do not get ready in time. All such extensions are not discussed in the proposed implementation because they would require the nontrivial addition of timing constraints among distributed agents.

The Santa Claus coordination problem is a classic exercise that focuses on concurrency and parallelism. A number of different solutions to the problem are available, and the main challenge that it puts forward is about the management of groups and of communication patterns. In the proposed JADEL implementation, the Santa Claus problem is solved by defining three types of agents, namely `Santa`, `Reindeer`, and `Elf` agents. Such definitions are sufficient for the underlying JADE platform to distinguish among the three agent roles in the problem.

Agents in the implemented multi-agent system can exchange messages, and the following communication ontology is provided to support the validation of the structure of messages.

```
ontology SantaOntology {
  predicate ElfReady(aid id)
  predicate ReindeerReady(aid id)
  proposition OK
}
```

The `SantaOntology` defines the proposition `OK`, which is used by Santa Claus when one groups becomes ready to inform group members that the work is about to start. The two other predicates (`ElfReady` and `ReindeerReady`) relate an *AID* (*Agent IDentifier*) to the type of the agent that sends them (elf or reindeer, respectively). The AID of an agent in JADEL (and in JADE) is unique to the platform and it consists of a name (chosen when the agent starts) and an address (the address of the machine where the agent executes). Reindeers and elves send inform messages that use such predicates to let Santa Claus know about their AIDs and their types. In the proposed solution, the ready message is sent from an elf or a reindeer to Santa Claus using the following behaviour.

```
oneshot behaviour Ready(AID santa,
    Predicate ready) {
  do {
    send message {
      performative is INFORM
      content is ready
      receivers are #[santa]
      ontology is SantaOntology
    }
  }
}
```

The behaviour is declared `oneshot`, and it contains a procedure in the `do` block. When an agent activates a `oneshot` behaviour, it adds the behaviour to its list of active tasks, and, during its life-cycle, it tries to execute all such tasks sequentially. Obviously, if a behaviour must wait for a message, and no messages are present in the agent mailbox, the behaviour is blocked and the agent does not execute its action. Generic `oneshot` actions can be performed without triggers, and, after their executions, they are removed from the list of active tasks. In this specific case, the behaviour `Ready` has two parameters, namely, the AID of Santa Claus and the predicate used to inform him about the availability of the agent that uses the

behaviour. This behaviour can be used by reindeers and elfs, and the `ready` predicate is meant to let Santa Claus know whether the sender is a reindeer or an elf.

A `Santa` agent must capture ready message and reply accordingly. The following behaviour shows the actions involved in the reception of an inform message whose content is a `ReindeerReady` predicate.

```
cyclic behaviour WaitForMessages
for Santa {
  on message msg
  when {
    content is ReindeerReady
  } do {
    extract predicate as
      reindeerReady

    theAgent.reindeers.add(
      reindeerReady.id)

    if (theAgent.reindeers.size == 9) {
      send message {
        performative is INFORM
        receivers are
          theAgent.reindeers.toList
        content is new OK
        ontology is SantaOntology
      }

      activate behaviour
        DeliverToys(theAgent)
    }
  }
}
```

The reception of messages requires a `cyclic` behaviour, which is a behaviour that remains in the list of active tasks of the agent also after the execution of its action. The keyword `for` is meant to specify a type of agent to which the behaviour is applicable. It is useful, for example, when the behaviour have access to internal fields of the agent. The special field `theAgent` is used inside the action of a behaviour to access the agent that is currently using the behaviour. The `on-when-do` declaration can be used to handle generic events, and it can be used also to handle incoming messages. The keyword `on` specifies the type of interesting events (messages, in this case), and the keyword `when` introduces a condition to filter events (messages, in this case). Finally, the keyword `do` allows expressing the actual procedure to be performed. The statement `extract-as` is used in the scope of an `on-when-do` declaration to extract the content of an incoming message. When Santa Claus receives nine `ReindeerReady` messages, he sends an `OK` message to all reindeers involved, and then it activates a new behaviour to deliver toys.

The reception of `ElfReady` messages can be added to the shown behaviour by simply extending the behaviour with a similar `on-when-do` declaration just below the declaration used to process `ReindeerReady` messages. Note that `on-when-do` declarations are used in the order in which they appear in the source code, so in this case they ensure that the priority of reindeer messages is higher than that of elf messages. When a reindeer or an elf receives an `OK` message, the following behaviour is activated.

```
cyclic behaviour WaitForOK {
  on message msg
  when {
    content is OK
  } do {
    if (theAgent instanceof Reindeer)
      activate behaviour
        DeliverToys(theAgent)
    else
      activate behaviour
        DiscussToys(theAgent)
  }
}
```

Finally, `Reindeer`, `Elf`, and `Santa` agent types can be defined as follows. For reindeers and elves, the declaration are very similar. They take as parameters a reference to Santa Claus agent, and they use the `SantaOntology` for exchanging messages. In their initialization phases, they activate the two behaviours shown above.

```
agent Reindeer(AID santa)
uses ontology SantaOntology {
  on create {
    activate behaviour
      WaitForOK(this)

    activate behaviour
      Ready(this, santa, new
        ReindeerMessage(this.AID))
  }
}
```

The definition of the single agent of type `Santa` in the system is very simple. It only has to sleep while waiting for groups of reindeer or groups of elves to become available for work. The source code of the Santa Claus agent is shown below.

```
agent Santa
uses ontology SantaOntology {
  var Set<AID> reindeers = newHashSet
  var Set<AID> elves = newHashSet

  on create {
    activate behaviour
      WaitForMessages(this)
  }
}
```

5 RESULTS AND CONCLUSIONS

This paper describes a JADEL multi-agent system intended to implement the scenario described in the simple version of the classic Santa Claus coordination problem (Trono, 1994). The proposed implementation uses one specific type of agent for each actor of the problem, and the interactions among agents are managed by the behaviours and the communication ontology described in the previous section. In order to assess relevant characteristics of the proposed implementation, the produced source codes are compared with their direct translations into Java source codes that use JADE. Table 1 shows a summary of such a comparison in terms of *Lines Of Code* (*LOC*s), under the assumption that one LOC contains exactly one statement, thus excluding empty lines and comments. The table shows that JADEL source codes are definitely more succinct than Java source codes that use JADE, especially for the definition of communication ontologies. In addition, the table shows the density of AOP features in source codes, which is computed by counting the LOCs that refer to AOP features, such as activation of behaviours and reception of messages. The table shows that agent-oriented features are more dense in JADEL source codes, mostly because of the specific linguistic constructs added in the language that provide succinct statements for complex tasks.

In summary, JADEL is meant to provide a light syntax to support the development of JADE agents and multi-agent systems in order to ease the adoption of the framework also from novice programmers. The effort made in carefully selecting appropriate abstractions seems to provide quantitative benefits, at least, in terms of the quantity of produced code and in terms of its maintainability. The example proposed in this paper follows other examples discussed in other papers, but this example emphasises the possibility of using JADEL to solve problems that are normally addressed using actors. JADE agents can behave as actors, and JADEL provides an effective means to develop and deploy them.

Besides improving the robustness of the tools that support JADEL, an envisaged improvement of the language regards the possibility of turning JADEL into a hybrid language in the spirit of Alma-0 (Apt et al., 1998). Such an extension woud give first class citizenship in the language to logic variables and to relevant types of constraints, like those discussed in (Bergenti et al., 2016d; Bergenti et al., 2016e; Bergenti and Monica, 2017c; Bergenti and Monica, 2017a), and it could move JADEL towards a more declarative style of programming, while remaining imperative and event-based.

Table 1: Number of LOCs and number of AOP features (in percentage over the total number of LOCs in each group of source codes) for JADEL and JADE implementations of the Santa Claus example.

	JADEL		
	Agents	Behaviours	Ontology
LOCs	19	68	5
AOP (%)	57.89	61.76	80
	JADE		
	Agents	Behaviours	Ontology
LOCs	54	237	64
AOP (%)	33.34	29.54	17.19

REFERENCES

Apt, K. R., Brunekreef, J., Partington, V., and Schaerf, A. (1998). Alma-0: An imperative language that supports declarative programming. *ACM Transactions on Programming Languages and Systems*, 20(5):1014–1066.

Bellifemine, F., Bergenti, F., Caire, G., and Poggi, A. (2005). JADE – A Java agent development framework. In Bordini, R. H., Dastani, M., Dix, J., and El Fallah Seghrouchni, A., editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 125–147. Springer.

Bergenti, F. (2014). An introduction to the JADEL programming language. In *Proc. IEEE 26th Int. Conf. Tools with Artificial Intelligence (ICTAI)*, pages 974–978. IEEE Press.

Bergenti, F., Caire, G., and Gotta, D. (2012). Interactive workflows with WADE. In *Proc. 21st IEEE Int. Conf. Collaboration Technologies and Infrastructures (WETICE 2012)*, pages 10–15. IEEE Press.

Bergenti, F., Caire, G., and Gotta, D. (2013). An overview of the AMUSE social gaming platform. In *Proc. 14th Workshop "Dagli Oggetti agli Agenti" (WOA 2013)*, volume 1099 of *CEUR Workshop Proceedings*. RWTH Aachen University.

Bergenti, F., Caire, G., and Gotta, D. (2014). Agent-based social gaming with AMUSE. In *Proc. 5th Int. Conf. Ambient Systems, Networks and Technologies (ANT 2014) and 4th Int. Conf. Sustainable Energy Information Technology (SEIT 2014)*, Procedia Computer Science, pages 914–919. Elsevier.

Bergenti, F., Caire, G., and Gotta, D. (2015). Large-scale network and service management with WANTS. In *Industrial Agents: Emerging Applications of Software Agents in Industry*, pages 231–246. Elsevier.

Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors (2004). *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Springer.

Bergenti, F., Iotti, E., Monica, S., and Poggi, A. (2016a). A case study of the JADEL programming language. In *Proc.* 17th *Workshop "Dagli Oggetti agli Agenti" (WOA 2016)*, volume 1664 of *CEUR Workshop Proceedings*, pages 85–90. RWTH Aachen University.

Bergenti, F., Iotti, E., Monica, S., and Poggi, A. (2016b). Interaction protocols in the JADEL programming language. In *Proc.* 6*th Int. Workshop Programming Based on Actors, Agents, and Decentralized Control (AGERE 2016)*, pages 11–20. ACM Press.

Bergenti, F., Iotti, E., Monica, S., and Poggi, A. (2017). Agent-oriented model-driven development for JADE with the JADEL programming language. *Computer Languages, Systems & Structures*, 50(C):142–158.

Bergenti, F., Iotti, E., and Poggi, A. (2016c). Core features of an agent-oriented domain-specific language for JADE agents. In *Trends in Practical Applications of Scalable Multi-Agent Systems,* 14th *Int. Conf. Practical Applications of Agents and Multi-Agent Systems (PAAMS 2016)*, volume 9662 of *Lecture Notes in Computer Science*, pages 213–224. Springer.

Bergenti, F. and Monica, S. (2016). Location-Aware Social Gaming with AMUSE. In *Trends in Practical Applications of Scalable Multi-Agent Systems,* 14th *Int. Conf. Practical Applications of Agents and Multi-Agent Systems (PAAMS 2016)*, volume 9662 of *Lecture Notes in Computer Science*, pages 36–47. Springer.

Bergenti, F. and Monica, S. (2017a). Hyper-arc consistency of polynomial constraints over finite domains using the modified Bernstein form. *Annals of Mathematics and Artificial Intelligence*, 80(2):131–151.

Bergenti, F. and Monica, S. (2017b). An optimization-based algorithm for indoor localization of JADE agents. In *Proc.* 18th *Workshop "Dagli Oggetti agli Agenti" (WOA 2017)*, volume 1867 of *CEUR Workshop Proceedings*, pages 65–70. RWTH Aachen University.

Bergenti, F. and Monica, S. (2017c). Satisfaction of polynomial constraints over finite domains using function values. In 32*nd Italian Conference on Computational Logic*, volume 1949 of *CEUR Workshop Proceedings*, pages 262–275. RWTH Aachen.

Bergenti, F., Monica, S., and Rossi, G. (2016d). Polynomial constraint solving over finite domains with the modified Bernstein form. In Fiorentini, C. and Momigliano, A., editors, 31*st Italian Conference on Computational Logic*, volume 1645 of *CEUR Workshop Proceedings*, pages 118–131. RWTH Aachen.

Bergenti, F., Monica, S., and Rossi, G. (2016e). A subdivision approach to the solution of polynomial constraints over finite domains using the modified Bernstein form. In Adorni, G., Cagnoni, S., Gori, M., and Maratea, M., editors, *AI*IA 2016 Advances in Artificial Intelligence*, volume 10037 of *Lecture Notes in Computer Science*, pages 179–191. Springer.

Bergenti, F., Rimassa, G., Somacher, M., and Botelho, L. M. (2003). A FIPA compliant goal delegation protocol. In Huget, M.-P., editor, *Communication in Multiagent Systems: Agent Communication Languages and Conversation Policies*, volume 2650 of *Lecture Notes in Artificial Intelligence*, pages 223–238. Springer.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.

Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., and Mernik, M. (2012). SEA_L: A domain-specific language for Semantic Web enabled multi-agent systems. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1373–1380.

Fichera, L., Messina, F., Pappalardo, G., and Santoro, C. (2017). A Python framework for programming autonomous robots using a declarative approach. *Science of Computer Programming*, 139:36–55.

Finger, M., Fisher, M., and Owens, R. (1993). MetateM at work: Modelling reactive systems using executable temporal logic. In *Proc.* 6*th Int. Conf. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-93)*.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

Monica, S. and Bergenti, F. (2015). Location-aware JADE agents in indoor scenarios. In *Proc.* 16th *Workshop "Dagli Oggetti agli Agenti" (WOA 2015)*, volume 1382 of *CEUR Workshop Proceedings*, pages 103–108, Napoli, Italy. RWTH Aachen University.

Monica, S. and Bergenti, F. (2016a). A comparison of accurate indoor localization of static targets via WiFi and UWB ranging. In *Trends in Practical Applications of Scalable Multi-Agent Systems,* 14th *Int. Conf. Practical Applications of Agents and Multi-Agent Systems (PAAMS 2016)*, volume 473 of *Advances in Intelligent Systems and Computing*, pages 111–123. Springer.

Monica, S. and Bergenti, F. (2016b). Experimental evaluation of agent-based localization of smart appliances. In 14*th European Conference on Multi-Agent Systems (EUMAS 2016)*, volume 10207 of *Lecture Notes on Artificial Intelligence*, pages 293–304. Springer.

Monica, S. and Bergenti, F. (2017). Indoor localization of JADE agents without a dedicated infrastructure. In *Proc.* 15*th German Conference on Multiagent System Technologies (MATES 2017)*, volume 10413 of *Lecture Notes in Artificial Intelligence*, pages 256–271. Springer.

Monica, S. and Ferrari, G. (2013a). Impact of the number of beacons in PSO-based auto-localization in UWB networks. In *Proc.* 15th *European Conference on the Applications of Evolutionary Computation (EvoApplications 2013)*, volume 7835 of *Lecture Notes in Computer Science*, pages 42–51. Springer.

Monica, S. and Ferrari, G. (2013b). Optimized anchors placement: An analytical approach in UWB-based TDOA localization. In *Proc.* 9th *International Wireless Communications & Mobile Computing Conference (IWCMC 2013)*, pages 982–987. IEEE Press.

Monica, S. and Ferrari, G. (2013c). Particle swarm optimization for auto-localization of nodes in wireless

sensor networks. In *Proc.* 11th *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2013)*, volume 7824 of *Lecture Notes in Computer Science*, pages 456–465. Springer.

Monica, S. and Ferrari, G. (2014a). Accurate indoor localization with UWB wireless sensor networks. In *Proc.* 23rd *IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2014)*, pages 287–289. IEEE Press.

Monica, S. and Ferrari, G. (2014b). An experimental model for UWB distance measurements and its application to localization problems. In *Proc.* 14th *IEEE International Conference on Ultra Wide Band (ICUWB 2014)*, pages 297–302, Paris, France. IEEE Press.

Monica, S. and Ferrari, G. (2014c). Swarm intelligent approaches to auto-localization of nodes in static UWB networks. *Applied Soft Computing*, 25:426–434.

Monica, S. and Ferrari, G. (2015a). A swarm intelligent approach to 3D distance-based indoor localization. In *Proc.* 17th *European Conference on the Applications of Evolutionary Computation (EvoApplications 2015)*, volume 9028 of *Lecture Notes in Computer Science*, pages 91–102. Springer.

Monica, S. and Ferrari, G. (2015b). UWB-based localization in large indoor scenarios: Optimized placement of anchor nodes. *IEEE Transactions on Aerospace and Electronic Systems*, 51(2):987–999.

Monica, S. and Ferrari, G. (2016). A swarm-based approach to real-time 3D indoor localization: Experimental performance analysis. *Applied Soft Computing*, 43:489–497.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away:* 7th *European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '96)*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55. Springer.

Rodriguez, S., Gaud, N., and Galland, S. (2014). SARL: A general-purpose agent-oriented programming language. In *Proc. IEEE/WIC/ACM Int. Joint Conferences of Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 103–110. IEEE Press.

Shoham, Y. (1991). AGENT-0: A simple agent language and its interpreter. In *Proc. of the* 9th *National Conference on Artificial Intelligence (AAAI)*, volume 91, pages 704–709.

Shoham, Y. (1997). An overview of agent-oriented programming. In Bradshaw, J., editor, *Software agents*, volume 4, pages 271–290. MIT Press.

Thomas, S. R. (1995). The PLACA agent programming language. In Wooldridge, M. J. and Jennings, N. R., editors, *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 355–370. Springer.

Trono, J. A. (1994). A new exercise in concurrency. *ACM SIGCSE Bulletin*, 26(3):8–10.