UNIVERSIDAD VERACRUZANA

CENTRO DE INVESTIGACIONES EN INTELIGENCIA ARTIFICIAL

JaCa-DDM: a framework for Distributed Data Mining based on the Agents & Artifacts paradigm

TRABAJO RECEPCIONAL EN LA MODALIDAD DE

TESIS

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE

DOCTOR EN INTELIGENCIA ARTIFICIAL

PRESENTA

M.I.A. HÉCTOR XAVIER LIMÓN RIAÑO

DIRECTOR

DR. ALEJANDRO GUERRA HERNÁNDEZ

CODIRECTOR

DR. NICANDRO CRUZ RAMÍREZ

XALAPA, VERACRUZ AGOSTO 2017

JaCa-DDM: a framework for Distributed Data Mining based on the Agents & Artifacts paradigm

By: Héctor Xavier Limón Riaño
Director: Dr. Alejandro Guerra Hernández
Co-director: Dr. Nicandro Cruz Ramírez

July 10, 2017

Contents

List of figures IV

List of tables VI

1. Introduction 1
- 1.1. Problem statement 1
- 1.2. Hypothesis 3
- 1.3. Justification 3
- 1.4. Objectives 4
- 1.5. Background 4
  - 1.5.1. Distributed Data Mining 5
  - 1.5.2. Agent-based DDM 7
  - 1.5.3. BDI agency 7
- 1.6. State of the art 10
  - 1.6.1. Agent Mining 11
  - 1.6.2. Related work 13
- 1.7. Scope and limitations 15
- 1.8. Document organization 15

I. Method 17

2. JaCa model 19
- 2.1. Jason 19
  - 2.1.1. Architecture 20
  - 2.1.2. Reasoning cycle in Jason 23
  - 2.1.3. Agent communication in Jason 26
- 2.2. CArtAgO and endogenous environments 27

3. JaCa-DDM 31
- 3.1. JaCa-DDM Model 31
- 3.2. JaCa-DDM Architecture 32
  - 3.2.1. Artifacts. 33
  - 3.2.2. Agents. 34
  - 3.2.3. Configuration 35
  - 3.2.4. JaCa-DDM Workflow 38

4. JaCa-DDM strategies **42**
4.1. Centralized strategies . . . 44
4.1.1. Centralized strategy . . . 44
4.1.2. Centralized Bagging . . . 45
4.2. Centralizing strategies . . . 47
4.2.1. Centralizing strategy . . . 47
4.2.2. Round strategy . . . 48
4.3. Meta-learning strategies . . . 49
4.3.1. Distributed Bagging . . . 50
4.3.2. Distributed Random Forest . . . 52
4.4. Windowing-based strategies . . . 53
4.4.1. Counter strategy . . . 57
4.4.2. Round Counter . . . 58
4.4.3. Parallel Round Counter . . . 59
4.4.4. GPU enhanced Windowing strategies . . . 60
4.5. Learning strategies summary . . . 68

5. Validation and analysis methods **70**
5.1. Cross validation . . . 70
5.2. Wilcoxon signed-rank test . . . 71
5.3. Forest plots . . . 72

II. Experiments **74**

6. General experiments **76**
6.1. Methodology . . . 76
6.2. Results . . . 77
6.3. Discussion . . . 80

7. GPU and large datasets **86**
7.1. Methodology . . . 86
7.1.1. Case study 1: known datasets . . . 87
7.1.2. Case study 2: pixel-based segmentation . . . 87
7.2. Results & discussion . . . 90
7.2.1. Case study 1: known datasets . . . 90
7.2.2. Case study 2: pixel-based segmentation . . . 92

III. Conclusions and future work **95**

8. Conclusions **97**

9. Future work **100**
9.1. Windowing analysis 101
9.2. Extending the JaCa distributed capabilities 103
9.2.1. Formal description 104
9.2.2. Quiting workspaces 107
9.2.3. Implementation 108
9.3. JaCa-DDM Web-based GUI 110

IV. Appendix: accepted papers **122**

List of Figures

2.1. The JaCa model 20
2.2. Types of AgentSpeak (L) terms in Jason. 22
2.3. Agent reasoning cycle in Jason 24
2.4. Graphical representation of an artifact. 28
2.5. The Agents and Artifacts meta-model. 29
2.6. Current CArtAgO environment model. 30

3.1. An overview of the JaCa-DDM architecture. 33
3.2. Contact person agent. 36
3.3. XML schema of the strategy deployment. 37
3.4. XML schema of an strategy. 37
3.5. Workflow step 1: Deployment of artifacts and agents. 39
3.6. Workflow steps 3 and 4. 40
3.7. Workflow step 5. 41

4.1. Centralized strategy sequence diagram. 45
4.2. Centralized bagging strategy sequence diagram. 46
4.3. Centralizing strategy sequence diagram. 48
4.4. Round strategy sequence diagram. 49
4.5. Distributed bagging strategy sequence diagram. 51
4.6. Distributed Random Forest strategy sequence diagram. 53
4.7. Windowing-based strategies general sequence diagram. 55
4.8. Auto-adjust method sequence diagram. 56
4.9. Counter examples gathering workflow. 58
4.10. The round counter strategy sequence diagram. 59
4.11. The parallel round counter strategy sequence diagram. 61
4.12. Alike counter examples. 62
4.13. Counter examples searching executed at each Windowing iteration. . . . 65
4.14. Parallel Counter GPU strategy sequence diagram. 66

5.1. Example of a forest plot 73

6.1. Results for accuracy and ratio of training examples used. 81
6.2. Suggested correlation between accuracy and number of examples used. . . 82
6.3. Correlation plot of accuracy and number of examples used. 83
6.4. Results for the ratio of process time, and ratio of generated traffic. 84

7.1. Example of a sequence of colposcopic images. 89

9.1. Correlation plot of accuracy and number of examples used from section 6.3.102
9.2. The intented view of an endogeneous environment. 105
9.3. Main screen of the new Web-based GUI for JaCa-DDM. 111
9.4. Experiment configuration wizard. 112
9.5. History results screen. 113

List of Tables

2.1. The BNF syntax of an agent program in Jason. 21

3.1. Predefined Weka based artifact types in JaCa-DDM. 34

4.1. Learning strategies summary 1. 68
4.2. Learning strategies summary 2. 69

5.1. 4-fold cross validation example. 71

6.1. Datasets used to explore generalization. 76
6.2. Experimental parameters. 77
6.3. General experiments results. 78

7.1. The properties of the adopted datasets. 87
7.2. Strategy configuration. 88
7.3. The six classes of the case study 2. 89
7.4. The properties of the case study dataset. 90
7.5. Results for the case study 1: known datasets. 90
7.6. Results for the case study 1: known datasets (continuation). 91
7.7. Results for case study 2. 92
7.8. Results for case study 2 (continuation). 93

8.1. Strong and weak points of learning strategies. 98

1. Introduction

This work introduces JaCa-DDM, an open framework to implement, deploy, and test Distributed Data Mining (DDM) strategies based on the agents & artifacts paradigm, as provided by Jason [13] and CArtAgO [74] technologies. In this chapter, the foundation of this work is presented, starting with a problem statement that highlights the main concepts and contributions of JaCa-DDM. Next, the hypothesis where this work revolves is addressed. On the following section, a justification for JaCa-DDM is presented, briefly comparing it with other solutions, and remarking its novel approach as an open framework for DDM. Then, the objectives of this work are promptly addressed. A background related to Distributed Data Mining, agent-based DDM, and BDI Agency, is introduced in order to set down the general concept where JaCa-DDM is based. Following the background, the state of the art where the concept of agent mining is introduced, highlighting main areas of research and challenges; and also, agent-based DDM related work is presented, addressing two commonly used approaches: centralizing, and meta-learning, as well as comparing the agent-based DDM state of the art with our approach. Also, the scope and limitations of this work are presented to give insights for future work. Finally, this part closes with the organization and brief introduction of each part of this document.

1.1. Problem statement

The discovery of knowledge in data bases, more commonly known as Data Mining, is a discipline that merges a myriad of diverse techniques to explore, analyze, and exploit large amounts of data, with the aim to discover patterns and significant rules that in some way lurk in the data [10]. The origin and distribution of the data that will participate in a Data Mining process is important to determine the best way to exploit it in an effective and efficient fashion. The traditional way dictates that all data must to be centralized in a single place, however, with the current ubiquity of computer networks and related services, the data pertaining to the same domain or system may be scattered among various places, which creates various challenges for Distributed Data Mining (DDM):

- Which is the best way to obtain a good learning models that takes into account the data from all places?
- Which is the best way to deal with heterogeneous data?
- Since in some applications the amount of data is significant, How can the communication of the data and Data Mining operations be optimized?

- How to deal with data privacy in sensible applications?
- How to deal with data that constantly grows and change in an efficient fashion?

All of the mentioned challenges are intrinsic to a key question: What is a good strategy to cope with a DDM scenario in an efficient and reliable way, in order to obtain the desired global result, by also taking into account some given constraints? By strategy, we mean a workflow that describes interactions between computational entities, agents, their tools and services, that may coordinate or collaborate in order to learn a model. Multi-agent systems are a natural way to deal wit distributed scenarios, since they are essentially distributed systems in their core, having all the technological foundations to allow flexible, robust, and scalable systems [56], so its adoption as a deployment technology for DDM seems intuitive, and hence its adoption. Designing a general strategy, such as mentioned before, seems to be a truly open question, without a unique answer. It is more reasonable to assume that certain strategies are better suited for some scenarios than others, given the DDM concerns. The goal of this work is to conceive a model and a tool, automatically enabling the deployment and evaluation of different DDM open strategies, allowing the developer to focus only on the definition of such strategies.

So, this work introduces JaCa-DDM[1], a novel agent-based Distributed Data Mining system founded on the Agents and Artifacts paradigm [63], and conceived to design, implement, deploy, and evaluate learning strategies that cope with DDM scenarios. Agents are modeled and implemented in Jason, a BDI agent oriented programming language that provides the right level of abstraction to describe DDM strategies in terms of events, beliefs, desires, intentions, plans, and speech acts; while artifacts, defined and implemented following the CArtAgO [75] model, resolve gracefully what Shoham [82] called "agentification", e.g., the transformation of an arbitrary device to be exploited in agent oriented programming terms. Artifacts encapsulate Weka 3 [91] objects, so that the agents in the system can create, perceive, compose, and use them. This allows the reusability of existing Data Mining tools in the strategies; while solving the interaction between agents and tools.

Also, a set of strategies for DDM are presented and evaluated, from centralizing to Meta-learning [69] approaches, highlighting a kind of new proposed strategies based on Windowing [70], a technique consisting on building an initial model with some examples, and then enhancing it by reviewing available counter examples. We study the suitability of such strategies in a distributed environment, based on the observation that Windowing reduces the number of training examples used for learning, potentially reducing data communication. We also enhance the Windowing approach through GPU computing.

The main contributions of the work are as follows:

- From the Multiagent Systems perspective, our approach illustrates how to exploit the so called "agentification" of Weka components for the sake of code reusability, while preserving the benefits of Belief-Desire-Intention (BDI) reasoning and representation provided by Jason.

[1] JaCa-DDM is available at http://jacaddm.sourceforge.net/

- From the DDM perspective, the resulting system, JaCa-DDM, an extensible tool to define and test distributed data mining agent-based strategies.
- An analysis of a set of strategies that explores centralizing, meta-learning, Windowing-based , and GPU enhanced approaches applied to distributed settings.

1.2. Hypothesis

The agent and artifacts paradigm enables the definition of a flexible framework for DDM, where it is possible to implement DDM solutions based on traditional approaches, as well as new ones, and whose performance, measured in terms of accuracy, time of convergence, and traffic load, can be compared fairly, even against centralized DM approaches.

1.3. Justification

JaCa-DDM, the proposed framework, provides:

- An open model for describing DDM processes in terms of agents exploiting artifacts that encapsulate DM tools, i.e., an agents & artifacts approach for DDM.
- A Multi-platform system for deploying and evaluating such models in a distributed environment.
- The API facilities and tools to develop new DDM workflows and artifacts based on our model.

From the best of our knowledge, agent-based DDM systems have focused on providing a kind of semi-autonomous DDM execution process, where agents try to come up with the best course of action given a predefined set of agent types and workflow, also focusing on the GUI facilities for the end user. Thus, they are end user focused systems, providing a small amount of flexibility and opportunities to change the behavior of agents and their interactions, making difficult to adapt the systems to particular concerns.

On the other hand, we propose an open approach for describing encapsulated DDM process through the description of agent behavior and their interactions with their tools, called learning strategy. To make this description as sophisticated as possible, we adopt the BDI model, which allows high level behavior and interactions between agents. We believe that this openness can bring new ideas about how to integrate agents and DDM in a vast array of scenarios with different concerns. We in fact have experienced this first hand with our proposed Windowing-based strategies discussed in this work.

It is not only desirable to have openness to try new agent-based approaches for DDM, but also a way to measure how good these approaches are in a distributed scenario. The intention of JaCa-DDM is exactly that: to have a model and framework to describe open agent-based encapsulated workflows for DDM (learning strategies), and also have a tool that takes into account distributed scenarios to test and deploy such workflows, providing an entire framework for experimentation.

With the previous description, it may be thought-out that JaCa-DDM is a developer focused system, but in fact, as learning strategies are encapsulated, end users can also benefit from previously created learning strategies, closing the gap between developers and end users, and making possible to share learning strategies.

We believe that our approach is of great value for future research on agent-based DDM, and we encourage its adoption.

1.4. Objectives

The objectives of this work are as follows:

- Create a model for describing open and flexible workflows for DDM based on the agents & artifacts paradigm, i.e.; learning strategies.
- Implement an extensible system, i.e.; JaCa-DDM, for the deployment and testing of learning strategies, also providing an API to implement learning strategies.
- Develop and test a series of learning strategies to explore traditional agent-based DDM approaches, e.g.; centralizing, and meta-learning.
- Propose and test a new approach for agent-based DDM in the form of Windowing-based learning strategies.
- Enhance through GPU computing the Windowing process to deal with large and distributed datasets.
- Test the enhanced Windowing process with large datasets, also considering a case study for pixel-based segmentation of images.

The objectives have the purpose to validate our proposed hypothesis through showing how JaCa-DDM can be of great value to propose and test new approaches for agent-based DDM, taking as a proof of concept our proposed Windowing approach.

1.5. Background

In order to assess our proposed framework, some background concepts are necessary. First, Distributed Data Mining is presented, highlighting its differences with traditional Data Mining, and focusing on concerns and challenges. Of particular interest is the introduction of Meta-learning as a way to deal with distributed settings. Next, agent-based DDM is introduced, creating a link between DDM and agent technology, drawing the way that JaCa-DDM follows as an agent-based DDM system. Finally, BDI Agency is presented, giving the fundamental concept of BDI Agent and its characteristics, also presenting historical insights of how BDI theories were created and what the BDI model for agents is about. BDI Agency is one of the core models in which JaCa-DDM is founded.

1.5.1. Distributed Data Mining

Traditionally, Data Mining algorithms assume centralization of the training data. That is, data is gathered at a single place. However, with the explosion of Internet and network based technologies, the scattering of data has become common, and so the necessity to come up with new Data Mining solutions to deal with such scenarios.

In these distributed environments, Data Mining faces two major challenges [97]: i) Data is generated so quickly, that it is not possible to process it in a time efficient manner; ii) Data is stored in several places, so the cost for centralizing them at a single place is proportional to their amount. Bandwidth limitations, and data privacy are other factors that impact data centralization.

To deal with the mentioned problems, DDM has aroused as new area to tackle Data Mining problems in distributed environments. The area has gained popularity, as the business intelligence market has grown [97].

DDM takes into account scenarios where the data is distributed or even the computation is distributed. DDM is used with various servers working in parallel, peer-to-peer networks (P2P), sensor networks, etc. Under certain circumstances, the DDM process may have privacy, communication, and/or computational resources restrictions. DDM techniques may be categorized as follows [97]:

- Central clusters VS peer-to-peer: the central cluster has a coordinator. The coordinator splits the work among several computers. This scheme is easy to use an configure, however, it is difficult to determine the optimal way to distribute work, and it suffers the single point of failure problem, if the central cluster fails the whole process also fails. For this scheme to work, it requires a stable environment, which may be more commonly found in local environments with several servers. On the other hand, P2P networks do not need a central coordinator, each node may bee seen as client and server. Each node in the network receives a fraction of work to deal with. Given that the P2P scheme is descentralized, each location has a limited view of the system, which allows global security of the system.

- Single model VS Meta-learning: in the single model approach, the data is scattered in various sites, local data may be sampled and transferred to a central site, where a single model is produced; local models may also be created and taken into account at a central location to obtain a final model. Meta-learning [69], defined as the learning from learned knowledge, has as objective the creation of a global model from widely spread and distributed data sources. In meta-learning a variable number of independent models are computed in parallel, one for each location, without the need of sharing information between sites. This method is flexible, allowing to combine different types of Data Mining models.

- Homogeneous data VS heterogeneous data: these terms may be better explained from the relational database concepts horizontal and vertical partitioning respectively. In horizontal partitioning (homogeneous) each distributed site has the same meta-data, that is, the table structure is the same, so partition happens at the record level, records are different in each site but share the same structure. Vertical partitioning (heterogeneous) on the other hand, segments each record information in several sites (or even the information may be repeated), so the structure is also distributed, the meta-data known in each site may be different. To deal with heterogeneous data in a DDM context is not only a matter of format unification, but a problem of semantic unification as well, each location may have its own way to describe the same problem, and to unify the different descriptions is a challenge.

Traditionally, DDM is approached in two ways: Centralizing data and exploiting meta-learning. Centralizing is about transmitting the available training data to a central place and from there apply traditional DM. Meta-learning is a technique that entails the creation of high level classifiers known as meta-classifiers, which integrate multiple classifiers computed separately over different training data [69]. Meta-learning deals with the problem of coming up with a global classifier from big and distributed datasets. Its objective is to compute in parallel a number of local independent classifiers from distributed data sources, applying different learning algorithms. Each local classifier is known as base classifier. Base classifiers are gathered and combined in a central location using a meta-classifier induction method, which tries to exploit the predictive power of each one. To create the meta-classifier, three techniques are generally used [69]:

- Voting: when a classification is required, each base classifier submits a vote (its predicted class), the prediction with more votes wins. A variation may be used that considers weighted votes.

- Arbitrating: an arbitrator classifier is learned from a separate and specially picked set of training examples. Arbitrating consists on using the predictions of the base classifiers, the arbitrator, and an arbitrating rule to obtain a consensus for the prediction. The arbitrating rule may be as simple as in case of a tie, use the arbitrator prediction.

- Combiners: generate meta-classifiers from meta-data (knowledge about how base classifiers behave) created form the base classifiers. An example of this variation is stacking [93].

The distributed nature of DDM, creates an additional challenge regarding its deployment technology, which may limit or empower the possibilities of the DDM process. In this work, we embrace the use of multi-agent systems, and related technologies such as CArtAgO artifacts, as a way to implement rich and flexible DDM processes (strategies) and their deployment. In the next section, the fundamental concepts related to agent-based DDM are introduced.

1.5.2. Agent-based DDM

Agent-based Data Mining, or simply Agent Mining [21], seeks to exploit the advantages of MAS in the Data Mining domain [2]. As an interdisciplinary area, it brings new opportunities and challenges: What is the interest of such efforts? What are the fundamental issues of a possible integration of agents and Data Mining tools? How are agents and Data Mining tools going to interact? What kind of tools are going to be used for this purpose? Of particular interest for us, is the case of Distributed Data Mining (DDM), defined basically as the application of Data Mining processes to distributed data sources [28]. Additional concerns present in DDM include [73]: communication and computational costs, generalization of results to all the distributed data, privacy, and heterogeneity.

Given its nature, it is easy to assume the interest in Agent Mining because of the inherent MAS advantages, including: flexibility, robustness, and scalability [56]. After all, MAS are by definition out of the box modular, scalable, decentralized and, last but not least, autonomous systems. However, we do not reduce the relevance of Agent Mining to this sense, but we argue that interaction and integration issues are central to such interest. MAS are relevant to DDM, if they provide expressive models enabling a natural and sophisticated way to describe workflows [57], in terms of high level interactions among agents and their tools. Even better if these workflows integrate existing Data Mining tools. All together, MAS can be ideal to deal with DDM scenarios [43].

JaCa-DDM uses an agency model known as BDI, which it is not traditionally embraced by agent-based DDM systems. The BDI approach can potentially be of great value in DDM, providing a rich representation for agents, which they can also reason about or talk about by means of speech acts [79]. We believe that the BDI model extends the possibilities for agent-based DDM systems, and as an open framework and model, JaCa-DDM entails the adoption of rich, flexible, and powerful technologies that enable a sophisticated approach for DDM. The fundamental concepts of BDI agency are introduced next. Also, section 1.6.2 expands more on agent-based DDM and related systems.

1.5.3. BDI agency

To better understand the terms agent and Multi-Agent System (MAS) it is better to start considering how agents relate to other types of software. First, considering functional programs. A functional program takes an input, process it, produces an output, and finishes [77]. They are called functional for the similarity they share with mathematical functions. As an extension to functional programs, object-oriented programs have aroused as a way to abstract problems on the form of objects, they represent individual and encapsulated elements that can communicate between each other through messages. There exists a wide variety of well established techniques to develop functional and object-oriented programs. Unfortunately, a variety of programs do not have a simple input-process-output operational structure. In particular, a lot of real world system need to have a reactive element, that is, they need to maintain a close interaction with their environment for a long term; they do not simple compute a process and then end. Reactive systems [55] can not be properly described from the functional and relational point of view. Reactive systems need to be described in terms of a continuous behavior. A special case of reactive system is concurrent system. A concurrent system is divided in several modules, each one being a reactive subsystem that interacts with the environment and other modules.

There is a kind of system even more complex than concurrent system, it is a kind of reactive system known as agent [13]. An agent is a reactive system that exhibits some degree of of autonomy, so certain tasks can be delegated to it, and the system by itself can determine the best way to accomplish the task.

The name agent comes from the concept of active, agents produce actions having a goal: agents are situated in an environment, so they can accomplish tasks in our behalf, it is desired that agents actively seek goals, finding out for themselves the way to accomplish them.

As agents are situated in an environment, they have to be able to perceive it (by means of sensors), and to change it though their actions (by means of actuators). Agents have to decide how their perception should be interpreted in order to act on the environment.

The agent's environment may be physical, as in the case of robots roaming around the physical world, or software-based, as with agents that inhabit an operating system or computational network. In the majority of application, agents just have partial control and perception of the environment. This is due the fact that other agents may be present in the environment, or the environment being too complex to be perceived and changed by an unique agent.

Besides being in an environment, a rational agent has the following properties [94]:

- Autonomy: it is a characteristic that may be measured in a continuous spectrum. In one extreme there are conventional computational programs, like word processors, and spreed sheets, which exhibit little or null autonomy. Everything that happens is a consequence of the interaction with the user. Such programs do not take any initiative in any sense. In the other extreme of the autonomy spectrum are human beings, which can decide what to do, in what things to believe, and which goals they want to pursue. In systems based on autonomous agents, aspects of the human autonomy are tried to be reproduced, however, it is not possible to expect a total agent autonomy currently, since the challenge is still too difficult for current technology development, a computer program in the middle of both extremes is more realistic. In other words, it is desired to delegate certain goals to agents, which then decide the best way to act in order to accomplish the given goal. Furthermore, the agent's ability to construct goals is directly related to the delegated goals. More specifically, the way the agents will act to accomplish their goals is directly related to the plans provided to the agents, that define the way in which an agent can act to accomplish goals and sub-goals.

  In its simplest form, autonomy means to be able to operate independently to accomplish delegated goals. An autonomous agent makes independent decisions about how to accomplish delegated goals, their decision and actions are under their own control and not the control of other agents.

- Proactiveness: it means to be able to exhibit a behaviour directed to accomplish goals. If a particular goal is delegated to an agent, then it is expected that the agent tries to fulfill the goal. Proactivineness discards passive agents, which never try to do something. As such, for example, objects, in the Java sense, are not agents: objects are passive, they only receive requests to execute a methods, they do something only when another object interacts with them. This also applies to other kinds of computational entities such as Web services to name another example.

- Reactiveness: to be reactive means to respond to changes in the environment. In real life, a plan rarely works without problems. A plan is frequently frustrated, accidentally or deliberately. When a human being notices that a plan is going wrong, it tries to take alternative courses of action. Some of those actions are on the reflex level, while others require to do some deliberation. To develop a system that simply responds to environment stimuli with direct reflexes is not difficult. It may be implemented as a table that maps a state of the system with an action. In the same way, to implement a system that it is purely directed to accomplish goals is not difficult either, it is what traditional computer programs do. Nevertheless, to implement a system that accomplishes an effective balance between the reactive and goal directed behavior is complicated.

- Social ability: every day, millions of computers around the world share information in a routinely manner. In this sense, to build a computer system that shows some degree of social ability is not complicated. However, the ability to exchange bytes it is not social in the desired sense for a rational agent. It is desired to have the ability to cooperate and coordinate activities with other agents, in such way that they can accomplish goals. To be able to have this kind of social ability, it is required to have agents that not only communicate in terms of bytes exchanges or method invocations, but also in terms of knowledge. That is, agents with the ability to communicate believes, goals, and plans to other agents.

On practice, it is common to find environment where various agents coexist. This is the kind of environment of interest for multi-agent systems. Each agent on the systems has a sphere of influence, this sphere determines the fraction of the environment that the agent controls, or partially controls. It may be possible that the spheres of influence overlap, this is the case when two or more agents have control over the same portion of the environment. This situation complicates the agent's work since to accomplish a desired result in the environment, they have to also consider how the other agents will act. Agents may organize through relationships between them, for example, an agent may obey another agent. Agents can have knowledge about other agents, it is also possible that an agent only knows some of the agents of the system.

The BDI architecture has its origins in the Rational Agency project by the research institute of Stanford University in the middle of the 80's. The model is an adaptation of the theory of practical reasoning, developed by Michael Bratman [15], which particularly focuses on the role that intentions play in practical reasoning. Bratman develops a planning theory of intention. Intentions are treated as elements of partial plans of action. These plans play basic roles in practical reasoning, roles that support the organization of our activities over time and socially. Bratman explores the impact of this approach on a wide range of issues, including the relation between intention and intentional action, and the distinction between intended and expected effects of what one intends. The conceptual framework of the BDI model is described in [16], also describing a specific agent architecture called IRMA.

The distinction between beliefs, desires, and intentions is as follows:

- Beliefs are information that the agent knows about their environment. This information can be imprecise and unreliable.
- Desires are possible states of the environment that the agents wants to accomplish. However, a desire does not necessarily implies that the agent will try to act in order to accomplish its desire: is just a potential influence over the agent actions. It is normal in a rational agent to have desires incompatible between each other. Desires are normally interpreted as options that the agent has.
- Intentions are states of the environment that the agent is currently pursuing through their actions. Intentions may be delegated goals to the agent, or they can be the result of options consideration: the agent may evaluate its options and chose some of them, options selected in this way become intentions. The agent starts with a delegated goal, then it considers possible options compatible with its delegated goal.

In the BDI model, agents are capable to communicate using speech acts [79], which it is a kind of higher level communication than the simple message passing of object-oriented programs. It has a variety of performative verbs to communicate facts, delegate work, ask for information, among others; which enable social communication.

1.6. State of the art

The interactions between agent technology and Data Mining have been widely explored [24, 22, 23, 21]. Research interests are bidirectional, including mutual problems, agents exploiting Data Mining, agents supporting Data Mining, adaptive learning agents, and applications. This part concerts the state of the art, which covers Agent Mining, highlighting the most important areas of research, and locating JaCa-DDM in one of them. As JaCa-DDM is intended for Distributed Data Mining, an overview of agent-based DDM is also addressed, presenting systems similar to JaCa-DDM.

1.6.1. Agent Mining

Agent-based DDM is included in a wider research area known as Agent Mining which is an emerging interdisciplinary area that integrates multi-agent systems, Data Mining and knowledge discovery, machine learning and other relevant fields [21]. The coupling of agent technology and Data Mining is driven by challenges faced by both communities.

Agent Mining is in essence the technologies, methodologies, tools and systems that synthesize multi-agent and Data Mining technologies for better addressing issues that cannot be tackled by any single technique with the same quality and performance [24]. Agent Mining brings multi-fold advantages to multi-agent systems, and Data Mining, as well as new theories, tools and applications that are beyond any individual technology [23].

Main areas of research in agent Mining

The main areas of research in agent-mining have quickly evolved. The research falls in the following categories [21]:

- Mutual problems: research problems in both areas that are equally relevant, such as representation of constraints and design of constraint-based systems and constraint-based Data Mining, the involvement of domain knowledge, knowledge representation, ontology and semantics for system design and content learning, organizational factors in conceptual modeling and system construction, social issues such as security, privacy and trust of data and policies, the role of data, and human-system interaction in interface design and modeling. One of these problems involves the relationship between multi-agent systems and Distributed Data Mining, which it is paramount in this work. An interesting example is [6] that proposes a model for integrating Data Mining with Agent Based Modeling and Simulation (ABMS) in order to improve both, so Data Mining techniques may be applied in ABMS, and ABMS may be applied to generate data in limited data scenarios for Data Mining.

- Agent-based Data Mining: agents and agent technology are used in Data Mining for different purposes. This includes agent-based Data Mining systems, as JaCa-DDM, agent-based data warehouse, agents for information retrieval, interface agents for interactive Data Mining, mobile agents for distributed Data Mining, agents for distributed learning and parallel learning, agent-based clustering, information gathering agents, and automated Data Mining using agents for mediation. Some recent examples of this kind of systems are i-Analyst [68], and EMADS [3], both conceived for agent-based Distributed Data Mining, and further discussed in the next section. Another recent example is [32] presenting an intelligent architecture of Decision Support System (DSS) based on visual Data Mining; the architecture applies the multi-agent technology to facilitate the design and development of DSS in complex and dynamic environment.

- Data mining-driven agents: seeks to improve agents and multi-agent systems through Data Mining. Research involves Data Mining for enhancing agent learning and adaptation, Data Mining for user modeling, Data Mining-based personalized agents, Data Mining for empowering recommendation agents, Data Mining-driven trading agents and trading strategies, Data Mining-based agent assistants, Data Mining for agent norm optimization, and Data Mining for agent behavior analysis. An example can be found in [85], presenting a methodology and the corresponding tool for transferring Data Mining extracted knowledge into newly created agents, Data Mining is used to generate knowledge models, which can be dynamically embedded into the agents.

- Agent learning: to integrate machine learning techniques in multi-agent systems, and to create intelligent and adaptive agents. Research involves mainly reinforcement learning for complex real world problem-solving. For example [30] presents a multi-agent reinforcement learning (MALR) approach to solve the load-frequency control (LFC) problem for a wide range of operating conditions in a multi-area power system.

- Agent Mining applications: as many references show, the integration of agents and data Mining, and the development of agent Mining are driven by broad and increasing applications in many areas for a variety of purposes. Typical applications include simulation of artificial immune systems, artificial stock markets with programmatic trading mechanisms, distributed data analysis, personalized service and recommendation in e-commerce, web personalized assistants, network intrusion detection systems, peer-to-peer services and systems, supply chain management, trading agents and systems, web and social networks, robots and games. An example of the integration between agents and Data Mining can be found in [88], where it is discussed how intelligent agents can be used to enhance Data Mining processes related to e-commerce.

Challenges and opportunities in agent Mining

The main challenges of agent Mining revolve around three main questions [21]:

- How can agents and Data Mining interact and be integrated?

- What methodologies are necessary and suitable for interacting, integrating and complementing agents and data Mining?

- What are the lifecycle, processes and work mechanisms for interacting, integrating and complementing agents and data Mining?

The key research challenges and opportunities in agent Mining that address the previous questions are the following:

- Theoretical foundation in agent Mining: involves the exploration and identification of the main problems that enable the integration between agents and Data

Mining. Challenges concern suitable methodologies for agent Mining, modeling human intelligence and roles in agent Mining systems and applications, modeling social, environmental, and organizational factors in agent Mining policies, design, rules and systems.

- Issues in agent-enhanced Data Mining: agents may be used in Data Mining to face challenges such as isolation, distribution, mobility, dynamics, various and large data sources, privacy and security of data. There are opportunities in emerging areas such as cloud computing, and Internet of Things.
- Issues in Data Mining-empowered agents: agent intelligence and MAS in general can be benefited from Data Mining, problem solving may be automated, adaptive, dynamic, inductive; enabling agent-oriented rules, policies, and processes. This can be achieved identified hidden information in a large scale of historical data and behaviour of agent networks, taking also into account the current dynamics and behaviors of the agents in the running system.
- Issues in agent learning: nowadays, this sub-area is dominated by work on reinforcement learning, specifically at the intersection between reinforcement learning and game theory. Opportunities appear in swarm intelligence, transfer learning, evolutionary learning, multi-view learning, multi-strategy learning, and parallel inductive learning.
- Agent Mining tools and applications: Data Mining applications have been used in applications such as fraud detection, risk management, and anomaly detection. These applications have gained acceptance in industry sectors, agent Mining applications are in the processes of gaining acceptance, there is a need to create effective tools, platforms and applications to support the exploration of the research issues introduced in this section.

1.6.2. Related work

Data Mining merges a wide variety of techniques intended for the exploration and analysis of huge amounts of data, with the goal of finding hidden patterns in it [91]. Data distribution is relevant for succeeding [67]. Traditionally, all data is available in a single site; but currently, it is common to face situations where data is distributed. DDM responds to the need of applying Data Mining processes over decentralized data sources [56]. From the Data Mining point of view, some issues may arise in these distributed scenarios, e.g., taking into account data from all sites; dealing with heterogeneous data; optimizing data communication; preserving privacy; exploiting distributed computational resources; and scalability to cope with changing data sources, that possibly grow constantly.

MAS are well suited to deal with such issues [44], since they are autonomous, distributed, robust, social systems by definition. Agent based DDM comprehends the methodologies, technologies, tools and systems that synthesize MAS technology, Data Mining, Machine Learning and other relevant techniques, such as Statistics and Semantic

Web, for better addressing DDM issues that cannot be tackled by any single technique with the same quality and performance [21].

Without surprise, a myriad of agent based DDM systems and frameworks [84, 53, 42, 5, 2, 37, 4, 58, 57, 98] have been proposed. Basically, all of them use one of the following two approaches to cope with DDM issues [73]:

- Centralizing: all distributed data is moved to a single site and merged together, obtaining in this way a traditional centralized Data Mining setting.
- Meta-learning: combines the results of a number of separate learning processes in an intelligent fashion [25]. Voting, arbitrating, and combining techniques can be used [69] for this purpose.

Meta-learning can be efficient and scalable, since data transmission is lower than in centralizing approaches, and it is inherently parallelized. Nevertheless, meta-learning is not as efficient as its centralizing counterparts for classifying, since the process involves various models. A major issue of meta-learning is obtaining models that represent a good generalization of all the data, considering that they have been built from incomplete local data [80]. The proposed solutions aggregate local results to estimate a global model, rather than producing a global one [4]. They include knowledge probing [38], mixture of expertise [96], Bayesian model averaging [71], and stacked generalization [93], among others.

JAM [84], BODHI [42], and Papyrus [5] are well known agent based DDM systems, reviewed in different surveys [43]. However, comparing JaCa-DDM with them is not fair, since they were developed before agent oriented programming facilities emerged. The following systems are closer in time and interests to our approach: i-Analyst [58, 57] is implemented using WADE [20], an extension of the agent middleware JADE [8] that integrates the notion of workflow. DDM processes defined as workflows are easy to configure, visualize, an execute. An API is provided with the same aim that our artifacts, to adopt new data mining algorithms. Three types of agents are provided, one for the user interface; one for checking and preparing resources; and one for learning. The workflow of the latter agent defines a Data Mining process. In our opinion, it is not clear what is the extent and limitation of the workflow definition language, particularly regarding communication and coordination. In some way, the system resembles the knowledge flow tool present in Weka [91], where it is possible to set sequential data mining workflows through graphical components. Workflows evoke JaCa-DDM strategies, however our definition language exploits the Agents and Artifacts paradigm, enabling sophisticated descriptions in a more natural way.

Another example is EMADS [2], an Agent Enriched Data Mining open framework, also implemented in JADE. The agents, contributed by a community, run in an Internet space, negotiating with each other while performing a variety of Data Mining tasks based on classification models and associative rules. Some of these agents encapsulate Data Mining algorithms; while JaCa-DDM encapsulates algorithms exclusively as artifacts, making a cleaner distinction between the agents controlling the DDM workflows and their tools. The goal of EMADS is to find the "best model" given different algorithms and data distributions. Although configurable, the actions available for the agents and the workflow are predefined. JaCa-DDM is much more extensible in this sense, since the set of algorithms and actions available to the agents can be extended by defining new artifacts, and the workflows can be freely defined in terms of Jason agent programs. SMAJL [95] learns associative rules exploiting data sampling to reduce communication. Some of the JaCa-DDM strategies presented in this paper, use a sampling technique known as windowing [70], with the same purpose.

All these systems adopt a weak notion of agency, instead of the complete Belief-Desire-Intention (BDI) approach adopted in JaCa-DDM. We believe that mature BDI theories and tools, as those displayed in Jason, are better suited for supporting DDM, given their well founded semantics for reasoning, communication, and organization, providing more possibilities to implement rich and sophisticated approaches for DDM. A preliminary concurrent, but not distributed, version of our system [50] was used to corroborate this intuition. The complete, general purpose, extensible, distributed version of JaCa-DDM that we present in this paper, is founded on the Agents & Artifacts paradigm described next.

1.7. Scope and limitations

Currently, the work has the following limitations:

- Only the induction of classification models are considering, extensions to support clustering and online-learning are envisaged.
- In JaCa-DDM, only Weka arff files are supported as data sources. Support for other formats, and even relational data base engines are planed as future work.
- Experiments where carried over stratified partitions of the original datasets, distributed in the available JaCa-DDM nodes. Testing the considered strategies when this is not the case would be of interest to evaluate their performance when facing locally overrepresented classes.
- Security and fault tolerance aspects of the JaCa-DDM platform are still on development.
- The current JaCa model raises some limitations on the form of distributed transparency, which impacts in the development of learning strategies. There exists an ongoing effort to improve distributed aspects of the JaCa model, which is discussed in our future work.

1.8. Document organization

In order to properly introduce JaCa-DDM, the rest of this work is organized as follows.

Part I introduces the related methods of this work, beginning with the JaCa model, providing formal definitions for Jason and CArtAgO, and their related concepts such as agent program, message, artifact type, and workspace. From there, JaCa-DDM is presented in detail. An abstract model of the system is presented, based on the concepts of JaCa-DDM strategy and deployment. Then the workflow to deploy a strategy in a network of distributed computer nodes is described. This includes the definitions of basic agents and artifacts available, as well as configuration issues. Next, a set of learning strategies are addressed, which demonstrates the flexibility and convenience of the system. Their primary aim is to encourage the use of our tool for developing and testing new strategies, suited to different DDM concerns. A first group of strategies exploit centralized approaches with strictly benchmark purposes. The second group of strategies addresses centralizing approaches. A third group explores meta-learning. Finally, a fourth group of strategies, exploits Windowing [70], a technique based on building an initial model with some examples, and then enhancing it by reviewing available counter examples, in this group we also introduce GPU-enhanced strategies. We study the suitability of such strategies in a distributed environment, based on the observation that Windowing reduces the number of training examples used for learning, potentially reducing data communication. At the end of this part, a presentation of the methods used for experimentation and result analysis are addressed, including cross validation, Wilcoxon signed-rank test, and Forest plots.

In part II the learning strategies presented previously are evaluated, and their yielded results are analyzed and discussed. Two sets of experiments were conducted:

- General experiments: to show, as a proof of concept, how the JaCa-DDM model can be used to implement and test a wide variety of learning strategies.
- GPU and large datasets: JaCa-DDM is tested for large and distributed datasets, in particular, using the Window-based GPU strategies described in section 4.4.4.

Finally, part III closes this work with some conclusions, and highlights future work. Also, at the end, a future work related to extending the JaCa model is presented in detail, along with the introduction of a new Web-based GUI for JaCa-DDM.

Part I.

Method

In this part, a throughout presentation of the most relevant methods that form part of JaCa-DDM and related experiments are addressed. This includes the JaCa model which is the paradigm in which JaCa-DDM is implemented, based on agents & artifacts, and concerning two technologies: Jason, a well known Agent Oriented Paradigm (AOP) language; and CArtAgO, a technology to define agent environments in terms of artifacts that provide services. As the technological foundation was provided, JaCa-DDM is presented in extend, covering its model for learning strategies and deployment, its architecture, and workflow. Next, given the learning strategy model presented previously, several learning strategies based in different concerns and ideas are defined, this includes learning strategies based on approaches such as traditional centralized, centralizing, Meta-learning, and Windowing-based. Finally, related methods used in our experiments and result analysis are addressed.

2. JaCa model

JaCa is the result of the composition of two technologies for MAS: Jason [13], and CArtAgO [74]. Jason initially stood for for "Java-based Agentspeak interpreter used with Saci for multi-agent distribution Over the Net". Since it is not based only on SACI anymore, as other infrastructures area available, it was decided the to use Jason as a proper name for the interpreter, also taken inspiration from Greek mythology. CArtAgO, on the other hand, stands for Common ARTifact infrastructure for AGents Open environments.

Jason provides the means for programming MAS. It is an agent oriented programming language that entails the BDI approach, it is based on the abstract language *AgentSpeak*(*L*) [72]. Apart from its solid BDI theoretical foundations, the language offers several facilities for programming Java powered, communicative MAS. Communication in Jason is based on Speech Acts, using a subset of KQML [33].

CArtAgO provides the means to program the environment, following an endogenous approach where the environment is part of the programmable system.

Figure 2.1, shows how Jason and CArtAgO integrate in a layered fashion, providing the means to characterize the environment in CArtAgO artifact terms, and feeding that representation to Jason agents that can also act upon the environment through artifact.

2.1. Jason

Apart from its solid BDI theoretical foundations, the language offers several facilities for programming Java based, communicative MAS, such facilities include:

- Speech acts based communication with annotations to specify the information origin.
- Annotations in plans labels which may be used for customization.
- Customizable functions for selection, trust, and customized agent architectures.
- Direct extensibility through internal actions defined by the programmer.
- Support for programming Java-based agent environments.
- An IDE and a plug-in for the Eclipse IDE to develop Jason MAS in an integrated environment that supports Java and *AgentSpeak*(*L*) editors, and proper debugging facilities (Mind Inspector tool) to inspect agent's internal state.

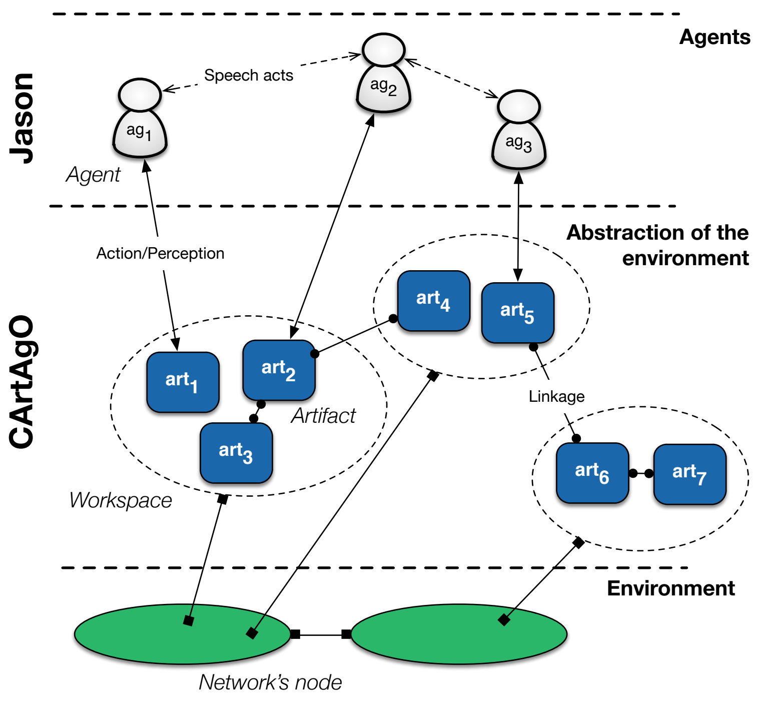


Figure 2.1.: The JaCa model.

The notion of agent is fundamental, and is the most relevant part taken from Jason in the JaCa model. An agent is defined as follows.

Definition 2.1.1 *An agent program in Jason (ag) is a set of beliefs (bs) and a set of plans (ps), as defined in Table 2.1.*

2.1.1. Architecture

There is an important distinction between agent program an agent architecture. The agent architecture is the agent framework where an agent program runs. The developer writes the program that directs the agent's behavior, but much of what the agent actually does is determined by the architecture, without the direct involvement of the developer. For example, the belief base of any agent is updated automatically.

A practical BDI agent is a reactive planification system [13]. This kind of system is designed to be constantly active, reacting to events. Actions may change the agent's environment, so agents goals are fulfilled. An agent is constantly perceiving its environment, reasoning about how to fulfill its goals, and acting to change the environment. The agent's practical reasoning is executed in accordance to its plan library developed by programmers.

To define agent behaviour, Jason has three main constructors:

$$
\begin{array}{lcll}
ag & ::= & bs \quad ps & \\
bs & ::= & b_1 \ldots b_n & (n \geq 0) \\
ps & ::= & p_1 \ldots p_n & (n \geq 1) \\
p & ::= & te : ct \leftarrow h & \\
te & ::= & +at \mid -at \mid +g \mid -g & \\
ct & ::= & ct_1 \mid \top & \\
ct_1 & ::= & at \mid \neg at \mid ct_1 \wedge ct_1 & \\
h & ::= & h_1; \top \mid \top & \\
h_1 & ::= & a \mid g \mid u \mid h_1; h_1 & \\
at & ::= & P(t_1, \ldots, t_n) & (n \geq 0) \\
 & \mid & P(t_1, \ldots, t_n)[s_1, \ldots, s_m] & (n \geq 0, m > 0) \\
s & ::= & \texttt{percept} \mid \texttt{self} \mid id & \\
a & ::= & A(t_1, \ldots, t_n) & (n \geq 0) \\
g & ::= & !at \mid ?at & \\
u & ::= & +b \mid -at &
\end{array}
$$

Table 2.1.: The BNF syntax of an agent program in Jason. Adapted from [13].

- Beliefs: they are the informational constituent of an agent. Each belief ($b_i \in bs$) is a first-order grounded atomic formulae. Atomic formulae (at), beliefs included, can be annotated. An annotation is a complex term that provides details about beliefs. By default, source annotations are considered: an agent can believe something because of perception ($percept$), self conclusion ($self$), or communication (i.e., being told by another agent id). So, beliefs change at runtime keeping the information owned by the agent updated. A **literal** is any predicate or its negation. An agent has a belief base which consists of several literals that represent information. Any symbol (sequence of characters) that begins with a lower case is an **atom** which represent objects or particular individuals. A symbol beginning with an upper case is known as **variable**, which are initially unbound, but they can be bounded through an operation known as **unification**. A **term** is any atom, variable, or structure. A **structure** may be a list, or a predicate with a functor and arguments. As in logic programming languages such as Prolog, beliefs can also be **rules**. Rules allow to conclude new information from previous knowledge, using a kind of modus-ponens logic derivation. A particularity of Jason is that rules can also use information from annotations. Figure 2.2 shows the types of terms hierarchy in AgentSpeak (L) that may apper in a Jason literal. For a more in depth treatment of first order logic see [14], [26], [51].

- Goals: this is a fundamental concept in agent oriented programming. While beliefs express properties believed to be true in the agent's environment, goals express properties of the state of the environment that the agent desires to be true. When a goal g is represented in an agent program, this means that the agent is compromised to act in such way that it will be able to change the state of the world in order to make g true in the perceived environment.

  In AgentSpeak (L) there exist two kinds of goals: **achievement goals**, and **test**

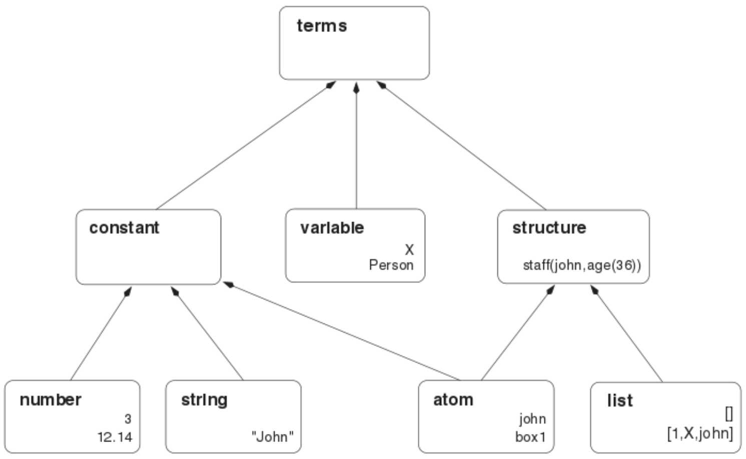


Figure 2.2.: Types of AgentSpeak (L) terms in Jason. Adapted from [13].

goals. Achievement goals use the ! operator, they instruct the agent to act upon a desired state of the environment, through the execution of the related plan which contains a series of actions. Test goals use the ? operator, and are normally used to retrieve information from the belief base, but in certain circumstances, test goals may also trigger the execution of plans.

There exists a fundamental distinction between the notion of goal in Prolog and AgentSpeak (L). A goal in Prolog is a conjunction of literals that the interpreter verifies if they can be concluded from the belief base, which essentially means that Prolog is testing if the goal is a logical consequence of the logic program.

- Plans: they are the know-how constituent of an agent. Each plan ($p_i \in ps$) has a head ($te : ctx$) and a body (h). The body is a sequence of steps to be performed by the agent, including: actions (a), subgoals (g), and belief updates (u). Updates can be additions or deletions of beliefs. Subgoals can in turn be of two types: Test goals ($?at$) that compute if an atomic formula is a logic consequence of the beliefs of the agent, i.e., $bs \models at$; and, Achieve goals ($!at$) that ultimately compute a new plan to form an intention for solving at. User defined actions can be implemented in Java, as instances of Jason predefined classes for this purpose; or as operations in artifacts, as stated in Definition 2.2.1. An empty body is denoted by $\top$.

  The head of a plan defines when it is relevant and applicable. A plan is relevant for a given event e, if its trigger event is a logical consequence of the event, i.e., $e \models te$. Trigger events include adding or deleting a goal or a belief. A relevant plan is applicable if its context is a logical consequence of the beliefs of the agent, i.e., $bs \models ctx$. That is, the context is used to verify the current situation to determine if a particular plan, among various alternatives, has any chance to be successful for the event, given the last known information about the environment. The true context, denoted by $\top$, means the plan is always applicable. A relevant applicable plan is a candidate to form an intention.

Plans can also be labeled. The label identifies the plan, and can also be annotated to use special directives for the interpreter, such as the atomic excitation of a plan, or also to allow the programmer to make special annotations to use in further custom processing. Labels of a plan must appear before the trigger event and they have to start with @.

2.1.2. Reasoning cycle in Jason

An agent operates through a reasoning cycle which, in the case of Jason, can be divided in ten main steps. Figure 2.3 shows the agent's architecture of a Jason agent and the functions that are executed during the reasoning cycle. In the figure, rectangles represent the main architectural components that determine the state of the agent, i.e.; the belief base, a set of events, a plan library, and the set of intentions. Rounded elements, and diamonds represent functions that can be modified by programmers, while circles represent essential parts of the interpreter that can not be modified. To be more precise, diamonds represent selection functions which take a list of elements and return one of them.

In what follows, each of the ten steps of the agent reasoning cycle is explained [13].

1. Perceive the environment: the fist thing than an agent does in its reasoning cycle is sensing the environment to update its beliefs about the state of the Environment. The agent architecture has a component to sense the environment following a symbolic representation such as a list of literals. Each literal is a perception, which is a symbolic representation of an environment property. The *perceive* method is used to implement the process that obtains the perceptions.

2. Update the belief base: once the list of perceptions has been obtained, the belief base needs to be updated to reflect the perceived changes in the environment. This is done through a belief base update function, the method that implement this function is called buf, the method can be customized. By default, the buf method assumes that everything that was perceived in the environment will be included in the list of perceptions obtained in the previous step, and in consequence the belief base will be updated adding new perceptions, updating the ones that changed, and deleting the ones that are not present in the list. Each change in the belief base generates an event.

3. Receive communications from other agents: the interpreter verifies messages that could be stored in the agent's *mailbox*. This operation is done by the $checMail$ method, which can be customized. The method simply takes the stored messages and returns them.

   In a single reasoning cycle, just one message is processed by the interpreter. It may be the case that in certain scenarios the messages need to have different priorities,

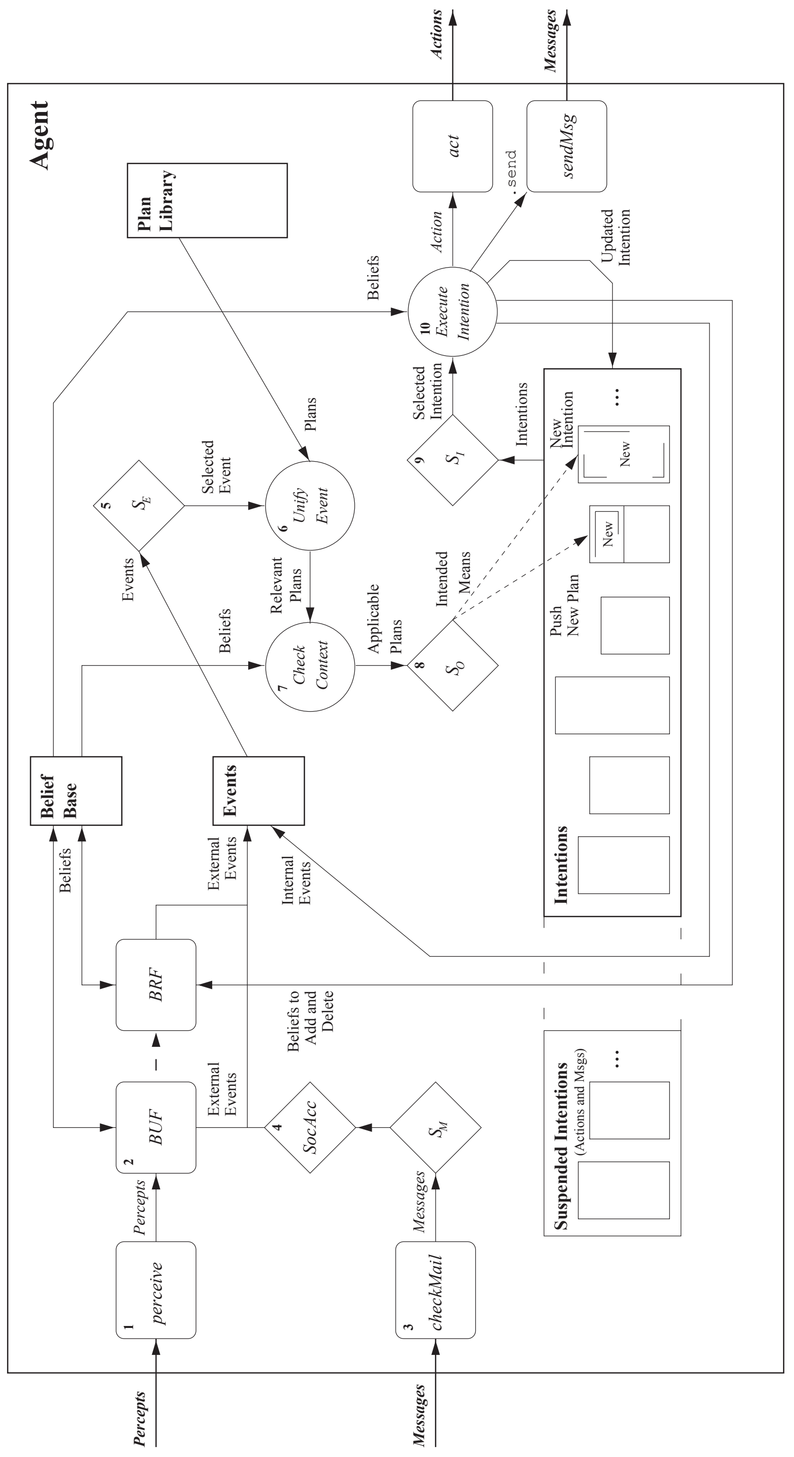


Figure 2.3.: Agent reasoning cycle in Jason. Adapted from [13].

so a selection function is needed. The selection function can be customized by the programmer, by default, the functions returns the messages in arrival order.

4. Select socially acceptable messages: before messages can be fully processed, they are processed to verify if they can be accepted by the agent. The interpreter's method that does this work is known as *SocAcc*, for socially acceptable. This method is usually customized by the programmer, taking into account the different kinds of agents, so an agent can only accept information or work from trusted agents. The default implementation simply accepts all the messages.

5. Select an event: BDI practical agents operate through continuous event management, which represent perceived changes in the environment or changes in the agent's goals. In each reasoning cycle, just one event is processed. There may be various pendant events, this may be because the environment had change recently but the agent has not executed enough reasoning cycles to process them. For the previous reason, it is necessary to select the event that will be processed in a particular reasoning cycle. This is done through an agent specific selection function that can be customized. When customized, the programmer usually wants to consider first events important for the context of the application. The default behaviour is to process the events in the arrival order, having a queue behavior, which is only recommended if there is not any special distinction between events.

6. Return relevant plans: once an event has been selected, it is necessary to find a plan that allows the agent to act in such way that it can process the event. First, relevant plans for the event are retrieved from the plan library. A plan is relevant if it has an event trigger that unifies with the selected event.

7. Determine applicable plans: plans have a context to determine if the plan can be used in a particular moment, given the information that the agent has. Even if relevant plans have been selected, it may not be possible to apply any of them for the current event. It is necessary to select, from the relevant plans, the ones that are applicable; that is, the plans that given the current beliefs of the agent, have more chances to be successful. So, it is necessary to verify if context of each relevant plan is true; that is, if the context is a logical consequence of the agent's belief base.

8. Select an applicable plan: all the applicable plans selected in the previous step are alternatives to manage the selected event. From the agent's point of view, any of those plans is equally good. The agent needs to choose one of the applicable plans and commit itself to execute it. To execute a plan means that the agent has the intention to follow the plan which in turns means that the selected plans will be included in the intentions of the agent. The selection of option function *SO* selects an applicable plan from the set of applicable plans.

   The goals that are currently in the set of events represent the different desires that the agent has available, and when one is chosen a commitment is created. On the other hand, the different applicable plans for a given goal represent alternative curses of actions to fulfill the goal. The *SO* functions, as other selection function, is customizable. The default behavior is to choose the first applicable plan from the plan library, the order is determined by the order in which appear in the plan's code, this can be useful for recursive plans.

9. Select an intention to continue its execution: typically, an agent has more than one indentation in its intention set, each one represents a different action focus. Intentions are competing for the agent's attention. However, in a reasoning cycle, at most one intention formulae from one intention can be executed. As such, before an agent can act, it is necessary to choose an intention from the agent's intention set. The selection intention function *SI* is in charge of such task, also the function is customizable. The default behavior of the function follows a round-robin schema.

10. Execute an intention step: once an intention was selected in the previous step, the intention starts or resumes its execution from the next formulae, which is in turn executed. The formulae may be an action that modifies the environment, an achievement goal, a test goal, a mental note, or an internal action.

Jason allows the programmer to modify the reasoning cycle through the definition of a custom agent architecture.

2.1.3. Agent communication in Jason

Communication in Jason is based on Speech Acts [79], using a subset of KQML, and extending its operation semantics [59]. The predefined action `.send` is used to exchange messages. When receiving a message, an agent puts it in a mailbox. Although messages are exchanged asynchronously, only one is selected from the mailbox, at the beginning of each reasoning cycle of the agent. Interactive messages, as asking something, have a colateral effect: the associated intention is temporarily suspended until the answer is received, or some defined lapse runs over.

Definition 2.1.2 *A message is a tuple $\langle id, ilf, cnt \rangle$ where:*

- *The agent identification id denotes the sender or receiver agent, depending if it is an input or output message.*
- *The performative ilf (illocutionary force) expresses the intention associated to the message. Performatives include: Tell, Untell, Achieve, Unachieve, TellHow, UntellHow, AskOne, AskAll, and AskHow.*
- *The content cnt of the message can be an atomic formula (at) or a set of them; a belief (b_i) or a set of them; and a plan (p_i) or a set of them, depending on the performative of the message.*

A more detailed discussion of agent communication in Jason, along with its related operational semantics, can be found in [59].

2.2. CArtAgO and endogenous environments

Traditionally, agents are conceived as entities situated in an environment, which they can perceive and modify through actions, also reacting to changes in it accordingly [77]. Not only that, but the agents' goal is to achieve an environment desired state. This conception of environment, as the locus of agent perception, action, reaction, interaction, and goals, stays true in current MAS development.

Two general perspectives are adopted when defined the concept of environment in MAS: exogenous, and endogenous [76]. The exogenous perspective is rooted in Artificial Intelligence, conceiving the environment as the external world, separated to the actual MAS, which can be only perceived and acted upon by agents. An example of this conception can be found in EIS [7]. In contrast, the endogenous perspective, grown in the context of Agent-Oriented Software Engineering (AOSE) [62], conceives the environment as a first class abstraction for MAS engineering [89], that not only can be perceived and acted upon, but it can also provide services and tools for the agents to aid them in their tasks, and as such, it is designed and implemented as part of the whole system. An example of this conception is found in CArtAgO [74, 75].

CArtAgO is a computational framework for the development of environments based on the paradigm of Agents and Artifacts [63]. The environment is conceived as a dynamic set of entities known as artifacts, representing resources and tools that can be used, perceived, and shared by the agents. From the design point of view, the artifacts are a first class abstraction, that provides computational services and resources to the agents. Agents, are yet still, the main abstraction for autonomous aspects of the system, in particular those related with goals and social interaction.

Thus, Jason agents in JaCa-DDM use artifacts to perceive and act on their environment. Figure 2.4 denotes graphically the core elements of an artifact. Since artifacts are better understood considering their interaction with agents, the Agents and Artifacts meta-model is shown in Figure 2.5, in order to make the following definitions clearer. The definitions presented here are based on [76]. First, the concept of artifact type is defined:

Definition 2.2.1 *A tuple $\langle OProps, Ops, OEvs, manual, ws \rangle$ denotes an artifact type, where:*

- *The set of observable properties $OProps = \{(var, val), (var', val'), \dots\}$ defines the artifact state variables (var), whose values (val) can be perceived by the agents as beliefs $var(val)$;*
- *The set of operations $Ops = \{o(t_1, \dots, t_n), o'(t_1, \dots, t_m) \dots\}$ defines the computational processes executed in the artifacts. Some operations can be triggered by agents as actions, providing what is called the usage interface of the artifact; others are triggered by other artifacts, providing an analogous link interface;*
- *The set of observable events $OEvs = \{(o, s), (o', s'), \dots\}$ represent observable internal events produced by operations (o), externalized as signals (s) that agents can perceive at the moment they occur, in a non-persistent way.*

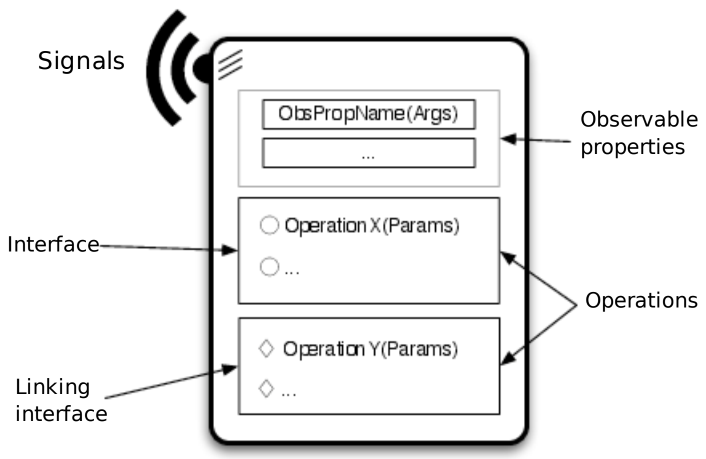


Figure 2.4.: Graphical representation of an artifact.

- *Optionally, every artifact can include a manual in a machine readable format, providing operation instructions for the agents that can consult it.*
- *The ws label denotes the workspace where the artifact is currently situated.*

Definition 2.2.2 *A workspace represents a place, the locus of activity involving a set of agents and artifacts. Agents can join and quit workspaces in the system. Three sets of actions are provided by a workspace for the agents:*

- *Actions to create, lookup and dispose artifact instances of the types provided by the workspace;*
- *Actions to use artifact instances: execute operations; perceive observable properties and events; and*
- *Actions to link and unlink artifact instances, so they can invoke operations between each other.*

In CArtAgO terms, the aspects that characterize a model for environment programming are the following [76]:

1. Action model: how to perform actions in the environment.
2. Perception model: how to retrieve information from the environment.
3. Environment computational model: how to represent the environment in computational terms.

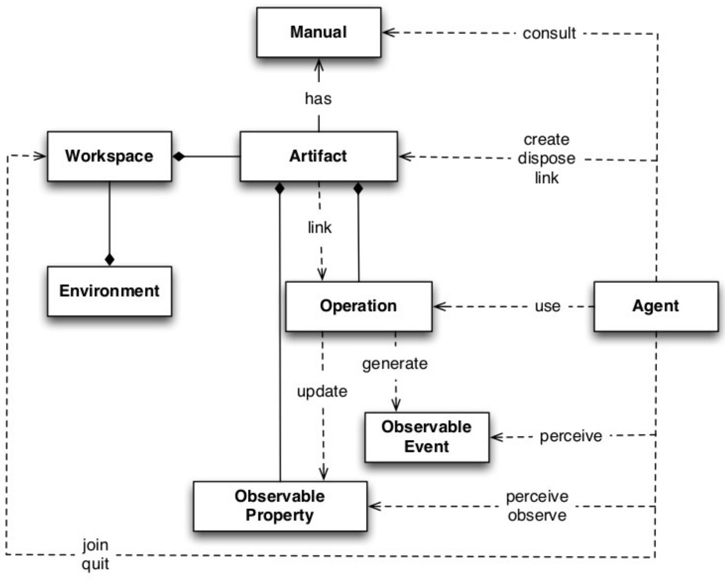


Figure 2.5.: The Agents and Artifacts meta-model, adapted from Ricci et al. [75]

4. Environment data model: how to share data between the agent and environment level to allow interoperability.

5. Environment distributed model: how to allow computational distributed environments.

Aspects 1-3 are directly supported by artifacts [64], which are dynamical sets of computational entities that compose the environment and encapsulate services and tools for the agents. Artifacts are organized and situated in workspaces, which essentially are logical places (local or remote) where agents center their attention and work. Aspect 5 is supported by workspaces, but also partially by artifacts, as artifact actions can be executed remotely. Aspect 4, on the other hand, depends on the underlying agent programming language used, and is not directly related to artifacts or workspaces.

CArtAgO considers distributed environments in its model. At the higher level, distribution is achieved through workspaces, which serve as logical places where agents may center their attention, and where artifacts are situated. Agents can create, join, and quit workspaces. If an agent is in a workspace, it can use the artifacts situated there.

At the low level, nodes enable distribution. A node is a CArtAgO process that can be remote, where workspaces can be spawned. When a JaCa MAS is deployed, it is contained in a default node, that node is also the default for agents, which consider it as it's local context, so workspaces created in that node are also local workspaces, but workspaces created in different nodes are considered remote workspaces. The distinction between remote and local workspace is not only conceptual, but also syntactical, requiring IP and port information at the agent level to manipulate remote workspaces.

Figure 2.6 depicts the current CArtAgO environment model from the workspaces and nodes perspective. From the figure, it is apparent the fact that there is no connection between nodes, and in consequence between workspaces in different nodes, needing to explicitly know the IP address and port of each node, which may be seen as a drawback. In section 9.2 a proposal that addresses this issue is presented.

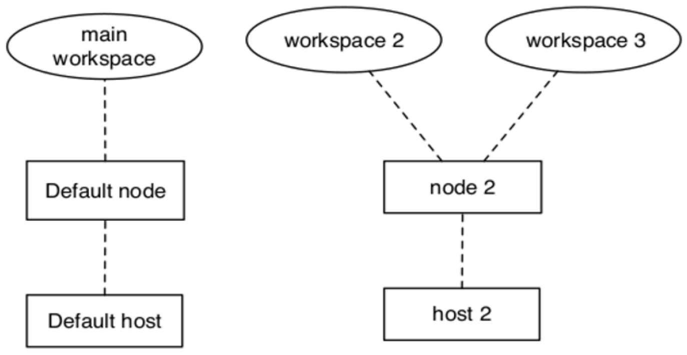


Figure 2.6.: Current CArtAgO environment model depicting multiple nodes and workspaces deployed.

3. JaCa-DDM

In what follows, the JaCa-DDM framework is described in detail. For this, some formal definitions of JaCa-DDM strategy and deployment are addressed. These definitions conform an abstract model of the system, and will guide its description. Then, the architecture induced by the deployment system is introduced, which includes the description of the primitive artifacts and agents of the system, and an account of the configuration and workflow of the deployment process.

3.1. JaCa-DDM Model

The JaCa-DDM model is built on the concepts of strategy and its deployment. While a strategy defines a DDM workflow in terms of the involved agents and artifacts, its deployment deals with configuration, distribution, and evaluation issues. Previous definitions of agent program, message, artifact type, workspace, and node are adopted.

Definition 3.1.1 *A tuple $\langle Ags, Arts, Params, ag_1\rangle$ denotes a JaCa-DDM strategy, where:*

- *$Ags = \{ag_1, \dots, ag_n\}$ is the set of user defined agent programs, involved in the strategy.*
- *$Arts = \{art_1, \dots, art_m\}$ is the set of user defined artifact types, used in the strategy.*
- *$Params = \{param_1 : type_1, \dots, param_k : type_k,\}$ is a set of paramenters and their associated data types, where $type_{1,\dots,k} \in \{int, bool, double, string\}$.*
- *$ag_1 \in Ags$ is a special agent program playing the role of contact person between the agents in Ags and the deployment system. This agent must have a plan to cope with a trigger event $+!start$, launching the strategy; and must eventually send a message $\langle ag_0, tell, finish(art)\rangle$ to announce the learning process is finished; where ag_0 is an agent in the deployment system, $art \in Arts$ stores the obtained model. Beyond that, ag_1 can be programmed to do any other tasks proper of a contact person.*

The workflow contained in a strategy, i.e., the way the agents learn together using their artifacts, is encapsulated in the agent programs. Chapter 4 uses UML sequence diagrams to describe a set of such workflows in detail. All the deployment system needs is an XML description of a strategy, this description follows definition 3.1.1.

Definition 3.1.2 *A tuple* $\langle Nodes, DS, Arts, Strat, Config, ag_0\rangle$ *is a JaCa-DDM deployment system, where:*

- $Nodes = \{node_0, node_1 \ldots, node_j\}$*, is a set of computational nodes, usually, but not necessarily, distributed in a network, where:* $node_0$ *is running Jason and CArtAgO, while* $node_{1,\ldots,j}$ *are running only CArtAgO. Each node defines a single CArtAgO workspace, where artifacts are to be created, but all agents run in* $node_0$*. Each node is denoted by a pair* $\langle nodeName, IPaddress : port\rangle$*.*

- $DS = \{ds_1, \ldots, ds_j\}$ *is a set of data sources associated to each node, excepting* $node_0$*. Data sources can be created dinamically at run time; or be statically defined in each node.*

- $Arts = \{art_1, \ldots, art_i\}$ *is a set of primitive artifact types, used to deploy the system.*

- *Strat is a learning strategy as stated in Definition 3.1.1.*

- $Config = \langle \delta, \pi\rangle$ *is a configuration for a strategy deployment. It has two components:*
  - $\delta = \{(ag, node, i), \ldots\}$*, is a set of allocations, i.e., an agent distribution specifying how many copies of a given agent program will be focusing (working) on a given node. Where* $ag \in Strat_{Ags}$ *is an agent program in the strategy, that will be cloned* $i \geq 1$ *times, and assigned to focus on* $node \in Nodes \backslash \{node_0\}$*. When the node is evident, cloning is denoted as* $ag\#i$*.*
  - $\pi = \{(p, v), \ldots\}$ *is a set of pairs: strategy parameter, initialization value; where for all* $p : t \in Strat_{Params}$*,* p *is a parameter of the strategy and* v *is its value of type* t*.*

The logic underlying the deployment is encapsulated in the agent program ag_0, deployment and configuration are described in detail in the next section.

3.2. JaCa-DDM Architecture

JaCa-DDM strategies can be deployed over a set of distributed computer nodes, as shown in Figure 3.1. All agents run and communicate at $node_0$; however, any node defines a workspace that the agents can logically join to create, perceive, and use artifacts on them. By the term logically, we mean that this is done exploiting Java remote method invocation (RMI), since agents always run at $node_0$. Apart from this node, all nodes have associated data sources. In this way, the artifacts running learning algorithms can be distributed efficiently, while communication among agents is kept as fast as possible. Although the JaCa-DDM architecture was designed with distribution in mind, it is also possible to execute it in a single computer to explore concurrency, running all workspaces $ws_{1,\ldots,j}$ also at $node_0$.

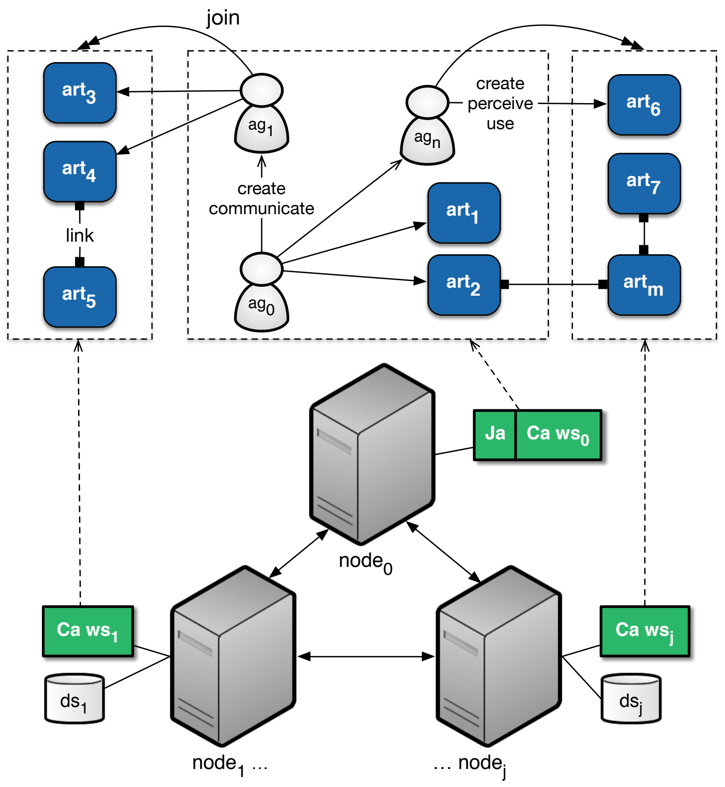


Figure 3.1.: An overview of the JaCa-DDM architecture, introducing agents and artifacts with their interactions. $node_0$ running Jason and a CArtAgO workspace, the rest of the nodes, running CArtAgO workspaces and having access to a data source.

In what follows, primitive artifacts and agents provided in JaCa-DDM are introduced. Artifacts come first, for the sake of clarity. Then, the configuration and deployment of strategies are described in detail.

3.2.1. Artifacts.

JaCa-DDM artifacts are designed as wrappers for the classes of Weka [91], the well known data mining environment; as well as MOA [11], the massive online analysis tool based on Weka. This decision was adopted for reusing code with comparison purposes: JaCa-DDM results can be fairly compared with the non-agent based results, obtained by Weka and MOA.

JaCa-DDM provides predefined artifact types, as stated in definition 3.1.2, which are listed in Table 3.1. User defined artifact types, as stated in definition 3.1.1, can also be added as required, e.g., Bayesian classifiers.

It is important to observe the following guidelines when deploying artifacts: The arti-

Artifact type	**Description**
Classifier	A classifier tool, there are different kinds, depending on the learning algorithm, e.g., J48, VFDT, and Bagging.
Directory	A localization service for agents, artifacts, and workspaces.
Evaluator	A Weka based evaluation tool for models.
GUI	A front end to configure and launch experiments.
FileManager	A tool to write ARFF files.
InstancesBase	A Weka based training set.
LogBook	An experiment logbook for reporting results.
Oracle	A tool to distribute centralized data sets.
TrafficMonitor	A sniffer to measure the load of a network.
Utils	A swiss army knife for the agents.

Table 3.1.: Predefined Weka based artifact types in JaCa-DDM.

facts created must have unique names, despite the fact that they are placed in different nodes. By convention, the name of an artifact includes the name of its creator to solve this. If an artifact is going to be referred by artifacts in other nodes, such artifact needs to be registered in the Directory.

3.2.2. Agents.

The agent program ag_0 is executed when launching JaCa-DDM, and it can be seen as an experimenter agent. It deals with the following tasks:

- Experiment configuration. Agent ag_0 optionally creates a *GUI* artifact to interactively configure the experiment, according to the adopted strategy. As stated in Definition 3.1.2, configuration includes the distribution (δ) of the agents over the nodes defined in the strategy; as well as the instantiation of its parameters (π).

- Dynamic data distribution. Agent ag_0 optionally creates an *Oracle* artifact to distribute a global data set among the nodes defined by the strategy. This is very useful to compare centralized approaches versus distributed ones. Alternatively, data can be already distributed among the nodes defined in the strategy, i.e., each node has a data source.

- Traffic monitor. Agent ag_0 creates a *TrafficMonitor* artifact in each node to measure the volume of network traffic generated by the experiment.

- Agents deployment. Agent ag_0 creates and assigns to nodes the strategy agents, as stated in the configuration. Initialization consists on communicating them useful information:
  - $node(NodeName)$. Each agent knows the logical name of its assigned node.

  - $ipNode0(IP)$. The IP address and port of $node_0$. This information can be helpful to use artifacts that are only in $node_0$, like the Directory.
  - $data(FilePath)$. The path to the data file of the node that is going to be used for training.
  - $param(ParamId, Value)$. A parameter assignment for the learning strategy.
  - Some core plans to register artifacts, exchange models among artifacts, and know how many copies of a given agent program are in the system.

- Evaluate models. Agent ag_0 creates an *Evaluator* artifact and a *LogBook* artifact to evaluate a model, and report results when receiving a message $\langle ag_1, tell, finish(Art)\rangle$.

- Cleanup. If the experiment evaluation involves iterated repetitions of the setting, agent ag_0 cleanups the system restarting agents and artifacts as required by the evaluation method.

Recall that $ag_1 \in Strat_{Ags}$ adopts the role of contact person between the agents in the strategy and ag_0. By Definition 3.1.1, there must be a single ag_1 that can join any workspace, as demanded by the strategy. As contact person, ag_1 has at least two responsibilities:

- Launching a learning process. When receiving a message $\langle ag_0, achieve, start\rangle$, ag_1 starts the strategy.

- Announcing a learning process is finished. When the learning process is finished, ag_1 sends a message $\langle ag_1, tell, finish(Art)\rangle$ to ag_0.

Other responsibilities for ag_1 could include initialization tasks, e.g., ask the other agents to load their training data, and even engaging in more complex coordination tasks. Figure 3.2 illustrates this. Observe that apart from the existence of ag_0 and ag_1, JaCa-DDM does not have any other assumption about the agents involved in a strategy. This enables the generalization of the deployment process to any kind of strategy.

3.2.3. Configuration

Following definition 3.1.2, every strategy deployment needs to be configured through an `XML` file that represents the experiment of interest. This `XML` file could be also generated by the GUI artifact. Figure 3.3 depicts the corresponding XML schema. Basically, it is possible to configure the following experimental aspects:

- Which learning strategy *Strat* is going to be adopted.

- The IP address and port of the different nodes $\{node_1 \ldots, node_j\}$ involved in the strategy deployment.

- The agent distribution δ; as well as the paramenter initialization π.

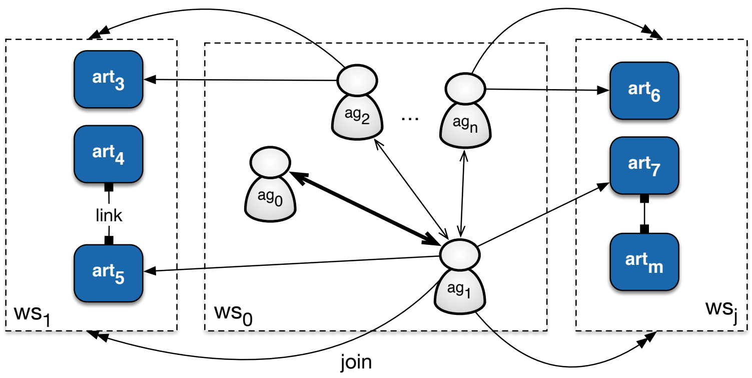


Figure 3.2.: Agent ag_1 adopts the role of contact person (bold arrow) between the ag_0 and the rest of agents in the strategy.

- The data source distribution $\{ds_1, \ldots, ds_j\}$.
- How many times an experiment is to be iterated.

Strategies need to be described in a standard way, following definition 3.1.1, in order to be loaded by the JaCa-DDM deployment system (to be able to choose which strategy to use). A `XML` file for each strategy needs to be defined for this purpose, which includes: The name of the adopted strategy *Strat*; The Jason agent programs in $Strat_{Ags}$, and; The parameters in $Strat_{Params}$. The `XML` files are validated through an `XML schema`, which it is showed in figure 3.4.

Data sources $\{ds_1, \ldots, ds_j\}$ can be created in two different ways:

- Static data distribution. Data sources are already in the available nodes and each node configuration indicates the path to its data source. Data sources can be pointed as:
  - A single file denoted by its name, e.g., `iris.arff`
  - Multiple files denoted by their root name followed by a number, e.g., `iris1.arff`, `iris2.arff`, etc.
- Dynamic data distribution. A single data source in $node_0$ is distributed among the other nodes, using one of two methods:
  - Hold-out. The data source is split in test and training partitions, according to a parameter defining the size of the test set (a percentage of the size of the data source). The training partition is splited again, according to the number of available nodes.
  - Cross validation. A fold parameter is used to indicate the ratio of the examples for training and testing. It also indicates how many iterations the testing will have. For example, a 5-fold cross validation splits the data source into five

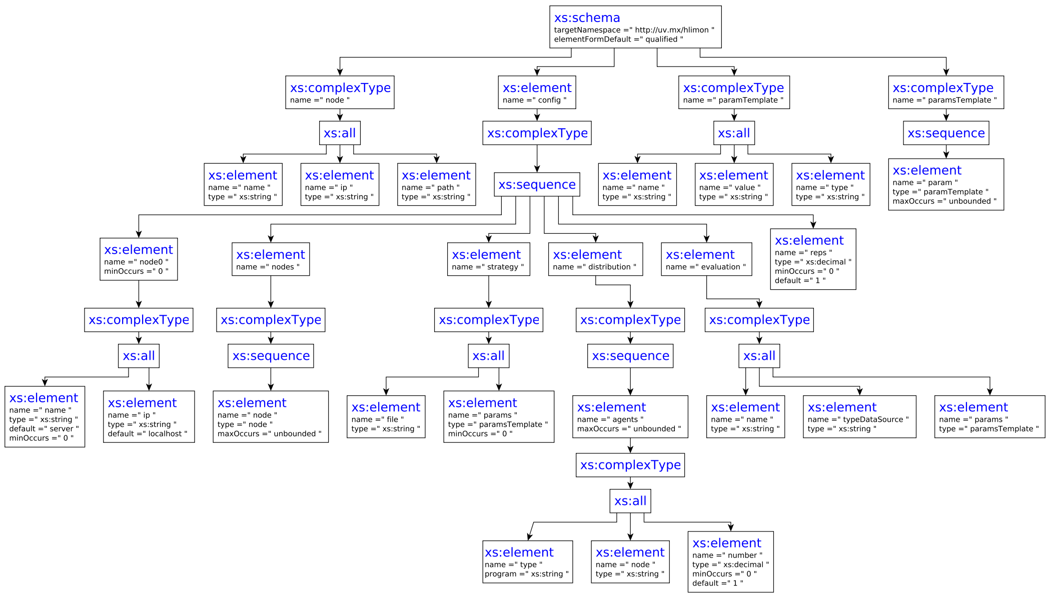


Figure 3.3.: XML schema of the strategy deployment.

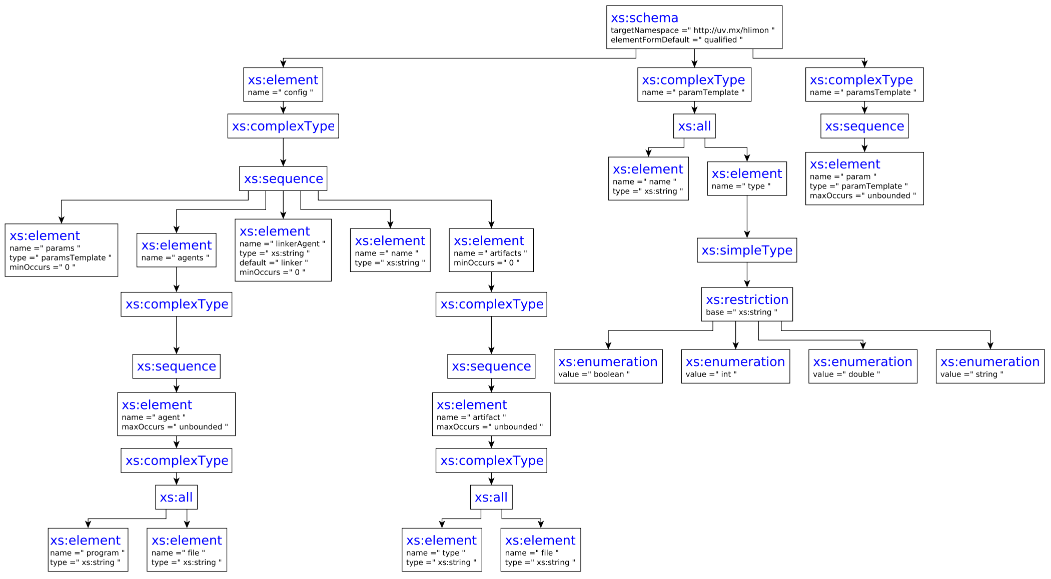


Figure 3.4.: XML schema of an strategy.

partitions, in each iteration one partition is reserved for testing and four are used for training. The process is repeated 5 times, selecting a different testing partition each time. Training partitions are distributed according to the number of available nodes.

The static and dynamic modes can be used together to do different experiments, with exactly the same data partitions. The idea is to run the first experiment under the dynamic mode; and then switching to the static mode, using the files produced in the first experiment for all the subsequent ones.

3.2.4. JaCa-DDM Workflow

The workflow of JaCa-DDM is as follows[1]:

1. If the user is interested in configuring an experiment, an artifact of type *GUI* is created by ag_0. The artifact produces an XML file coding such configuration. It is also possible to directly execute the experiment taking a configuration file as parameter. Then, ag_0 creates the following artifacts in $node_0$: a *LogBook*, an *Evaluator*, a *TrafficMonitor*, a *Directory*, a *Utils* artifact, and optionally an *Oracle* artifact if the data source distribution is dynamic. ag_0 also creates some artifacts in each available node ($node_1, \ldots, node_n$), a *FileManager* and a *TrafficMonitor*. Once the artifacts are created, ag_0 computes the data distribution specified in the configuration, so that each $node_1, \ldots, node_n$ gets a path to its data source. Afterwards, ag_0 creates the agents specified in the strategy; and communicate them core plans for joining remote workspaces, sending models, and registering artifacts in the directory. After that, ag_0 asks the created agents to join their respective workspaces, as specified in the configuration. Figure 3.5 illustrates this first step.

2. A message $\langle ag_i, tell, ready \rangle$ is sent to ag_0 every time an agent $ag_{i=1,\ldots,n}$ joins her/his workspace, to announce they are ready for starting the learning process.

3. ag_0 sends a message to ag_1, to launch the learning process. Once started, ag_1 may tell the other agents to initialize the learning process. This may include, for example, the creation and configuration of new artifacts. Meanwhile ag_0 keeps waiting for a termination message.

4. At the end of the learning process, a model has been computed and stored in a given artifact, depending on the adopted strategy. Once ag_1 is informed of the model location, it sends a message $\langle ag_0, tell, finish(ArtifactName) \rangle$ to ag_0. Figure 3.6 summarizes steps three and four.

[1]A tutorial explaining how to get started with JaCa-DDM is available at http://sourceforge.net/p/jacaddm/wiki

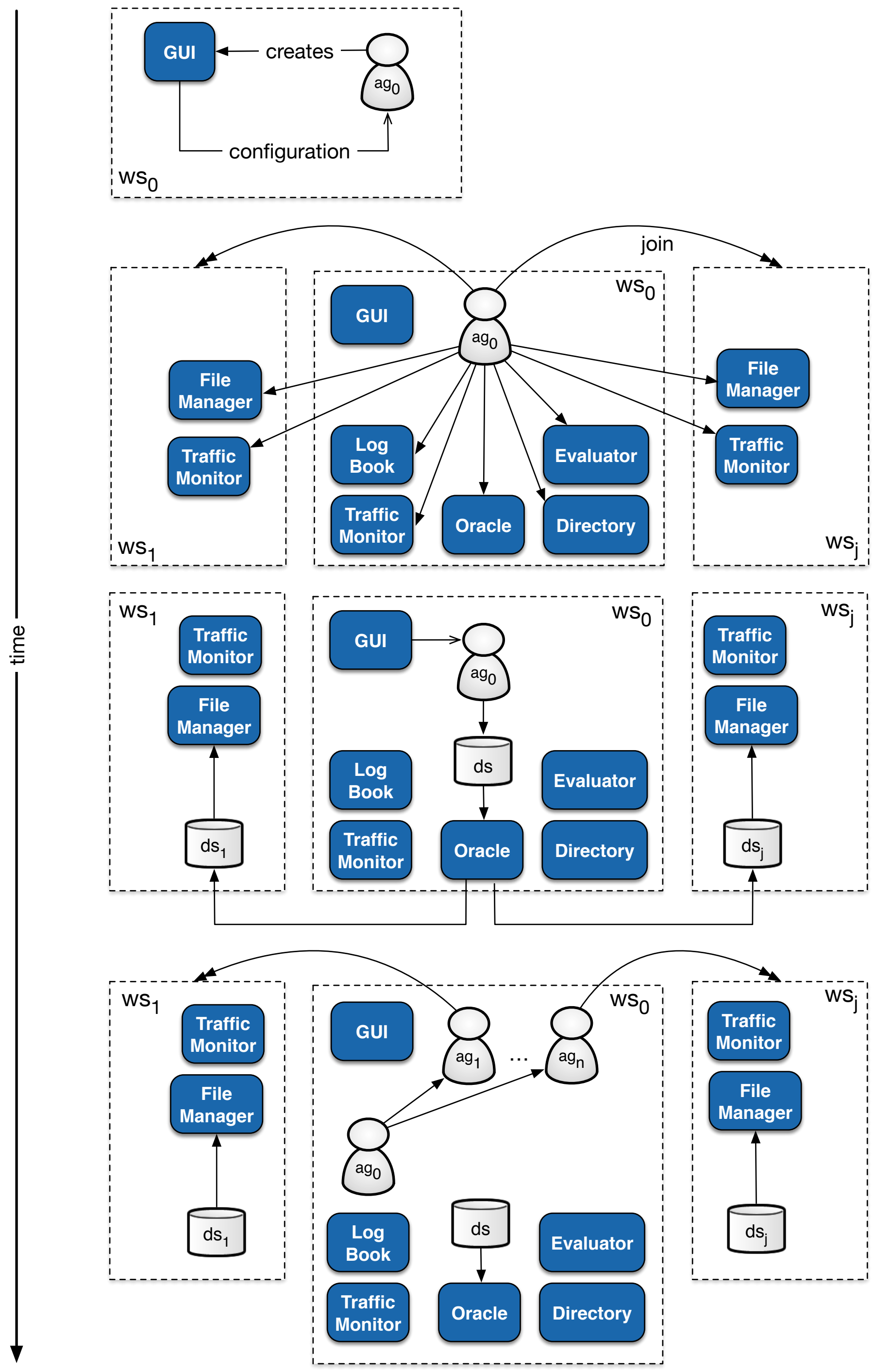


Figure 3.5.: Workflow step 1: Deployment of artifacts and agents.

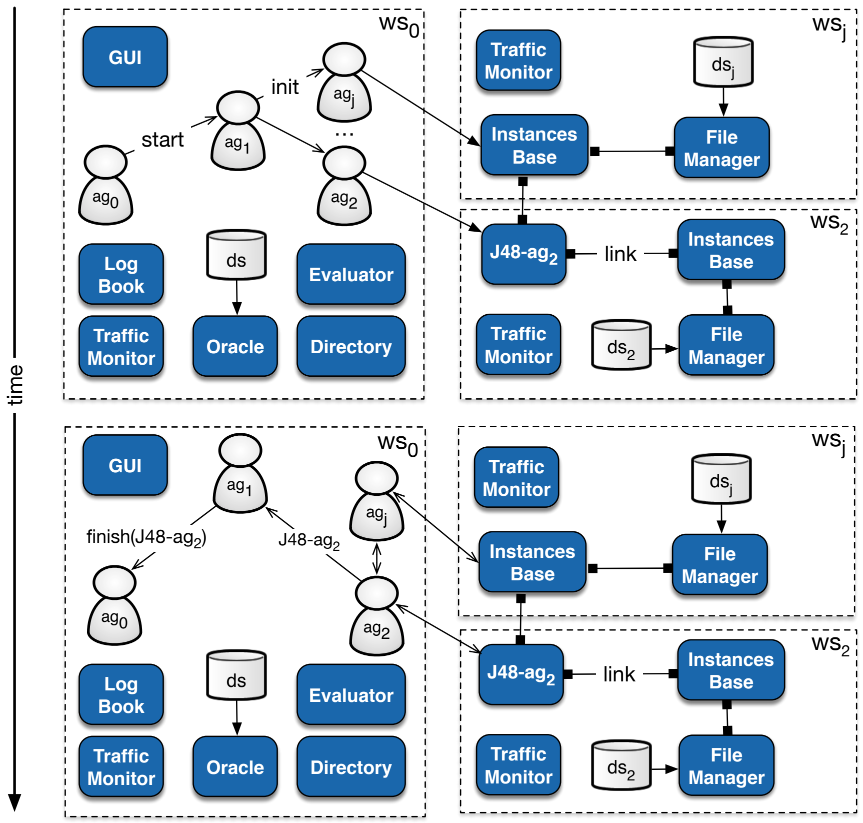


Figure 3.6.: Workflow steps 3 and 4: ag_0 starts a learning process, ag_1 initialize the agents in the strategy. More artifacts are created and configured in the workspaces. Once the process is over, ag_1 announce ag_0 where is the learned model.

5. When ag_0 receives the message $\langle ag_1, tell, finish(ArtifactName)\rangle$, the model is retrieved for evaluation. Results are then displayed in the GUI artifact, and at the same time, saved in a log file for future reference. Figure 3.7 summarizes this step.

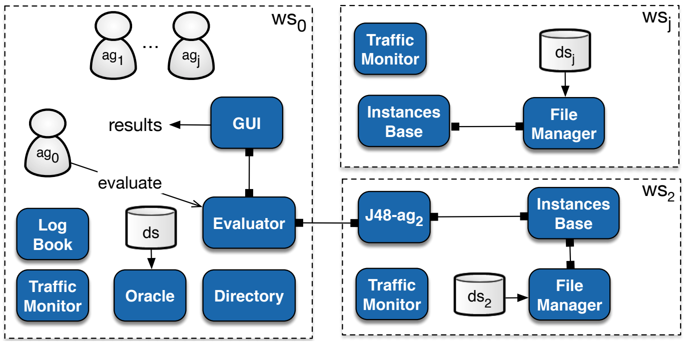


Figure 3.7.: Workflow step 5: The model in J48-ag_2 is evaluated, results are displayed in the *GUI* artifact.

6. ag_0 joins each workspace and deletes the agents and artifacts created. Then, a new iteration or experiment can be started.

In the next part, a series of learning strategies that exploit the concepts presented here are introduced. These strategies serve as a proof of concept of how JaCa-DDM can be used to implement traditional agent-based DDM approaches, as well as new ones, in this case based on a technique know as Windowing.

4. JaCa-DDM strategies

As mentioned in section 1.6.2, agent based DDM has been focused in two extensively discussed approaches: centralizing and meta-learning. The JaCa-DDM model is well suited to deal with both approaches, that can be studied and extended through it. This section addresses, as a proof of concept, the design and implementation of a set of strategies covering centralizing and meta-learning. A third group of strategies explores thoroughly an in between new approach, exploiting a technique known as Windowing [70, 35]. The technique consists on building an initial model in a given node, using local training data, and then enhancing it with counter examples, those not covered yet, from other nodes. Such process is not fully centralizing, since only the counter examples are to be collected; and that a kind of sampling, different from the ones performed in meta-learning approaches, is adopted. The set of Windowing-based strategies illustrates how JaCa-DDM can be adopted to design and enhance new DDM systems.

Focus in Decision Trees is due to the adoption of Windowing, as a way of keeping comparisons fair enough. Nevertheless, the proposed strategies are independent of the target model and the associated learning algorithm, i.e., other models and algorithms can be adopted. Extending JaCa-DDM in this sense is straightforward, e.g., use another classifier artifact. Learning algorithms include: J48, the Weka implementation of classical batch learning algorithm C4.5 [70]; VFDT [31], an incremental algorithm for on-line learning; Bagging [17] the ensemble technique for meta-learning, and Random Forest [18] a variant of Bagging. VFDT is able to update the learned model in the presence of new training examples, this should allow faster strategies based on model communication, since models use to be smaller than instances bases; but it is expected that the strategies based on instances communication have higher accuracy. Bagging is an ensemble technique where a set of learning models of the same type (J48 trees in this case, for experimental purposes), are created from separate training datasets representing the same problem. The different models are gathered at a single place, and used for classification, following a majority vote scheme. If the training data is scarce, Bagging can be used in conjunction with bootstrap sampling, where all the training data of size m is gathered, producing N training sets of size m by means of applying random sampling with replacement. Generally, the more training sets, and in consequence the more learning models, the better the results, since this reduces the variance of the method [91]. In the Weka definition of Bagging, it is possible to apply bootstrap sampling if desired, and also compute the internal learning models in parallel. In Random Forest the result is a set of decision trees, induced from bootstrap samples. Each tree induction is boosted by randomly sampling the attribute vector for each node split, based on a parameter K, determining the number of attributes taken into account at each node split. This makes possible to process datasets with a large numbers of attributes. Small K values are recommended, e.g., $log_2(|attributes|) + 1$. Map-Reduce extensions [36] apply a careful subsampling, in order to avoid sampling bias, but this is difficult in distributed scenarios where sites are naturally biased. Windowing may be useful as a sampling method, purposefully trying to skew the training example distribution, by considering only the counter examples found while learning [35]. Since Windowing samples while the model is learned, no extra sampling step is needed.

Then, four sets of strategies are addressed in this section:

- Centralized. Designed for benchmark. They are not true DDM strategies as they have as a basis the traditional centralized scenario, where all the training data is in one place. When possible, centralized approaches are the best, since there is no communication cost. Unfortunately, this approach could not scale well when dealing with massive amounts of data.

- Centralizing. Collect, if possible, all the training data distributed in the system in a single node, and then proceed as in the precedent case. They incur in data transmission costs.

- Meta-learning. Focus is on the way that meta-learning is supported by JaCa-DDM trough the implementation of a Bagging based strategy. Although results are compared with the other groups of strategies, the main interest here is to show the flexibility and extensibility of our approach, rather than establishing which set of strategies is the best. Bagging was chosen considering that it is well suited for distributed environments, since it does not necessarily need a special sampling of the data (in distributed scenarios it is normal to have biased data), whereas other traditional methods such as Boosting [34] do. It is worth noting that JaCa-DDM also supports special sampling of the data through the use of the Oracle artifact, and user defined artifacts.

- Windowing-based. Proposed alternative, exploring the use of the Windowing technique in DDM settings, as its potential reduction of training examples needed for learning, could be exploited in order to reduce data communication and also reduce induction time for large datasets.

The processes underlying these strategies are described as UML sequence diagrams. As stated in Section 9.2.1, the actual process is defined through agent programs. In each sequence diagram, artifacts are represented as rounded boxes, while agent programs appear as human figures labeled with the name of the agent, and suffixed with their number of clones, e.g., $contactPerson\#1$ produces the label *contactPerson_1*. Requests from agents to artifacts represent the execution of an artifact operation, parameters may be required. Requests from agents to agents represent speech acts messages. Their illocutionary force ilf (See Definition 2.1.2) are denoted as follows: tell (+), achieve (!), and ask (?), e.g., when an agent ag_1 requests ag_2 to $!start$, it is sending a message $\langle ag_2, achieve, start\rangle$; while ag_2 is receiving a message $\langle ag_1, achieve, start\rangle$. Requests from artifacts to artifacts represent linked operations, possibly including parameters.

When an artifact A makes a request to artifact B, it is executing a link operation of B. Most return messages from artifact operations are omitted, for the sake of readability. Representing nodes required a slight non standard extension in the UML diagrams, artifacts situated in the same node are surrounded by a dotted frame. Because of the available space, the agent ag_0 is represented implicitly outside the diagrams, as sending the *start* message and receiving the $finish$ message. Sub-diagrams are used as required.

4.1. Centralized strategies

The set of strategies presented in this section are used only as a benchmark, not being DDM approaches of interest. Centralized strategies include:

- Centralized. This is not properly a strategy, but the reproduction of a traditional centralized scenario. All data is in a single node, and the Data Mining process is run as usual. It is used to compare the results of the JaCa-DDM strategies, with the results that Weka and MOA would usually obtain. Observe there is no data communication in these scenarios.
- Centralized Bagging. Equivalent to applying Weka Bagging with bootstrap sampling in a centralized environment. It is essentially the same strategy as Centralized, but presented separately as it is used for meta-learning, requiring different parameters.

In what follows, each centralized strategy is described in detail, following JaCa-DDM model definition 3.1.1.

4.1.1. Centralized strategy

The centralized strategy reproduces the standard centralized Data Mining setting, where all data is in a single node and the Data Mining process is executed there. The components of this strategy are as follows:

- $Ags = \{contactPerson\}$, where:
  - $contacPerson$ is the required agent program playing this rol. Beyond its basic competences, it is in charge of inducing learned models.
- $Arts = \{Classifier, InstancesBase\}$, where:
  - $Classifier$. It is used to induce models and classify instances. Predefined classifiers include $J48$, and $VFDT$.
  - $InstancesBase$. Used to store and manipulate the learning examples.
- $Params = \{Classifier : String, Prunning : Bool\}$, where
  - $Classifier \in \{J48, VFDT\}$ specifies the adopted learning algorithm.
  - $Prunning$, if true, forces the learning algorithms to use post pruning.

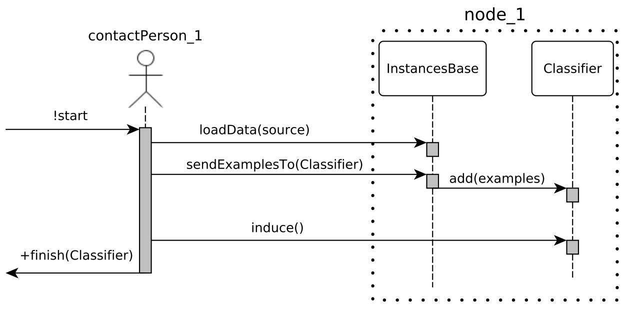


Figure 4.1.: Centralized strategy sequence diagram. The $!start$ message is sent by ag_0 in the deployment system, whereas the $+finish$ message is received by it.

A typical configuration for deploying this strategy is as follows:

- $\delta = \{(contactPerson, node_1, 1)\}$;
- $\pi = \{(Classifier, J48), (Pruning, true)\}$.

Figure 4.1 shows the workflow of this strategy, under such a configuration. When *contacPerson* receives a message $!start$ from ag_0, asking him to achieve the DDM process, it creates an *InstancesBase* and a J48 *Classifier* at $node_1$. Then, it loads the data source associated to $node_1$ in its *InstancesBase*. Once there, the training examples are sent to the linked artifact *Classifier*, where the learned model is induced. Finally, *contactPerson* sends ag_0 a message telling him that the DDM process is over, and the obtained model is in *Classifier*.

4.1.2. Centralized Bagging

The strategy applies the Bagging meta-learner with bootstrap sampling in a standard centralized Data Mining setting, as it would be done with Weka. The components of this strategy are as follows:

- $Ags = \{contactPerson\}$, where:
  - *contacPerson*, beyond its basic competences, is in charge of inducing learned models.
- $Arts = \{ClassifierBagging, InstancesBase\}$, where:
  - $ClassifierBagging$. It is used to induce meta-models and classify instances. It can use as internal models any classifier provided by Weka, such as bayesian networks, decision trees, support vector machines, etc.
  - $InstancesBase$. Used to store and manipulate the learning examples.

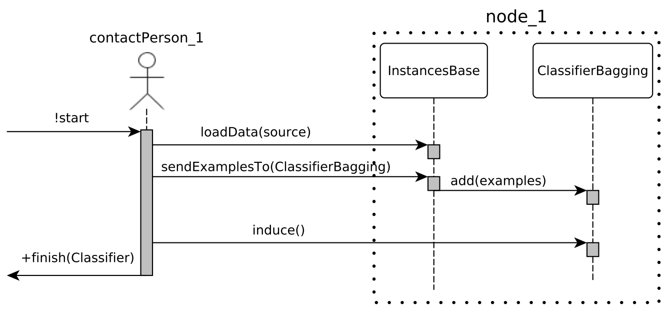


Figure 4.2.: Centralized bagging strategy sequence diagram.

- $Params = \{Threads : Int, NumberOfModels : Int, ClassifierType : String, ClassifierParams : String\}$, where
  - $Threads$ establishes the number of threads that the induction process will have. If not provided (value 0) it uses the maximum number of parallel threads that the computer node can support.
  - $NumberOfModels$ specifies the number of models in the ensemble, this is equivalent to establishing the number of bootstrap data partitions from the training set. If not provided (value 0) it takes the same value as the $Threads$ parameter.
  - $ClassifierType$ is the Weka classifier that will be used as the internal models of the ensemble, by default J48.
  - $ClassifierParams$ specifies the optional parameters of the internal models following Weka notation.

A typical configuration for deploying this strategy is as follows (by default, in Weka J48 prunning is active):

- $\delta = \{(contactPerson, node_1, 1)\}$;
- $\pi = \{(Threads, 32), (NumberOfModels, 32), (ClassifierType : "weka.classifiers.trees.J48"), (ClassifierParams : "")\}$.

Figure 4.2 shows the workflow of this strategy, under such configuration.

At the beginning of the process, *contacPerson*$_1$ creates an *InstancesBase* and a *ClassifierBagging* at *node*$_1$. Then, it loads its data source associated in its *InstancesBase*. Next, the training examples are sent to *ClassifierBagging*, where the meta-classifier is induced.

4.2. Centralizing strategies

Centralizing strategies consider distributed settings, they use all the available data in the learning process. This set of Strategies include:

- Centralizing. The same as centralized, but data sources are actually distributed in different nodes. It takes into account the time and communication required to centralize the data.
- Round. This strategy also uses all data in the system, but instead of communicating examples, it moves the learned model through the nodes. In this way, not only communication cost is improved, but privacy too. Incremental learning algorithms, e.g., VFDT, are required to implement this strategy.

4.2.1. Centralizing strategy

The centralizing strategy consists in sending all the available training examples to a single node, and proceed then as in the centralized case. The components of this strategy are as follows:

- $Ags = \{contactPerson, sender\}$, where:
  - *contactPerson* builds the learned model, beyond its basic competences.
  - *sender* is in charge of collecting training instances for *contactPerson*.
- $Arts = \{Classifier, InstacesBase\}$, as before.
- $Params = \{Classifier : String, Prunning : Bool\}$, as before.

A typical configuration for deploying this strategy is as follows:

- $\delta = \{(contactPerson, node_1, 1), (sender, node_1, 1), \ldots, (sender, node_j, 1)\}$;
- $\pi = \{(Classifier, J48), (Pruning, true)\}$.

Figure 4.3 shows the workflow of this strategy. When *contactPerson* receives an $!start$ message from ag_0, it creates a *Classifier* artifact. There are *sender#j* clones, one for each $node_{i=1,\ldots,j}$. Each *sender* creates an $InstancesBase$ in its node. Observe that the diagram focus on the interactions between a sender and a node, but this generalizes to j agents, focusing on j nodes. The *contactPerson* asks all the *sender* agents to load their data sources. Once the data sources are loaded, each *sender* adds its training examples to the *Classifier* in $node_1$. Once all training exampres are centralized, *contactPerson* builds a new model, and notifies ag_0.

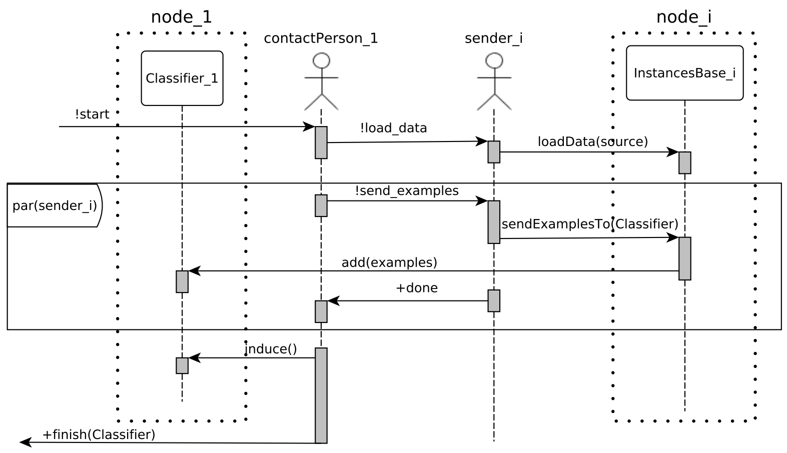


Figure 4.3.: Centralizing strategy sequence diagram. Observe that par(sender_i) means these parts of the process are parallel for each agent program $sender_i$, where $i = 1, \ldots, j$ denotes any *sender* (an so, any associated *node*).

4.2.2. Round strategy

The round strategy consists in learning an initial model with all the examples from one node, and then updating it to each node in a round fashion. When it is the turn of a given node, the model is updated with all the locally available data, and then it is moved to the next node. The process continues until a round is completed. Because of its nature, the use of incremental learning algorithms, e.g., VFDT, is mandatory. The components of the strategy are as follows:

- $Ags = \{contactPerson, worker\}$, where:
  - *contactPerson* controls the rounds, beyond its basic competences.
  - *worker* is in charge of updating a learned model it receives, using its training examples.
- $Arts = \{Classifier, InstancesBase\}$, as before;
- $Params = \emptyset$, since only the VFDT algorithm is available. If other incremental algorithms were available, this is the place to choose one of them.

A typical configuration for deploying this strategy is as follows:

- $\delta = \{(contactPerson, node_1, 1), (worker, node_1, 1), \ldots, (worker, node_j, 1)\}$;
- $\pi = \emptyset$.

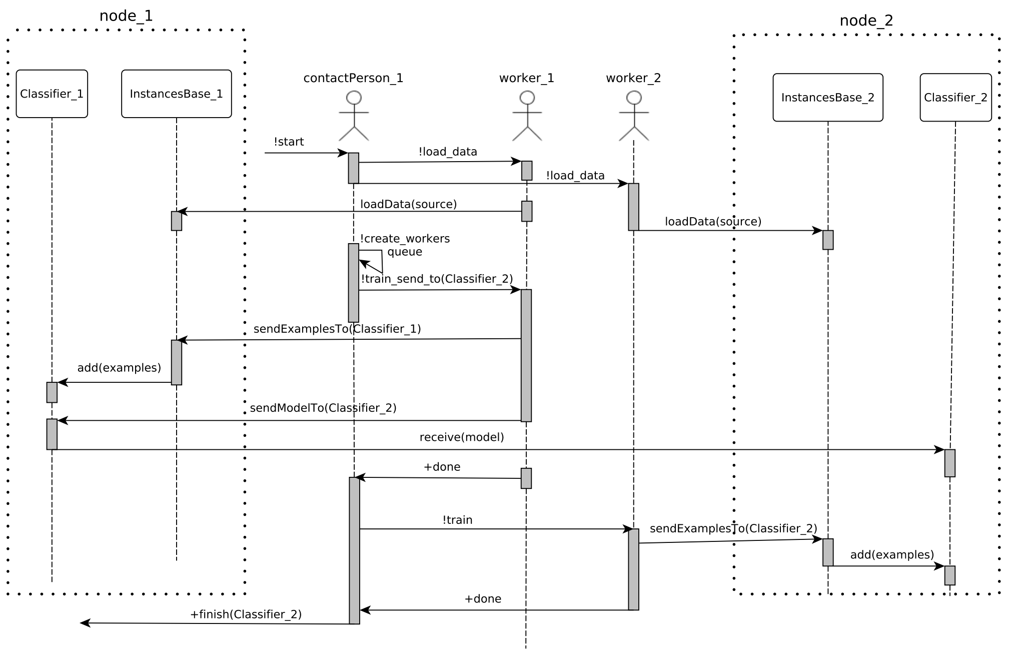


Figure 4.4.: Round strategy sequence diagram. Observe that the operation $add(examples)$ does an implicit induction, since the classifier is incremental.

Figure 4.4 shows the workflow for this strategy. Because of the available space, only the first two workers are shown, but the workflow generalize to j workers focusing on j nodes. Once *contactPerson* starts the DDM process, it asks the worker agents to create an *InstancesBase* and a *Classifier* in their nodes. Then, *contactPerson* asks the workers to load their data sources and creates a queue of available workers. After that, *contactPerson* asks $worker_1$ to send its examples to its classifier, inducing a model; and to pass it to $worker_2$. Recall that the learning algorithm is incremental, it updates the model when receiving examples. Once this is done, *contactPerson* is notified by $worker_1$. Then, $worker_2$ is asked to do the same and share the model with $worker_3$ and so on. As usual, *contactPerson* notifies ag_0 when the rounds of all nodes is over, and the obtained model is in $Classifier_j$ ($j = 2$ in the diagram).

4.3. Meta-learning strategies

This section illustrates how meta-learning based strategies for DDM can be defined and implemented in JaCa-DDM terms, using as a case study the Bagging meta-classifier. Although different from centralizing approaches, meta-learning methods can also be represented through a set of coordinated and communicative agents, that use Weka-based artifacts in a distributed network.

The strategy presented in this section, Distributed Bagging, is intended for DDM settings, and it constitutes an ensemble of ensembles, that is, in each distributed node a Bagging with bootstrap sampling learner is produced, and then each meta-learner is assembled as a single one at a centralized node.

Note that it is also possible to implement the distributed Bagging strategy using individual classifiers in each node instead of the Bagging meta-classifier, but as the JaCa-DDM model entails the reuse of existing code through artifacts, and the Weka implementation of Bagging has the advantage of being parallelizable, it was choosen to exploit this characteristic in order to have more models in each node with no special extra cost in time, except for transmission time. In what follows the distributed Bagging strategy is addressed.

4.3.1. Distributed Bagging

This strategy consists on building Bagging meta-learners with bootstrap sampling in each distributed node, and then assembly them as a single one in a centralized node (i.e., all the internal models are aggregated to the same bag). The components of the strategy are as follows:

- $Ags = \{contactPerson, bagger\}$, where:
  - $contacPerson$ beyond its basic competences, asks the bagger agents to induce and send its meta model to the Bagging assembler that controls.
  - $bagger$ uses its training data to build a Bagging meta-classifier, and then sends it to a centralized Bagging assembler.
- $Arts = \{ClassifierBaggingEnsemble, ClassifierBagging, InstancesBase\}$, where:
  - $ClassifierBaggingEnsemble$. Assembles various Bagging models, in order to treat them as a single Bagging model.
  - $ClassifierBagging$. It is used to induce meta-models and classify instances. It can use as internal models any classifier provided by Weka, such as bayesian networks, decision trees, support vector machines, etc.
  - $InstancesBase$. Used to store and manipulate the learning examples.
- $Params = \{ThreadsPerNode : Int, ModelsPerNode : Int, ClassifierType : String, ClassifierParams : String\}$, where
  - $ThreadsPerNode$ establishes the number of threads that the induction process in any node will have. If not provided (value 0) it uses the maximum number of parallel threads that the computer node can support.
  - $ModelsPerNode$ specifies the number of models in the ensemble for any node, this is equivalent to establishing the number of bootstrap data partitions from the training set. If not provided (value 0) it takes the same value as the $ThreadsPerNode$ parameter.

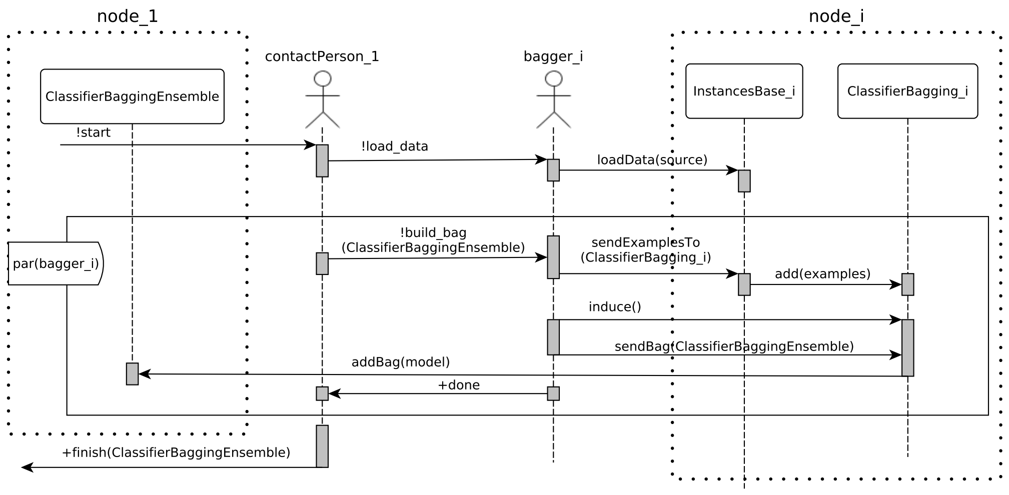


Figure 4.5.: Distributed bagging strategy sequence diagram. bagger_i represents any bagger, i.e $i = 1, ..., j$ (the same goes for node_i).

- $ClassifierType$ is the Weka classifier that will be used as the internal models for the $ClassifierBagging$ artifacts, by default J48.
- $ClassifierParams$ specifies the optional parameters of the internal models for the $ClassifierBagging$ artifacts, following Weka notation.

A typical configuration for deploying this strategy is as follows (by default, in Weka J48 prunning is active):

- $\delta = \{(contactPerson, node_1, 1), (bagger, node_1, 1), ..., (bagger, node_j, 1)\}$;
- $\pi = \{(ThreadsPerNode, 32), (ModelsPerNode, 32), (ClassifierType : "weka.classifiers.trees.J48"), (ClassifierParams : "")\}$.

Figure 4.5 shows the workflow of this strategy, under the mentioned configuration. At the beginning of the process, *contacPerson*$_1$ creates a *ClassifierBaggingEnsemble* at *node*$_1$ and asks the bagger agents to load their data into their *InstancesBase* artifacts and to create *ClassifierBagging* artifacts. When all the bagger agents announce to the *contacPerson*$_1$ that they are ready, *contacPerson*$_1$ asks them to create their Bagging model and send it to its *ClassifierBaggingEnsemble*. The baggers send their training data to their *ClassifierBagging* and then induce a Bagging model. When this process is done, the resulting model is sent to the central *ClassifierBaggingEnsemble*.

4.3.2. Distributed Random Forest

This strategy consists on building Random Forest meta-learners with bootstrap sampling in each distributed node, and then assembly them in a single forest in a centralized node (i.e., all the internal forest are aggregated to the same forest). The components of the strategy are as follows:

- $Ags = \{contactPerson, bagger\}$, where:
  - $contacPerson$ beyond its basic competences, asks the bagger agents to induce and send its meta model to the Bagging assembler that controls.
  - $worker$ uses its training data to build a forest meta-classifier, and then sends it to a centralized forest assembler.
- $Arts = \{ClassifierRandomForestEnsemble, ClassifierRandomForest, InstancesBase\}$, where:
  - $ClassifierRandomForestEnsemble$. Assembles various forest models, in order to treat them as a single forest.
  - $ClassifierRandomForest$. It is used to induce meta-models and classify instances. It can use as internal models any classifier provided by Weka, such as bayesian networks, decision trees, support vector machines, etc.
  - $InstancesBase$. Used to store and manipulate the learning examples.
- $Params = \{ThreadsPerNode : Int, ModelsPerNode : Int, ClassifierType : String, ClassifierParams : String\}$, where
  - $ThreadsPerNode$ establishes the number of threads that the induction process in any node will have. If not provided (value 0) it uses the maximum number of parallel threads that the computer node can support.
  - $ModelsPerNode$ specifies the number of models in the ensemble for any node, this is equivalent to establishing the number of bootstrap data partitions from the training set. If not provided (value 0) it takes the same value as the $ThreadsPerNode$ parameter.
  - $ClassifierType$ is the Weka classifier that will be used as the internal models for the $ClassifierBagging$ artifacts, by default J48.
  - $ClassifierParams$ specifies the optional parameters of the internal models for the $ClassifierRandomforest$ artifacts, following Weka notation.

A typical configuration for deploying this strategy is as follows (by default, in Weka J48 prunning is active):

- $\delta = \{(contactPerson, node_1, 1), (worker, node_1, 1), ..., (worker, node_j, 1)\}$;
- $\pi = \{(ThreadsPerNode, 32), (ModelsPerNode, 32), (ClassifierType : "weka.classifiers.trees.J48"), (ClassifierParams : "")\}$.

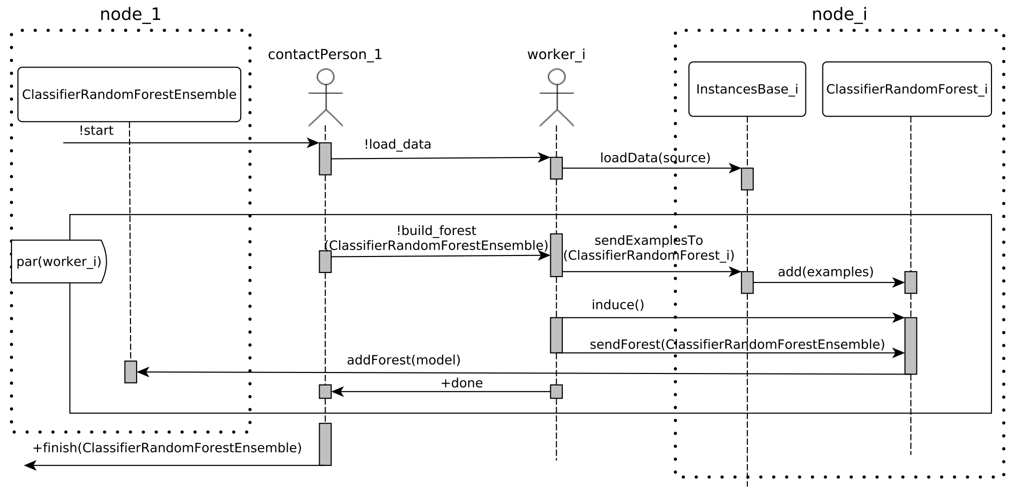


Figure 4.6.: Distributed Random Forest strategy sequence diagram. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i).

Figure 4.6 shows the workflow of this strategy, under the mentioned configuration. At the beginning of the process, $contacPerson_1$ creates a *ClassifierRandomForestEnsemble* at $node_1$ and asks the worker agents to load their data into their *InstancesBase* artifacts and to create *ClassifierRandomForest* artifacts. When all the bagger agents announce to the $contacPerson_1$ that they are ready, $contacPerson_1$ asks them to create their forest model and send it to its *ClassifierRandomForestEnsemble.* The workers send their training data to their *ClassifierRandomForest* and then induce a forest model. When this process is done, the resulting model is sent to the central *ClassifierRandomForestEnsemble.*

4.4. Windowing-based strategies

The following strategies exploit a technique known as Windowing [70], originally designed to cope with memory limitations when executing C.45, the classical system for inducing Decision Trees. Windowing (Algorithm 4.4.2) consists in learning a model from a small random sample extracted from the whole training set, the window. Then, the counter examples found in the remaining training set, if any, are added to the window. A counter example is a missclassified example, i.e., its class is not predicted correctly by the model. A new model is computed with this extended window. The process is repeated until a stop criteria is met, e.g., there are no more available counter examples.

Algorithm 4.4.1 *The Windowing algorithm.*

```
function WINDOWING(Examples)
    $Window \leftarrow sample(Examples)$
    $Examples \leftarrow Examples - Window$
    repeat
        $stopCond \leftarrow true$
        $model \leftarrow induce(Window)$
        for $example \in Examples$ do
            if $classify(model, example) \neq class(example)$ then
                $Window \leftarrow Window \cup \{example\}$
                $Examples \leftarrow Examples - \{example\}$
                $stopCond \leftarrow false$
    until $stopCond$
    return $model$
```

Windowing did not get very popular [35], due to the fast memory size improvements and, more importantly, because the learned models computed in this way, do not improve significantly; while, searching the counter examples increases the computational costs. Nevertheless, if a centralizing strategy is not possible, nor convenient, windowing provides a tool for approaching models as good as those obtained in standard centralized scenarios.

All the Windowing-based JaCa-DDM strategies presented in this section do the rounds of the available nodes, as suggested by the main loop of Algorithm 4.4.2. Variations include: the number of examples used to compute the initial model; the way counter examples are collected, and their number in each round; the stop conditions, e.g., a threshold in accuracy improvement. The general sequence diagram for all Windowing-based strategies is shown in Figure 4.7. When receiving a $!start$ message, $contactPerson_1$ asks all $worker$ agents to load their data sources. For this, each $worker_i$ has created previously an $InstancesBase_i$ artifact. Then, $contacPerson_1$ asks $worker_1$ to create a learned model using a percentage of its available training examples. Once this is done, $contactPerson_1$ creates a queue of $worker$ agents to iterate on them the process of counter example gathering. JaCa-DDM Windowing-based strategies vary in their counter examples gathering processes. For the sake of clarity, the workflow to determine the stop criteria is simplified in Figure 4.7 : $contactPerson_1$ asks $roundController_1$ if another round is required. Depending on the answer, $contactPerson_1$ can iterate the process one more time; or tell ag_0 that the process is over and the learned hypothesis is in $Classifier_1$. There are different possible ways of determining the stop criteria. Fewer rounds produce faster DDM processes, but the accuracy of the obtained models tends to decrease. Iterating until all counter examples have been processed, as in Algorithm 4.4.2, can be impractical, if few counter examples are considered in each round; or unnecessary, if updating converges fast to an acceptable model.

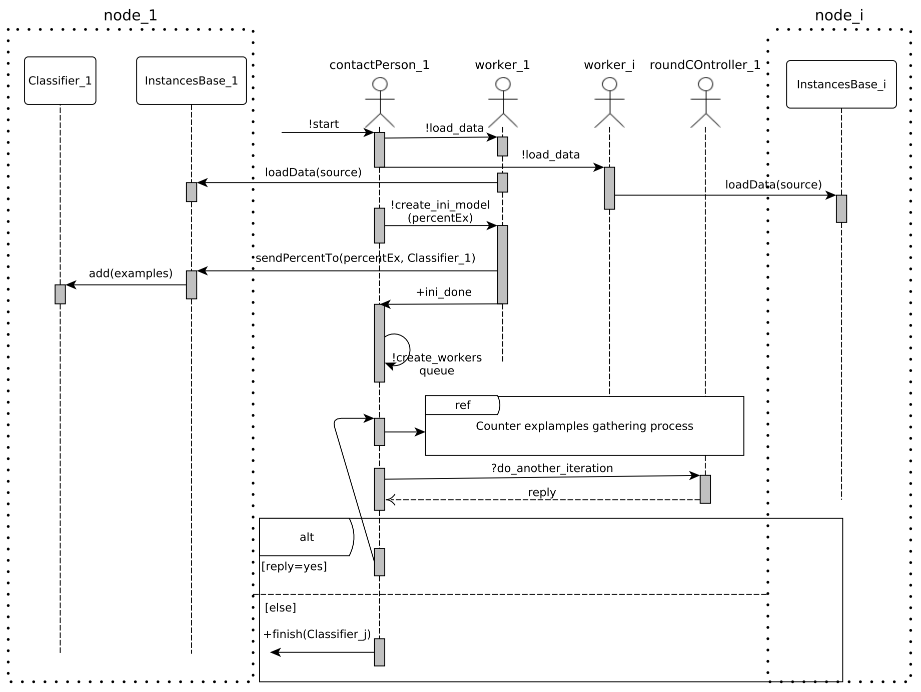


Figure 4.7.: Windowing-based strategies general sequence diagram. worker_i represents any worker except worker_1 (the same goes for associated node_i), i.e. $i = 2, ..., j$.

Figure 4.8 details the workflow for an auto-adjust method to determine the stop criteria. The method consists on reserving a validation set to evaluate the model at the end of each round. If the accuracy of the current model compared with the accuracy of the previous round surpasses a given threshold, then the process continues, otherwise the process stops. The threshold value and the percentage of the training data used for validation are configurable parameters. The workflow for the auto-adjust method starts when $contactPerson_1$ tells $roundController_1$ the name of the node where the last computed model for the iteration ($node_j$) will be located, where $roundController_1$ creates a $Evaluator_j$ and reserves a validation set in it. After the counter examples gathering process, $roundController_1$ evaluates the accuracy of the obtained model to tell $contactPerson_1$ if another iteration should be done or not.

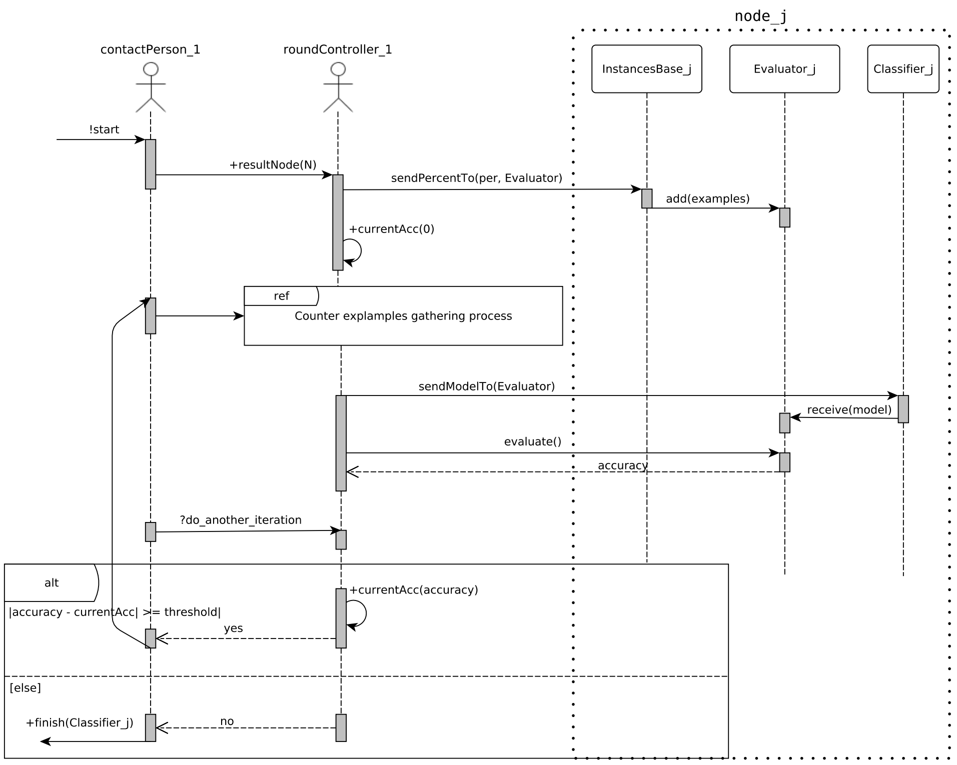


Figure 4.8.: Auto-adjust method sequence diagram for determining the stop criteria. This diagram complements Figure 4.7.

The set of Windowing-based strategies include:

- Counter. All counter examples are sent to the node where the initial model has been computed. The process iterates n predefined rounds, or an auto-adjust procedure can stop the iterations.
- Round Counter. The learned model is communicated instead of the counter examples. It requires an incremental learning algorithm as VFDT. The process iterates n predefined rounds, or an auto-adjust procedure can stop the iterations.
- Parallel Round Counter. It is similar to the previous strategy, but counter examples are searched in parallel, once the initial model is shared in all the nodes. Updating the hypothesis proceeds as in Round Counter.

In what follows, each Windowing-based strategy is described in detail, following Definition 3.1.1.

4.4.1. Counter strategy

The counter strategy consists in gathering all the counter examples found in a node, and sending them to the classifier artifact used to build the learned model for updating it. The processes continues for a given number or rounds or until the round auto-adjust procedure determines it is over. The componentes of this strategy are as follows:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the rounds and induces the learned model, beyond its basic competences.
  - *worker* gathers counter examples.
  - *roundController* determines if the auto-adjust stop condition has been met.
- $Arts = \{Classifier, InstancesBase, Evaluator\}$, where:
  - $Classifier$. Defined as before.
  - $InstancesBase$. Defined as before.
  - $Evaluator$. It is used to compute the accuracy of a model given a validation set, for the auto-adjust stop procedure.
- $Params = \{Classifier : String, Prunning : Bool, InitPercentage : Double, ValidationPercentageForRounds : Double, ChangeStep : Double, MaxRounds : Int\}$, where:
  - $Classifier$. Defined as before.
  - *Prunning*. Defined as before.
  - $InitPercentage$ defines the size of the initial training set, i.e., the initial window size.
  - $ValidationPercentageForRounds$ defines the size of the validation set for the auto-adjust stop procedure.
  - $ChangeStep$ defines a threshold of minimum change between two consecutive rounds. Used by the auto-adjusted stop procedure.
  - $MaxRounds$ defines the maximum number of rounds.

A typical configuration for deploying this strategy is:

- $\delta = \{(contactPerson, node_1, 1), (roundController, node_1, 1), (worker, node_1, 1), \ldots, (worker, node_j, 1)\}$;
- $\pi = \{(Classifier, J48), (Pruning, true), (InitPercentage, 0.25), ValidationPercentageForRounds, 0.20), (ChangeStep, 0.35), (MaxRounds, 15)\}$.

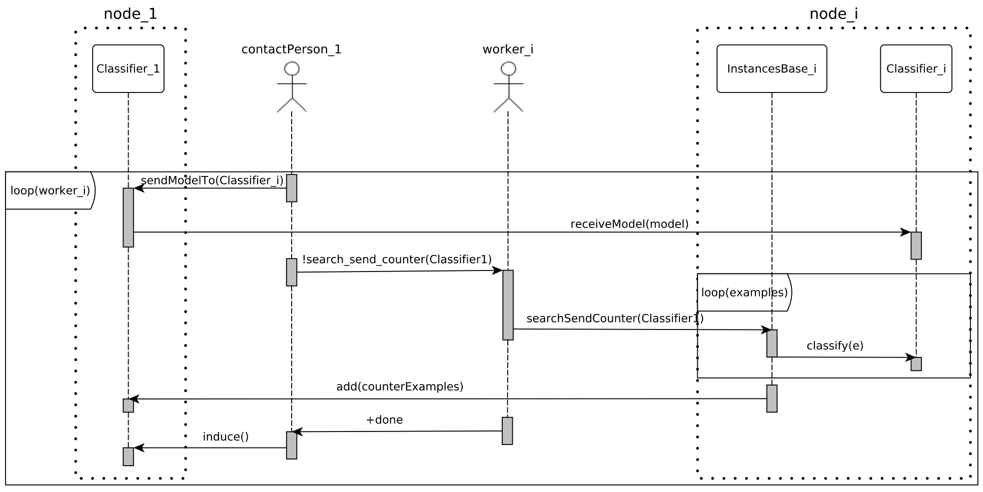


Figure 4.9.: Counter strategy sequence diagram for counter examples gathering workflow. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i). Recall this process is part of the general windowing based strategy shown in Figure 4.7.

Figure 4.14 shows the workflow of the counter strategy. The process iterates over all workers.

At the beginning, $contacPerson_1$ sends the current model to the $Classifier_i$ artifact of $worker_{i=1}$. Then it asks the worker to search for counter examples in their $InstancesBase_i$ and send them to $Classifier_1$, where a new model is induced. Then the process is repeated with $worker_{i=2}$, and so on.

4.4.2. Round Counter

This strategy combines the round (Section 4.2.2) and the counter (Section 4.4.1) strategies, so that it requires an incremental learning algorithm, e.g. VFDT. The idea is to share a model, created as in the counter strategy, doing the rounds of the available nodes. In each node, the model is updated with local counter examples. The process is repeated a given number of times, or the auto-adjust method is used instead. The componentes of this strategy are as follows:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the rounds, beyond its basic competences.
  - *worker* searches for counter examples and integrates them into the current model, then passes the model to the next worker.
  - *roundController* determines if the auto-adjust stop condition has been reached.

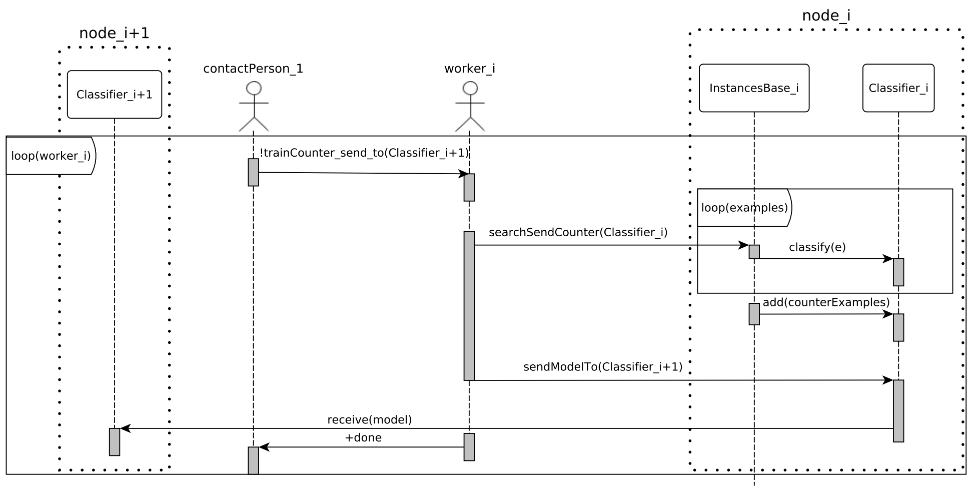


Figure 4.10.: The round counter strategy sequence diagram. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i). Note that when $i = j$, $i + 1 = 1$. Recall this process is part of the general windowing based strategy shown in Figure 4.7.

- $Arts = \{Classifier, InstancesBase, Evaluator\}$, defined as before.
- $Params = \{InitPercentage : Double, ValidationPercentageForRounds : Double, ChangeStep : Double, MaxRounds : Int\}$, defined as before.

A typical configuration for deploying this strategy is as follows:

- $\delta = \{(contactPerson, node_1, 1), (roundController, node_1, 1), (worker, node_1, 1), \ldots, (worker, node_j, 1)\}$;
- $\pi = \{(InitPercentage, 0.25), ValidationPercentageForRounds, 0.20), (ChangeStep, 0.35), (MaxRounds, 15)\}$.

Figure 4.10 shows the sequence diagram for the round counter strategy. At the beginning,$contactPerson_1$ asks each $worker_i$, one at a time, to proceed as follows: To search for counter examples in $InstancesBase_i$ and update the model in $Classifier_i$ with them; then, to send the updated model to the $Clasifier_{i+1}$ of the next worker.

4.4.3. Parallel Round Counter

While the round counter strategy iterates over the nodes in a strict sequence, this strategy parallelize the searching for counter examples, in an attempt to accelerate the overall process. An initial model is computed and copied to all the nodes. In parallel, each node gathers counter examples; After that, a round counter of the nodes starts to update the initial model, exploring only the counter examples gathered in the previous parallel phase. The process is repeated a given number of times, or the auto-adjust method is used to determine the stop criteria. The componentes of this strategy are as follows:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the parallel and the round phases, beyond its basic competences.
  - *worker* searches counter examples of the initial model during the parallel phase; then, in the round phase, searches for further counter examples to update the current model, passing it to the next worker.
  - *roundController* determines if the auto-adjust stop condition has been reached.
- $Arts = \{Classifier, InstancesBase, Evaluator\}$, defined as before.
- $Params = \{InitPercentage : Double, ValidationPercentageForRounds : Double, ChangeStep : Double, MaxRounds : Int\}$, as before. A typical configuration for deploying this strategy is as before for the Round Counter strategy.

Figure 4.11 shows a sequence diagram for the parallel round counter strategy. At the beginning, $contactPerson_1$ asks each $worker_i$ to create an artifact $InstancesBase_{counti}$ to store counter examples. Then it sends the current model to each $Classifier_i$, asking each $worker_i$ to search for counter examples of it. Counter examples are gathered in parallel in $InstancesBase_{counti}$. Once this is done, the workflow continues in the same way as in the round counter strategy (Section 4.4.2).

4.4.4. GPU enhanced Windowing strategies

The Algorithm 4.4.2 describes the basic Windowing method. Although the stopping condition may vary, the traditional criterion is to stop when no more counter examples are found.

Algorithm 4.4.2 *The basic Windowing algorithm.*

```
1:  function WINDOWING(Exs)
2:      $Window \leftarrow sample(Exs)$
3:      $Exs \leftarrow Exs - Window$
4:      repeat
5:          $stopCond \leftarrow true$
6:          $model \leftarrow induce(Window)$
7:          for $ex \in Exs$ do
8:              if $classify(model, ex) \neq class(ex)$ then
9:                  $Window \leftarrow Window \cup \{ex\}$
10:                 $Exs \leftarrow Exs - \{ex\}$
11:                 $stopCond \leftarrow false$
```

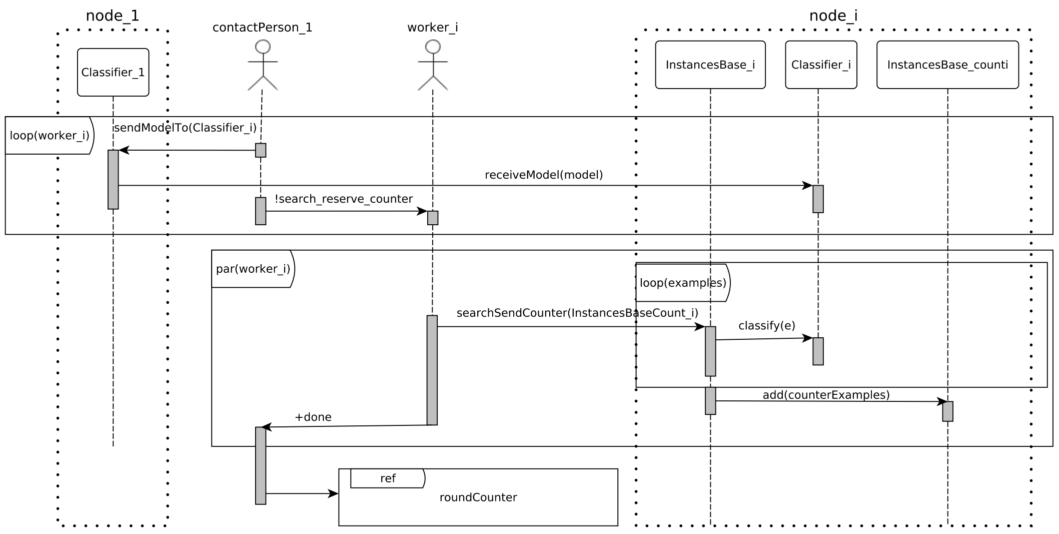


Figure 4.11.: The parallel round counter strategy sequence diagram. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i). Recall this process is part of the general Windowing based strategy shown in Figure 4.7.

```
12:     until stopCond
13:     return model
```

Windowing was criticized because the learned models were not only unable to significantly outperform the traditional centralized approaches, but an extra computational cost resulted of the search for counter examples. Nevertheless, the method can achieve significant run-time gains in noise-free domains [35] and reduces significantly the number of training examples used to induce the models.

The Algorithm 4.4.2, involves two main subprocesses repeated iteratively: the model induction (Line 6); and the search for counter examples (Lines 7–11). It was found that, when large amounts of data are involved, reducing the number of examples used in the induction can potentially boost time performance, even if the process is repeated iteratively; but, for this to happen, the searching for counter examples must also be accelerated using GPUs.

Related to GPUs, there are efforts to induce decision trees using GPUs [52, 81]; and different frameworks that try to boost time efficiency of Data Mining process through GPUs have been proposed [54, 86], but distributed settings have not been considered. Using JaCa-DDM further enhances the performance of the processes and helps to overcome GPU memory limitations.

Enhancing the inductive process

There are two ways of improving the time performance of the inductive process: altering the inductive algorithm itself, e.g., using parallel computing, incremental computing, GPU boosting, etc.; and keeping the size of the window as small as possible. The second approach is adopted here, while the first one is considered for future work.

The proposed enhancement exploits the fact that some counter examples seem redundant in the following sense: suppose a decision tree is computed with a given window and the remaining set of training examples are classified as shown in Fig. 4.12. In order to enhance such a tree, Windowing adds all the counter examples to the window to execute a new inductive process. This happens in all the leaves of the tree. Now, if two counter examples reach the same leaf when classified, they are alike in the sense that they were misclassified for similar reasons, i.e., their attributes values are similar.

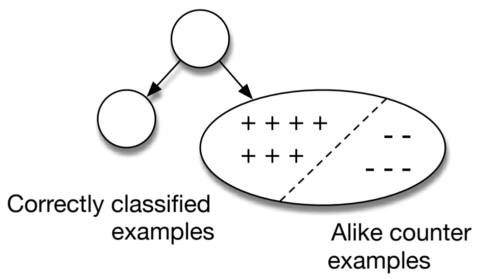


Figure 4.12.: Alike counter examples (-) are those that reached the same leaf when classified. Correctly classified examples (+) are not considered in the iteration, while some alike counter examples are added to the window in each iteration.

Since smaller windows mean faster inductions, is hypothesized that it is unnecessary to add all the alike counter examples to the window at once, in order to obtain the desired accuracy levels, faster. Three parameters are proposed to control the number of alike counter examples being added to the window in each iteration:

- μ defines the percentage of randomly selected counter examples per leaf to be added to the window;
- Δ defines a percentage increment of μ, applied at each iteration and;
- γ defines the minimun number of examples per leaf, required to apply any filtering at all.

With these parameters, the function to know how many counter examples are sampled for each tree node in any given iteration is the following:

$$keep(C,i) = \begin{cases} |C| & \text{If } |C| < \gamma \ \vee \ \mu + incr(\mu) \geq 1 \\ |C| \times (\mu + incr(\mu)) & \text{Otherwise} \end{cases}$$

Where: C is a set of counter examples in a given node; i the current iteration starting at 0; and $incr(\mu) = i \times \Delta$. On the first rounds of Windowing, the sets of alike counter examples tend to be big; as the model improves and more leaves are created, these sets become smaller. Given these observations, parameters are set to discard more counter examples at the beginning of the process, and discard less counter examples as Windowing progresses. The method is implemented in GPUs as a part of the parallel search of counter examples.

Enhancing the searching for counter examples process

The searching for counter examples can be accelerated using GPUs, in order to achieve a negligible time cost for this process. This enhancement requires representing the decision trees and the training examples in data structures well suited for CUDA [61], as well as implementing the corresponding classification and filtering algorithms as kernels.

Decision trees have two kinds of nodes: internal and leaf nodes. Internal nodes represent attributes and leaf nodes, class values. Arcs represent a boolean function over the attribute values, and each function over the same node is mutually exclusive. There are three kinds of arc functions, each of them bound to the boolean operators: $\leq$, $>$, $=$. The first two are for numerical attributes, and the last for nominal ones. Given a decision tree, and an unclassified instance, a classification process consist of traversing arcs yielding true values on its function, from the root to a leaf.

When using GPUs, it is good practice to avoid irregular and complex data structures. Scattered memory access is not efficient and affects the performance of the GPU cache memories. It is better to read large blocks of memory in order to exploit coalesced memory access, i.e., combining multiple memory accesses into a single transaction. With these ideas in mind, a plain representation based on one dimensional arrays was adopted to represent GPU Decision Trees. The structure consists on various properties, related to node and arc information:

- NODES_NUM : how many nodes (including leaves) the tree has.
- nodeAttr[NODES_NUM]: contains the attribute index for each node. On the case of a leaf node, it contains the index of the class value.
- nodeLeaf[NODES_NUM]: a unique number that identifies the leaf, an internal node contains value 0.
- nodeArcs[NODES_NUM]: the number of outgoing arcs of each node.
- ARCS_NUM: the number of arcs in the tree.
- arcType[ARCS_NUM]: the evaluation function of the arc: $\leq$, $>$, $=$.
- arcVal[ARCS_NUM]: the evaluation value of the arc.
- arcNode[ARCS_NUM]: the index of the destination node pointed by the arc.

A method that takes a Weka J48 Tree and transforms it to a GPU Decision Tree was implemented in the J48 artifact. A kernel (in CUDA terms) is a function that executes on a device (GPU). Kernels are intended to be executed in parallel, receiving parameters to define the number of threads to be used. The implemented kernels take into account the counter examples filtering process and the counter examples reduction method described before. The implemented kernels include:

- classify : Return the index value of the predicted class of an instance, and the identifier of the leaf node reached.
- searchCounter: Classifies each instance within the instance set in GPU, and if the predicted class is different from the actual class, then it saves the index of the instance in an array as big as the instance set. Each thread receives the number of instances that will process. At the end of its work, each thread also saves the number of counter examples found and the leaf indexes of them.
- countSize: Computes a sum over each thread result of the searchCounter kernel to yield the total number of counter examples found.
- countersPerLeaf: Creates a vector with the necessary number of counter examples per leaf, taking into account a percentage and minimum number of counter examples according to the proposed counter examples reduction method (section 4.4.4).
- genResult: "Shrinks" the array that contains the counter examples indices found by the searchCounter kernel, saving the indices on a new array that is the exact size of the counter examples found.
- filterParallel: Filters out the counter examples from the dataset in the GPU.

Training examples are represented in the GPU as numeric arrays of size n, where each index $0, \ldots, n-1$ represents the value of the attribute with the same index. The last element of the array represents the class value. The searching for counter examples process requires to load the training examples into the GPU at the beginning of the Windowing process. A copy of them is held in the CPU. It is also necessary to determine the number of multi-processors, and the maximum number of parallel threads of each multi-processor, in order to define an ideal number of working threads. The filtering process, from the host's (CPU) point of view, can be summarized in the following steps:

1. Transform the current J48 Tree into a GPU Decision Tree.
2. Load the GPU Decision Tree in the GPU.
3. Invoke the searchCounter kernel on the ideal number of threads.
4. Invoke the countSize kernel on one thread.
5. Use the result from countSize to invoke countersPerLeaf to reserve enough memory on the GPU to save all the counter example indices on an array.

6. Invoke genResult on one thread to fill the array created previously.
7. Invoke filterParallel on the ideal number of threads to erase the counter examples found from the instance set in the GPU.
8. Use the array with counter examples indices, and filter all the counter examples in the CPU to obtain a counter examples instance set.
9. Free the memory not needed anymore on the GPU.

The filtering process is summarized in Fig. 4.13.

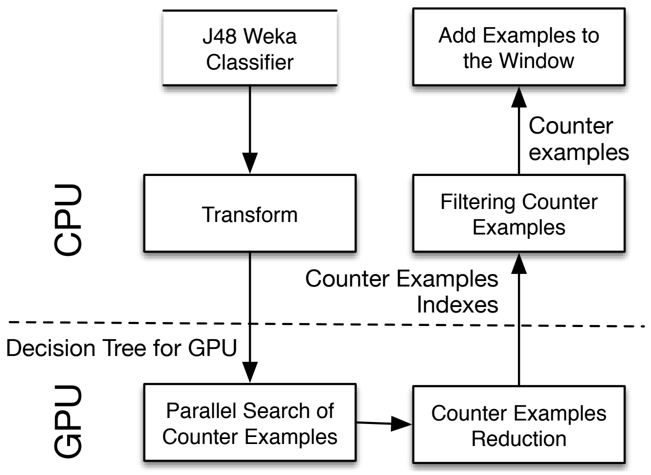

J48 Weka Classifier
Add Examples to the Window
Counter examples
CPU
Transform
Filtering Counter Examples
Counter Examples Indexes
Decision Tree for GPU
GPU
Parallel Search of Counter Examples
Counter Examples Reduction

Figure 4.13.: Counter examples searching executed at each Windowing iteration.

Note that the search process on the GPU only finds index values, the actual filtering is done on the CPU. This design choice was made to reduce data transmission between the CPU and the GPU, and thus improve performance for large datasets.

Parallel Counter GPU strategy

This strategy only implements the discussed GPU search for counter examples improvement. This strategy was introduced in [48]. Following the JaCa-DDM model, the proposed strategy has the following components:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the process and induces the learned model.
  - *worker* gathers counter examples in each distributed node.
  - *roundController* evaluates the termination criterion.
- $Arts = \{Classifier, InstancesBase, Evaluator\}$, where:

- *Classifier*. It is a Weka J48 classifier, extended with GPU capabilities. It is used for inducing decision trees.
- *InstancesBase*. It is a Weka instances base used to store and manipulate training examples. It is also extended with GPU capabilities.
- *Evaluator*. It used to compute the accuracy of a given model, with a set of reserved training examples.

- *Params* include:
  - *Prunning* : *Bool* defines if post pruning is to be used.
  - *WindowInitPerc* : *Double* defines the size of the initial window, as a percentage of the available training examples.
  - *StopTestSetPerc* : *Double* defines the size of the validation set used for computing the termination criterion, as a percentage of the available training examples.
  - *AccuracyThr* : *Double* defines a threshold of minimum acceptable accuracy change between two consecutive rounds. Used by the termination criterion.
  - *MaxRounds* : *Int* defines the maximum number of rounds of the process. It subsumes the termination criterion.

The resulting workflow for one iteration of the process is shown in Fig. 4.14.

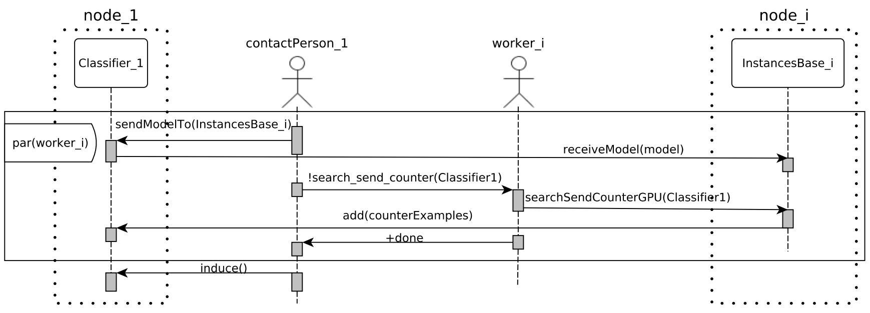


Figure 4.14.: Parallel Counter GPU strategy sequence diagram.

The agent *contactPerson*_1 builds a decision tree using *Classifier*_1 with a subset of the available training examples, i.e., the initial window, and sends the resulting model to the instances base artifacts of each worker. Once this is done, *contactPerson*_1 asks the workers to search for counter examples and send them to the *Classifier*_1 artifact. This searching process is GPU optimized as described before. Once the counter examples are collected, each worker agent sends them to the classifier artifact, in order to enhance the current decision tree with a new induction over the extended window. The process iterates until the termination criterion is met: determining, for a pair of rounds, if the obtained accuracy computed over a validation set, is better enough in the second round.

Parallel Counter GPU Extra strategy

The *extra* in the proposed strategy is due to the control in the number of counter examples aggregated to the window, additional to the GPU search for counter examples previously proposed as Parallel Counter GPU [48]. This strategy was introduced in [49]. Following the JaCa-DDM model, the proposed strategy has the following components:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the process and induces the learned model.
  - *worker* gathers counter examples in each distributed node.
  - *roundController* evaluates the termination criterion.
- $Arts = \{Classifier, InstancesBase, Evaluator\}$, where:
  - $Classifier$. It is a Weka J48 classifier, extended with GPU capabilities. It is used for inducing decision trees.
  - $InstancesBase$. It is a Weka instances base used to store and manipulate training examples. It is also extended with GPU capabilities.
  - $Evaluator$. It used to compute the accuracy of a given model, with a set of reserved training examples.
- $Params$ include:
  - $Prunning$: *Bool* defines if post pruning is to be used.
  - $WindowInitPerc$: *Double* defines the size of the initial window, as a percentage of the available training examples.
  - $StopTestSetPerc$: *Double* defines the size of the validation set used for computing the termination criterion, as a percentage of the available training examples.
  - $AccuracyThr$: *Double* defines a threshold of minimum acceptable accuracy change between two consecutive rounds. Used by the termination criterion.
  - $MaxRounds$: *Int* defines the maximum number of rounds of the process. It subsumes the termination criterion.
  - μ : *Double* defines the initial percentage of counter examples to be collected per leaf, as described before.
  - Δ : *Double* defines an increment percentage for μ applied at each iteration, as described before.
  - γ : *Int* defines the minimum number of counter examples needed in a leaf to apply filtering, as describe before.

The workflow is the same as in the Parallel Counter GPU strategy.

4.5. Learning strategies summary

In this section, a summary of the presented learning strategies is addressed, this summary is in the following characteristics:

1. Is a strategy that can be adopted in distributed settings?
2. Is a strategy that shares training instances or models?. Sharing training instances may have an impact in sensible applications where data privacy is required.
3. Is a strategy that reduces the number of training instances for learning?
4. Is a strategy that can deal with large datasets?
5. Is a strategy that uses online-learning models such as VFDT?, this implicitly allows to share models instead of only training instances.
6. Which is its best quality?, the best quality may be good accuracy, traffic reduction, reduced training instances used for learning, or fast convergence.
7. Following the previous question, Which is its worst quality?.

Table 4.1 shows a summary of characteristics from 1 to 4, and table 4.1 a summary of characteristics from 5 to 7. The summary was done considering each learning strategy description and also taking into account obtained results from section 6.2, and section 7.2, where aspects such as speed of convergence, generated network traffic, accuracy, and number of training instances used where measured. Note that the characteristics are only assessed considering the strategy by itself, not as related with the others.

Table 4.1.: Learning strategies summary in regard of distributed, sharing type, and reduction of instances used. A ✓means that the strategy complies with the characteristic while a ✗means it does not.

Strategy	**Distributed**	**Sharing type**	**Less instances**	**Large datasets**
Centralized J48	✗	None	✗	✗
Centralized VFDT	✗	None	✗	✓
Centralized Bagging	✗	None	✗	✗
Centralizing J48	✓	Instances	✗	✗
Centralizing VFDT	✓	Instances	✗	✗
Round	✓	Models	✗	✓
Distributed Bagging	✓	Models	✗	✓
Distributed Random Forest	✓	Models	✗	✓
Counter J48	✓	Instances	✓	✗
Counter VFDT	✓	Instances	✓	✓
Round Counter	✓	Models	✓	✓
Parallel Round Counter	✓	Models	✓	✓
Parallel Counter GPU	✓	Instances	✓	✓
Parallel Counter GPU extra	✓	Instances	✓	✓

Table 4.2.: Learning strategies summary in regard of large datasets, online model, best quality, and worst quality. A ✓means that the strategy complies with the characteristic while a ✗means it does not.

Strategy	**Online model**	**Best quality**	**Worst quality**
Centralized J48	✗	Accuracy	Speed
Centralized VFDT	✓	Speed	Accuracy
Centralized Bagging	✗	Accuracy	Speed
Centralizing J48	✗	Accuracy	Traffic
Centralizing VFDT	✓	Speed	Traffic
Round	✓	Traffic	Accuracy
Distributed Bagging	✗	Accuracy	Traffic
Distributed Random Forest	✗	Speed	Accuracy
Counter J48	✗	Instances used	Speed
Counter VFDT	✓	Speed	Accuracy
Round Counter	✓	Instances used	Speed
Parallel Round Counter	✓	Instances used	Traffic
Parallel Counter GPU	✗	Accuracy	Speed
Parallel Counter GPU extra	✗	Instances used	Accuracy

The tables are just a brief reference of each learning strategy, the actual behaviour of the strategy may vary depending on the dataset. A more in depth analysis of each learning strategy for actual cases is presented in the next part.

5. Validation and analysis methods

In this chapter, related methods used in our experiments and result analysis are addressed.

First, to know if a given classification method, in this context learning strategy, yields good results, it is necessary to test it in different scenarios, using different criteria (accuracy, time of convergence, etc.), and having a proper validation scheme. Traditional schemes of validation use a common dataset, which it is split in training and testing sets. The training set is used to train a model, and test set is used to test it, obtaining different values for the different criteria at the end. A well established validation schema is cross validation [45], which it is introduced first in this chapter.

Second, in our experimental results, we obtain an accuracy result for each tested learning strategy against each tested dataset, every strategy using the same data samples. It is desirable to employ an statistical hypothesis test, to see if there is a significant differences between the results of the strategies. As the accuracy results obtained may vary, we adopt a non-parametric test, in order to deal with results that do not follow a normal distribution. To this end, the Wilcoxon signed-rank test is introduced in the second part of this chapter.

Finally, as some of the results obtained include various strategies and various datasets, it becomes difficult to interpreter and visualize them, and thus a meta-analysis approach was adopted in the form of forest plots, to better discuss yielded results.

5.1. Cross validation

Cross validation has its origins in the hold-out method. Hold-out consist on the following schema: take a labeled dataset and split it in two parts, one for training and the other for testing. The induction of the learning model uses only the training set, while testing consists on using the test set and see if the learning model predict its instances correctly, i.e.; the predicted label is the same as the actual class label.

From the previous scheme, it can be deducted that it is always better to have a lot of data, this way, induced models will approach more closely to the real problem that the dataset describes, and there will be the possibility to conduct more tests, giving trustworthy results. Unfortunately, it is no always the case that a lot of data is available for training the model. Usually, hold-out uses tow thirds of the available data for training, while the other third is used for testing. With few data it is more probable that the data is not representative. To mitigate this problem, it is recommended that the proportion of class values in the training and testing set be similar. This process is known as stratification [45].

A way to mitigate any data bias created by the sampling of the training and testing set in the hold-out method, is to repeat the process, training and testing with different random data samples. Each iteration a certain proportion of the data, possibly stratified, is chosen for training, and the rest for testing. The accuracy error estimation for the different iteration is averaged to generate a global accuracy error. This method is known as repeated hold-out [45].

An small variation of repeated hold-out form the basis to cross validation [45]. In cross validation it is possible to decide the number of partitions (folds) that the data will have. For example, for a 4-fold cross validation, the data is divided in four equally sized parts, one of the folds is used for testing while the rest for training, this process is repeated four times, so each fold is chose for testing one time per iteration. At the end, the classification accuracy error is averaged to obtain a global error and standard deviation. Table 5.1 shows the previous example graphically.

Table 5.1.: 4-fold cross validation example.

Iteration 1	Iteration 2	Iteration 3	Iteration 4
Test	Train	Train	Train
Train	Test	Train	Train
Train	Train	Test	Train
Train	Train	Train	Test

Usually, a 10-fold stratified cross validation is used to estimate the accuracy error of a learning model. The 10 folds are recommended due a series of experiments that demonstrate that this number is adequate[45], but it is not a restriction [91]. To obtain a better accuracy error estimation, stratified cross validation can be repeated, ten repetitions is recommended to have a good enough statistical coverage.

5.2. Wilcoxon signed-rank test

The Wilcoxon signed-rank test [90] is a non-parametric, i.e.; it does not assumes a normal distribution, statistical hypothesis test used more commonly to compare two matched samples (but it can also be used with repeated measurements on a single sample), to determine if there is a significant difference between their mean rank. It can be used as an alternative to the parametric method Paired t-test, when a normality assumption has failed [65]. In what follows, the two samples version of the Wilcoxon signed-rank test for a small sample of data is explained, this explanation is an adaptation from [66]. Only the version for small samples is considered as the one for large samples converges to a normal distribution, and it was not used in this work.

Let (X_i, Y_i) be the ith pair in a paired random sample of continuous values of size n drawn from population X and Y with unknown median M_1 and M_2 respectively. Interest may be in testing that the unknown population medians are equal, that is $M_1 = M_2$

or that one population median is equal to at least some multiple of other population median, that is $M_1 = cM_2 + k$, where c $(c > 0)$ and k are real numbers versus appropriate two-sided or one sided alternative hypotheses. First, we find the difference between the paired observations $d_i = x_i – cy_i – k$ for $i = 1, 2, ..., n.$. We then take the absolute values of these differences and rank them either from the smallest to the largest or from the largest to the smallest, always taking note of the ranks of the absolute values with positive differences and those with negative differences. The requirement that the populations from which the samples are drawn are continuous makes it possible to state, at least theoretically, that the probability of obtaining zero differences or tied absolute values of the differences is zero. Now, let $r(|d_i|)$ be the rank assigned to $|d_i|$, the absolute value of the ith difference d_i; for $i = 1, 2, ..., n$.

$$\text{Let } Z_i = \begin{cases} 1, & if \quad d_i > 0 \\ 0, & if \quad d_i < 0 \end{cases} \tag{5.1}$$

The test static T^+ is the sum of the ranks of the absolute values with positive differences:

$$T^+ = \sum_{i=1}^{n} Z_i r(|d_i|) \tag{5.2}$$

In a Wilcoxon signed-rank test, the null hypothesis H_0 states that the difference between the pairs of both samples follows a symmetric distribution around zero. T^+ can be compared with a critical value from a reference table $T^+_{critical}$, based on the number of samples and a significance evidence, a significance of $\alpha = 0.05$ is usually adopted. In a two sided test, the null hypothesis H_0, is rejected if $|T^+| > T^+_{critical}$.

In the context of this work, when comparing the results of two learning strategies for the same dataset, the rejection of the null hypothesis H_0 means that there is a significant difference between strategies, meaning that one of them is better or worst than the other.

5.3. Forest plots

Forest plots is a kind of meta-analysis, which it is a method for statistically combine the results of various studies that are included in a systematic review, to come to a conclusion about their overall effects [19]. In simple terms, a meta-analysis is the analysis of an analysis, used to summarize results.

A forest plot provides a simple visual representation of the amount of variation between the results of a group of studies, as well as an estimate of the overall result of all the studies together [27] . In a forest plot, the results of component studies, usually a mean or median, are shown as squares centred on the point estimate of the result of each study [46]. A horizontal line runs through the square to show its confidence interval, usually a tow tailed 95% confidence interval. The overall estimate from the meta-analysis and its confidence interval are put at the bottom, represented as a diamond. The centre

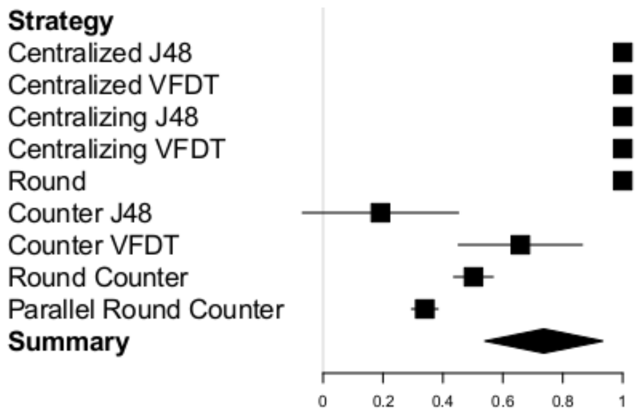


Figure 5.1.: Example of a forest plot. Percentage of instances used for training by various strategies.

of the diamond represents the pooled point estimate, and its horizontal tips represent the confidence interval. Also, as confidence intervals are depicted for each study, if two studies do not intersect horizontally, i.e.; their confidence interval lines do not share any x-axis value, it can be concluded that there is a significant difference between them.

As an example, figure 5.1 shows a forest plot, taken from section 6.3, which analyses the mean of the percentage of instances used for training for a group of strategies given a set of different datasets (18 for this example), with a confidence level of 95%, and normalized on a scale $0 - 1$, which maps to the percentage of training examples used, i.e; 1 means 100%.

The figure shows, for example, that the Centralized J48 strategy always uses 100% of the training examples with no variation, while Counter J48 uses an average of 20% with a considerable variation. As the results of Counter J48 do not intersect horizontally with Counter J48, it can be concluded that Counter J48 has a significant reduction of instances used for training in comparison with Centralized J48; but the same does not apply when comparing Counter J48 against Parallel Round Counter. The summary at the end shows that all methods average to about 75% of training examples used, with a considerable variation, which it is not a surprise since various strategies always used 100% of the training instances, while others have a considerable reduction. Another thing that can be concluded is, for examples, that the Parallel Round Counter strategy is more stable than Counter VDFT as their variation is lower.

Part II.

Experiments

In order to test JaCa-DDM, two sets of experiments were created:

- General experiments: to show, as a proof of concept, how the JaCa-DDM model can be used to implement and test a wide variety of learning strategies. A first group of strategies exploit centralized approaches with strictly benchmark purposes. The second group of strategies addresses centralizing approaches. A third group explores meta-learning. Finally, a fourth group of strategies, exploits Windowing [70], a technique based on building an initial model with some examples, and then enhancing it by reviewing available counter examples. We study the suitability of such strategies in a distributed environment, based on the observation that Windowing reduces the number of training examples used for learning, potentially reducing data communication.

- GPU and large datasets: JaCa-DDM is tested for large and distributed datasets, in particular, using the window-based GPU strategies described in section 4.4.4. Two experimental settings are used to evaluate the strategies, comparing them with the centralized use of the inductive algorithm, and JaCa-DDM strategies based on VFDT, Bagging, and Random Forest. The first setting uses some datasets from well known repositories; the second one is a pattern recognition case study, based on pixel-based image segmentation for the identification of precancerous cervical lesions on colposcopy images.

6. General experiments

6.1. Methodology

In order to evaluate JaCa-DDM strategies, a series of experiments were designed to measure the learned model accuracy; the number of training examples used to build it; the overall process time in seconds; and the traffic of data in megabytes. Some datasets from the UCI [47] and the MOA [11] repositories were selected, varying in the number of instances, attributes, and values for the class. Some of them (covtypeNorm, poker, imdb-D) have more that 100K training instances, to explore scalability while still doing comparisons with the centralized approaches. Table 6.1 describes the selected datasets.

Table 6.1.: Datasets used to explore generalization.

DS	Dataset	Instances	Attributes	Classes
1	adult	48842	15	2
2	australian	690	15	2
3	breast	683	10	2
4	credit-g	1000	21	2
5	covtypeNorm	581012	55	7
6	diabetes	768	9	2
7	ecoli	336	8	8
8	german	1000	21	2
9	hypothyroid	3772	30	4
10	imdb-D	120919	1002	2
11	kr-vs-kp	3196	37	2
12	letter	20000	17	26
13	mushroom	8124	23	2
14	poker	829201	11	10
15	segment	2310	20	7
16	sick	3772	30	2
17	splice	3190	61	3
18	waveform-5000	5000	41	3

All the strategies were evaluated using a ten-fold stratified cross-validation, as presented in section 5.1, with two repetitions over the 18 datasets. Traffic is measured using tcpdump though a *TrafficMonitor* artifact. The experiments ran in a cluster of 8 computers ($node_0, \ldots, node_7$), connected through Gigabit Ethernet at 1Gbps of speed, each one with the following configuration:

- Two Xeon processors at 2.00 GHz with eight cores, and two threads each (32 simultaneous threads).
- 32 GB of RAM

The parameters of each JaCa-DDM strategy are shown in Table 6.2.

Table 6.2.: Experimental parameters. When J48 is used, prunning is adopted.

Strategy	Parameter	Value
Centralized	Classifier	J48
	Prunning	True
	Classifier	VFDT
Centralized Bagging	Threads	32
	NumberOfModels	32
	ClassifierType	J48
	ClassifierParams	Prunning on
Centralizing	Classifier	J48
	Prunning	True
	Classifier	VFDT
Round	*None*	-
Distributed Bagging	ThreadsPerNode	32
	ModelsPerNode	32
	ClassifierType	J48
	ClassifierParams	Prunning on
Counter	Classifier	J48
	Prunning	True
	Classifier	VFDT
	InitPercentage	0.25
	ChangeStep	0.35
	TestPercentage	0.25
Round Counter	Classifier	VFDT
	InitPercentage	0.25
	ChangeStep	0.35
	TestPercentage	0.25
Parallel Round Counter	Classifier	VFDT
	InitPercentage	0.25
	ChangeStep	0.35
	TestPercentage	0.25

6.2. Results

Table 6.3 shows the results for all the datasets. The column *DS* refers to the indexes for the datasets in Table 6.1. Displayed values are the average of 20 runs (two repetitions of a ten fold stratified cross-validation). Time is measured in seconds and traffic in megabytes. Remember that the strategies round, round counter, and parallel round counter use the VFDT learning algorithm.

Table 6.3.: General experiments results. Bold numbers represent the best value for each category

DS	Strategy	Accuracy	Used instances	Time	Traffic
1	Centralized J48	86.106 ± 0.320	43958 ± 0	2.271 ± 0.149	0 ± 0
1	Centralized VFDT	84.455 ± 0.703	43958 ± 0	**0.399** ± 0.182	0 ± 0
1	Centralized Bagging J48	86.167 ± 0.460	43958 ± 0	6.934 ± 0.400	0 ± 0
1	Centralizing J48	86.055 ± 0.290	43958 ± 0	2.415 ± 0.196	11.576 ± 0.197
1	Centralizing VFDT	84.663 ± 0.697	43958 ± 0	0.681 ± 0.391	11.631 ± 0.089
1	Round	84.317 ± 0.692	43958 ± 0	1.098 ± 0.762	**2.307** ± 0.360
1	Distributed Bagging J48	**86.318** ± 0.487	43958 ± 0	1.379 ± 0.365	47.573 ± 0.737
1	Counter J48	85.988 ± 0.927	14291 ± 629	23.431 ± 8.803	12.046 ± 2.895
1	Counter VFDT	83.887 ± 1.919	20801 ± 2175	2.509 ± 1.307	13.403 ± 3.436
1	Round Counter	83.580 ± 2.003	13280 ± 891	15.902 ± 5.153	25.248 ± 4.760
1	Parallel Round Counter	83.215 ± 2.990	**7052** ± 799	9.848 ± 3.317	33.561 ± 9.857
2	Centralized J48	85.362 ± 4.973	621 ± 0	**0.054** ± 0.034	0 ± 0
2	Centralized VFDT	85.797 ± 3.808	621 ± 0	0.064 ± 0.044	0 ± 0
2	Centralized Bagging J48	86.521 ± 3.553	621 ± 0	0.121 ± 0.143	0 ± 0
2	Centralizing J48	85.246 ± 3.720	621 ± 0	0.415 ± 0.338	0.542 ± 0.063
2	Centralizing VFDT	85.217 ± 4.692	621 ± 0	0.346 ± 0.284	**0.532** ± 0.049
2	Round	86.376 ± 2.676	621 ± 0	0.703 ± 0.373	0.682 ± 0.067
2	Distributed Bagging J48	86.449 ± 4.079	621 ± 0	0.439 ± 0.277	1.367 ± 0.074
2	Counter J48	86.159 ± 3.686	219 ± 16	1.849 ± 0.939	1.923 ± 0.544
2	Counter VFDT	**87.101** ± 3.788	208 ± 16	1.301 ± 0.676	1.431 ± 0.314
2	Round Counter	86.594 ± 3.451	165 ± 11	1.724 ± 0.682	1.803 ± 0.408
2	Parallel Round Counter	85.797 ± 3.140	**114** ± 9	2.143 ± 0.728	3.480 ± 0.774
3	Centralized J48	95.388 ± 2.340	615 ± 0	**0.046** ± 0.026	0 ± 0
3	Centralized VFDT	97.361 ± 2.268	615 ± 0	0.081 ± 0.118	0 ± 0
3	Centralized Bagging J48	96.121 ± 1.909	615 ± 0	0.078 ± 0.064	0 ± 0
3	Centralizing J48	95.689 ± 2.685	615 ± 0	0.413 ± 0.315	0.492 ± 0.073
3	Centralizing VFDT	**97.441** ± 1.482	615 ± 0	0.253 ± 0.193	**0.489** ± 0.065
3	Round	**97.441** ± 1.482	615 ± 0	0.755 ± 0.455	0.676 ± 0.075
3	Distributed Bagging J48	96.777 ± 2.103	615 ± 0	0.362 ± 0.224	1.044 ± 0.070
3	Counter J48	94.881 ± 3.193	110 ± 15	1.576 ± 0.781	1.542 ± 0.601
3	Counter VFDT	97.149 ± 1.915	78 ± 24	1.236 ± 0.562	1.141 ± 0.194
3	Round Counter	96.640 ± 2.213	39 ± 4	1.463 ± 0.553	1.326 ± 0.147
3	Parallel Round Counter	97.223 ± 1.821	**38** ± 3	1.979 ± 0.965	2.516 ± 0.547
4	Centralized J48	71.150 ± 3.660	900 ± 0	0.071 ± 0.050	0 ± 0
4	Centralized VFDT	74.900 ± 4.435	900 ± 0	**0.069** ± 0.051	0 ± 0
4	Centralized Bagging J48	74.900 ± 3.093	900 ± 0	0.126 ± 0.170	0 ± 0
4	Centralizing J48	70.900 ± 3.024	900 ± 0	0.343 ± 0.273	**0.710** ± 0.072
4	Centralizing VFDT	**75.100** ± 4.037	900 ± 0	0.455 ± 0.360	0.717 ± 0.064
4	Round	**75.100** ± 4.037	900 ± 0	0.665 ± 0.534	0.733 ± 0.061
4	Distributed Bagging J48	73.050 ± 2.928	900 ± 0	0.456 ± 0.343	2.456 ± 0.075
4	Counter J48	71.000 ± 3.260	571 ± 41	2.648 ± 1.228	3.753 ± 1.478
4	Counter VFDT	72.700 ± 5.611	534 ± 30	1.891 ± 1.006	2.348 ± 0.636
4	Round Counter	73.750 ± 4.632	428 ± 33	2.148 ± 1.111	2.673 ± 0.528
4	Parallel Round Counter	69.850 ± 3.828	**296** ± 35	2.669 ± 1.250	4.358 ± 1.407
5	Centralized J48	94.661 ± 0.127	522911 ± 0	940.861 ± 78.644	0 ± 0
5	Centralized VFDT	77.047 ± 0.309	522911 ± 0	9.536 ± 0.447	0 ± 0
5	Centralized Bagging J48	**96.719** ± 0.089	522911 ± 0	1892.998 ± 414.823	0 ± 0
5	Centralizing J48	94.641 ± 0.106	522911 ± 0	750.028 ± 100.995	424.876 ± 5.222
5	Centralizing VFDT	76.684 ± 0.265	522911 ± 0	9.546 ± 0.513	416.274 ± 0.327
5	Round	76.956 ± 0.325	522911 ± 0	**8.345** ± 2.011	**15.947** ± 1.160
5	Distributed Bagging J48	92.084 ± 0.084	522911 ± 0	53.855 ± 3.378	828.699 ± 2.533
5	Counter J48	92.715 ± 0.512	**166136** ± 6400	3763.525 ± 794.745	486.69 ± 65.13
5	Counter VFDT	77.642 ± 0.445	329417 ± 7791	18.285 ± 2.014	329.055 ± 18.637
5	Round Counter	75.141 ± 0.844	250878 ± 6408	180.884 ± 18.544	307.412 ± 16.301
5	Parallel Round Counter	73.125 ± 1.640	172687 ± 5311	119.792 ± 14.532	575.479 ± 55.166
6	Centralized J48	74.085 ± 3.872	691 ± 0	0.080 ± 0.117	0 ± 0
6	Centralized VFDT	75.711 ± 5.596	691 ± 0	**0.057** ± 0.044	0 ± 0
6	Centralized Bagging J48	75.260 ± 5.071	691 ± 0	0.138 ± 0.197	0 ± 0
6	Centralizing J48	74.416 ± 6.073	691 ± 0	0.276 ± 0.215	**0.468** ± 0.060
6	Centralizing VFDT	75.588 ± 4.982	691 ± 0	0.333 ± 0.293	0.477 ± 0.064
6	Round	75.588 ± 4.982	691 ± 0	0.632 ± 0.416	0.644 ± 0.052
6	Distributed Bagging J48	77.025 ± 5.716	691 ± 0	0.326 ± 0.255	1.614 ± 0.066
6	Counter J48	74.284 ± 5.385	421 ± 35	2.023 ± 1.065	1.799 ± 0.642
6	Counter VFDT	**76.100** ± 4.876	375 ± 24	1.801 ± 0.846	1.617 ± 0.402
6	Round Counter	75.787 ± 5.180	303 ± 18	2.110 ± 0.874	2.086 ± 0.426
6	Parallel Round Counter	69.851 ± 13.179	**217** ± 18	2.398 ± 1.473	3.567 ± 0.929
7	Centralized J48	83.614 ± 6.391	302 ± 1	0.069 ± 0.112	0 ± 0
7	Centralized VFDT	85.276 ± 3.900	302 ± 1	**0.053** ± 0.038	0 ± 0
7	Centralized Bagging J48	**85.574** ± 5.203	302 ± 1	0.087 ± 0.102	0 ± 0
7	Centralizing J48	82.290 ± 4.943	302 ± 1	0.328 ± 0.240	**0.394** ± 0.055
7	Centralizing VFDT	84.830 ± 5.727	302 ± 1	0.407 ± 0.352	0.400 ± 0.063
7	Round	84.830 ± 5.727	302 ± 1	0.564 ± 0.329	0.655 ± 0.063

Continues on next page

Table 6.3 – continuation from previous page

DS	Strategy	Accuracy		Used instances		Time		Traffic	
7	Distributed Bagging J48	81.114 ±	6.284	302 ±	1	0.448 ±	0.311	1.699 ±	0.06
7	Counter J48	81.568 ±	4.604	129 ±	8	1.566 ±	0.708	1.522 ±	0.580
7	Counter VFDT	82.745 ±	6.713	110 ±	9	1.389 ±	0.962	1.235 ±	0.210
7	Round Counter	76.029 ±	19.124	102 ±	9	1.561 ±	0.814	1.565 ±	0.240
7	Parallel Round Counter	76.934 ±	19.264	**83** ±	15	1.804 ±	0.948	2.454 ±	0.791
8	Centralized J48	72.850 ±	4.283	900 ±	0	0.086 ±	0.114	0 ±	0
8	Centralized VFDT	**75.150** ±	3.166	900 ±	0	**0.065** ±	0.048	0 ±	0
8	Centralized Bagging J48	73.500 ±	3.576	900 ±	0	0.130 ±	0.156	0 ±	0
8	Centralizing J48	71.750 ±	3.385	900 ±	0	0.344 ±	0.238	**0.705** ±	0.056
8	Centralizing VFDT	75.000 ±	3.713	900 ±	0	0.417 ±	0.334	0.709 ±	0.064
8	Round	75.000 ±	3.713	900 ±	0	0.622 ±	0.388	0.720 ±	0.060
8	Distributed Bagging J48	72.150 ±	3.297	900 ±	0	0.448 ±	0.241	2.613 ±	0.083
8	Counter J48	70.550 ±	4.570	557 ±	45	2.176 ±	1.582	3.205 ±	1.260
8	Counter VFDT	72.650 ±	3.297	536 ±	18	1.754 ±	0.674	2.277 ±	0.582
8	Round Counter	72.100 ±	4.506	412 ±	46	2.130 ±	0.987	2.361 ±	0.475
8	Parallel Round Counter	71.350 ±	2.580	**299** ±	28	2.839 ±	1.859	4.718 ±	2.670
9	Centralized J48	**99.562** ±	0.301	3395 ±	0	0.168 ±	0.248	0 ±	0
9	Centralized VFDT	96.647 ±	1.573	3395 ±	0	**0.109** ±	0.073	0 ±	0
9	Centralized Bagging J48	99.522 ±	0.305	3395 ±	0	0.156 ±	0.144	0 ±	0
9	Centralizing J48	99.522 ±	0.350	3395 ±	0	0.287 ±	0.158	1.968 ±	0.070
9	Centralizing VFDT	95.558 ±	1.727	3395 ±	0	0.449 ±	0.210	1.972 ±	0.070
9	Round	96.288 ±	1.800	3395 ±	0	0.712 ±	0.393	**0.825** ±	0.108
9	Distributed Bagging J48	98.820 ±	0.616	3395 ±	0	0.350 ±	0.268	1.561 ±	0.07
9	Counter J48	99.483 ±	0.326	**148** ±	9	1.548 ±	0.982	1.379 ±	0.212
9	Counter VFDT	92.827 ±	1.076	608 ±	28	1.852 ±	1.192	2.386 ±	0.523
9	Round Counter	92.046 ±	1.220	430 ±	23	2.478 ±	2.058	2.458 ±	0.570
9	Parallel Round Counter	91.555 ±	2.145	306 ±	29	3.313 ±	1.430	4.913 ±	1.755
10	Centralized J48	63.072 ±	0.425	108827 ±	0	13735.375 ±	347.085	0 ±	0
10	Centralized VFDT	70.243 ±	0.187	108827 ±	0	44.612 ±	0.632	0 ±	0
10	Centralized Bagging J48	Unfinished							
10	Centralizing J48	63.108 ±	0.307	108827 ±	0	14114.590 ±	402.841	224.400 ±	87.603
10	Centralizing VFDT	70.191 ±	0.365	108827 ±	0	**44.539** ±	0.796	**53.816** ±	0.546
10	Round	70.138 ±	0.220	108827 ±	0	49.098 ±	3.123	99.215 ±	11.550
10	Distributed Bagging J48	**70.542** ±	0.097	108827 ±	0	750.707 ±	5.406	220.104 ±	4.215
10	Counter J48	Unfinished							
10	Counter VFDT	70.089 ±	0.172	93317 ±	1518	60.648 ±	2.356	405.965 ±	11.027
10	Round Counter	69.803 ±	1.385	61944 ±	656	165.997 ±	7.577	546.158 ±	145.299
10	Parallel Round Counter	66.997 ±	6.187	**41688** ±	2941	116.698 ±	14.055	887.564 ±	260.205
11	Centralized J48	**99.405** ±	0.496	2876 ±	1	**0.088** ±	0.042	0 ±	0
11	Centralized VFDT	93.507 ±	1.713	2876 ±	1	0.099 ±	0.070	0 ±	0
11	Centralized Bagging J48	99.420 ±	0.479	2876 ±	1	0.143 ±	0.124	0 ±	0
11	Centralizing J48	99.327 ±	0.619	2876 ±	1	0.327 ±	0.203	2.023 ±	0.070
11	Centralizing VFDT	93.694 ±	1.723	2876 ±	1	0.348 ±	0.279	2.024 ±	0.064
11	Round	93.632 ±	1.208	2876 ±	1	0.693 ±	0.405	**0.823** ±	0.070
11	Distributed Bagging J48	96.918 ±	1.174	2876 ±	1	0.475 ±	0.249	2.37 ±	0.108
11	Counter J48	99.327 ±	0.478	**237** ±	21	1.363 ±	0.754	1.712 ±	0.369
11	Counter VFDT	96.417 ±	1.313	514 ±	36	1.594 ±	0.785	2.266 ±	0.540
11	Round Counter	96.526 ±	1.121	389 ±	27	2.530 ±	1.264	2.772 ±	0.693
11	Parallel Round Counter	95.337 ±	1.551	294 ±	20	2.927 ±	0.987	5.151 ±	1.350
12	Centralized J48	87.979 ±	0.535	18000 ±	0	1.421 ±	0.127	0 ±	0
12	Centralized VFDT	71.030 ±	2.090	18000 ±	0	**0.747** ±	0.115	0 ±	0
12	Centralized Bagging J48	**93.727** ±	0.488	18000 ±	0	2.415 ±	0.194	0 ±	0
12	Centralizing J48	87.932 ±	0.731	18000 ±	0	1.559 ±	0.159	5.456 ±	0.131
12	Centralizing VFDT	70.317 ±	4.588	18000 ±	0	1.073 ±	0.314	5.451 ±	0.075
12	Round	69.890 ±	5.482	18000 ±	0	1.538 ±	0.677	**1.763** ±	0.106
12	Distributed Bagging J48	89.175 ±	0.559	18000 ±	0	1.953 ±	0.350	152.690 ±	0.944
12	Counter J48	87.832 ±	1.159	7735 ±	465	29.168 ±	13.589	129.241 ±	55.15
12	Counter VFDT	74.064 ±	1.614	9443 ±	703	4.196 ±	1.129	8.287 ±	1.917
12	Round Counter	73.847 ±	1.700	8410 ±	344	14.855 ±	3.356	16.363 ±	2.259
12	Parallel Round Counter	71.847 ±	1.394	**6795** ±	100	10.074 ±	1.994	27.905 ±	2.237
13	Centralized J48	**100.00** ±	0	7312 ±	1	**0.111** ±	0.119	0 ±	0
13	Centralized VFDT	99.593 ±	0.327	7312 ±	1	0.128 ±	0.097	0 ±	0
13	Centralized Bagging J48	100.00 ±	0	7312 ±	1	0.176 ±	0.157	0 ±	0
13	Centralizing J48	**100.00** ±	0	7312 ±	1	0.384 ±	0.298	3.065 ±	0.065
13	Centralizing VFDT	99.716 ±	0.301	7312 ±	1	0.480 ±	0.327	3.070 ±	0.056
13	Round	99.741 ±	0.225	7312 ±	1	0.849 ±	0.464	**1.173** ±	0.112
13	Distributed Bagging J48	99.833 ±	0.145	7312 ±	1	0.687 ±	0.351	2.259 ±	0.079
13	Counter J48	**100.00** ±	0	**210** ±	17	1.078 ±	0.619	1.222 ±	0.076
13	Counter VFDT	99.526 ±	0.552	649 ±	207	1.473 ±	0.863	2.045 ±	0.365
13	Round Counter	99.981 ±	0.045	257 ±	5	1.994 ±	1.189	1.999 ±	0.188
13	Parallel Round Counter	99.852 ±	0.297	256 ±	7	1.921 ±	1.114	3.879 ±	1.060
14	Centralized J48	**99.788** ±	0.006	746281 ±	0	193.763 ±	20.857	0 ±	0
14	Centralized VFDT	86.497 ±	1.066	746281 ±	0	**4.006** ±	0.410	0 ±	0
14	Centralized Bagging J48	**99.788** ±	0.004	746281 ±	0	548.682 ±	26.580	0 ±	0
14	Centralizing J48	99.787 ±	0.006	746281 ±	0	166.608 ±	13.771	151.383 ±	0.993
14	Centralizing VFDT	87.277 ±	1.965	746281 ±	0	4.807 ±	0.818	149.717 ±	0.206
14	Round	86.772 ±	2.058	746281 ±	0	4.879 ±	1.786	**9.278** ±	0.604
14	Distributed Bagging J48	99.636 ±	0.003	746281 ±	0	28.319 ±	2.621	508.892 ±	7.497
14	Counter J48	99.780 ±	0.015	**48830** ±	1852	77.269 ±	11.117	58.384 ±	5.224
14	Counter VFDT	87.817 ±	2.505	364235 ±	32998	11.608 ±	2.281	105.084 ±	8.391
14	Round Counter	79.232 ±	1.302	341137 ±	16683	230.961 ±	30.135	373.218 ±	21.067

Continues on next page

Table 6.3 – continuation from previous page

DS	Strategy	Accuracy		Used instances		Time		Traffic	
14	Parallel Round Counter	76.561 ±	1.609	227856 ±	7636	136.943 ±	10.370	709.752 ±	39.394
15	Centralized J48	97.056 ±	0.905	2079 ±	0	0.113 ±	0.060	0 ±	0
15	Centralized VFDT	79.696 ±	2.197	2079 ±	0	**0.097** ±	0.062	0 ±	0
15	Centralized Bagging J48	**97.424** ±	0.800	2079 ±	0	0.215 ±	0.215	0 ±	0
15	Centralizing J48	97.359 ±	0.919	2079 ±	0	0.454 ±	0.312	1.043 ±	0.061
15	Centralizing VFDT	79.653 ±	1.815	2079 ±	0	0.433 ±	0.268	1.042 ±	0.063
15	Round	79.653 ±	1.815	2079 ±	0	0.651 ±	0.378	**0.777** ±	0.086
15	Distributed Bagging J48	93.896 ±	1.655	2079 ±	0	0.435 ±	0.303	3.564 ±	0.125
15	Counter J48	95.909 ±	1.321	**364** ±	28	2.000 ±	1.149	2.418 ±	1.069
15	Counter VFDT	83.961 ±	2.164	730 ±	21	1.720 ±	0.690	2.043 ±	0.430
15	Round Counter	84.134 ±	1.778	644 ±	33	2.619 ±	1.256	2.645 ±	0.337
15	Parallel Round Counter	80.562 ±	15.768	489 ±	51	2.600 ±	1.393	4.780 ±	0.916
16	Centralized J48	98.727 ±	0.380	3395 ±	0	0.125 ±	0.070	0 ±	0
16	Centralized VFDT	95.810 ±	1.307	3395 ±	0	**0.105** ±	0.084	0 ±	0
16	Centralized Bagging J48	**98.873** ±	0.529	3395 ±	0	0.194 ±	0.153	0 ±	0
16	Centralizing J48	98.502 ±	0.511	3395 ±	0	0.338 ±	0.197	1.949 ±	0.047
16	Centralizing VFDT	96.168 ±	1.696	3395 ±	0	0.423 ±	0.285	1.977 ±	0.076
16	Round	95.931 ±	1.464	3395 ±	0	0.700 ±	0.348	**0.826** ±	0.114
16	Distributed Bagging J48	98.104 ±	0.614	3395 ±	0	0.376 ±	0.245	1.424 ±	0.061
16	Counter J48	98.621 ±	0.647	**270** ±	26	1.993 ±	1.087	2.264 ±	0.579
16	Counter VFDT	93.584 ±	0.909	498 ±	26	1.349 ±	0.684	1.721 ±	0.346
16	Round Counter	93.266 ±	1.766	383 ±	23	2.360 ±	1.711	2.368 ±	0.680
16	Parallel Round Counter	93.305 ±	1.409	298 ±	29	2.731 ±	1.278	3.720 ±	0.914
17	Centralized J48	94.373 ±	1.622	2871 ±	0	**0.109** ±	0.052	0 ±	0
17	Centralized VFDT	**95.501** ±	1.204	2871 ±	0	**0.109** ±	0.062	0 ±	0
17	Centralized Bagging J48	94.608 ±	1.227	2871 ±	0	0.176 ±	0.137	0 ±	0
17	Centralizing J48	93.808 ±	1.459	2871 ±	0	0.376 ±	0.227	3.061 ±	0.059
17	Centralizing VFDT	95.423 ±	0.934	2871 ±	0	0.468 ±	0.344	3.059 ±	0.066
17	Round	95.423 ±	0.934	2871 ±	0	0.637 ±	0.394	**1.036** ±	0.081
17	Distributed Bagging J48	90.015 ±	1.714	2871 ±	0	0.524 ±	0.318	7.228 ±	0.17
17	Counter J48	93.746 ±	1.730	824 ±	24	2.703 ±	1.000	6.560 ±	1.695
17	Counter VFDT	94.811 ±	1.029	475 ±	52	1.492 ±	0.882	3.03 ±	0.651
17	Round Counter	95.062 ±	1.061	409 ±	24	2.523 ±	1.375	3.078 ±	0.842
17	Parallel Round Counter	94.420 ±	1.227	**338** ±	12	2.994 ±	1.270	6.492 ±	1.675
18	Centralized J48	74.479 ±	2.247	4500 ±	0	0.527 ±	0.141	0 ±	0
18	Centralized VFDT	79.260 ±	2.868	4500 ±	0	**0.182** ±	0.087	0 ±	0
18	Centralized Bagging J48	83.310 ±	1.156	4500 ±	0	0.791 ±	0.164	0 ±	0
18	Centralizing J48	75.120 ±	1.990	4500 ±	0	0.657 ±	0.172	3.119 ±	0.087
18	Centralizing VFDT	80.169 ±	1.592	4500 ±	0	0.486 ±	0.349	3.131 ±	0.061
18	Round	78.930 ±	3.238	4500 ±	0	0.783 ±	0.455	**0.881** ±	0.073
18	Distributed Bagging J48	**84.419** ±	1.292	4500 ±	0	0.605 ±	0.313	6.656 ±	0.067
18	Counter J48	74.450 ±	2.154	3185 ±	425	21.545 ±	12.748	18.442 ±	9.300
18	Counter VFDT	77.919 ±	10.437	1752 ±	79	1.824 ±	0.932	3.069 ±	0.484
18	Round Counter	80.270 ±	2.079	1580 ±	84	3.756 ±	1.694	4.150 ±	0.794
18	Parallel Round Counter	80.739 ±	1.694	**853** ±	93	3.882 ±	1.432	6.318 ±	1.287

6.3. Discussion

Results are summarized using forest plots, as presented in section 5.3, with a confidence level of 95%, and normalized on a scale $0 - 1$. With the exception of accuracy (which normalization is obvious), the normalization is based on ratios, i.e., for each dataset, the greatest value produced by a strategy is 1 and the rest of the values are proportional.

Figure 6.1 (left) shows the results for accuracy. Without surprise, the centralized and centralizing strategies (centralized, centralized Bagging, centralizing, and round) always obtain comparable accuracies for a given learning algorithm. This is due to the fact that they are using all the available examples in the system, being centralized Bagging the overall best strategy, in accuracy terms. For the Windowing-based strategies, the counter strategy produced the best overall accuracy, in much cases comparable with the corresponding centralized ones. Indeed, there is always a Windowing-based strategy that can obtain accuracies similar to the centralized ones. For the Bagging strategies (centralized Bagging, and distributed Bagging), the obtained accuracies are very similar. We expected to see a wider difference between centralized and distributed Bagging, considering that in the distributed version the Bagging meta-learners were induced with

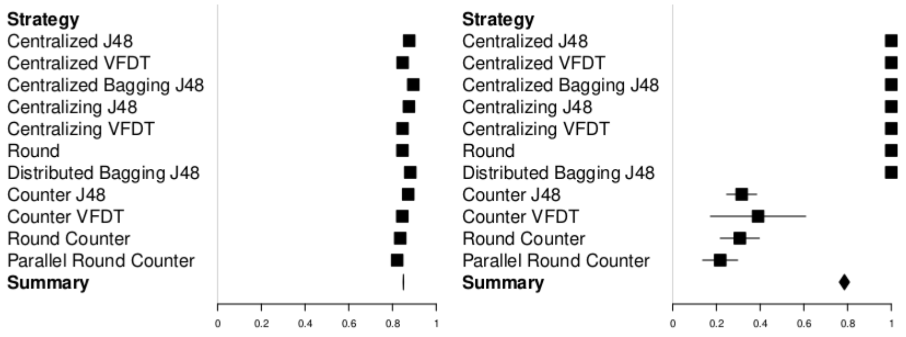


Figure 6.1.: Results for accuracy (left) and ratio of training examples used (right). Higher values for accuracy are better, while lower values of training examples used are better. Summary is the overall value, commonly added in this kind of plots.

incomplete local data, but the stratified distribution (not necessarily biased) of the data as well as the many internal models (256 in total) seem to mitigate the problem. The variability in the accuracy obtained by each strategy is very low (see the detailed results in the table at the end of this section). Because of this, the confidence interval is so small, that it is covered by the black boxes in the forest plot. Accuracy variability among the strategies seems to be due to the learning algorithm used in the strategies. This is more evident in the multivalued class datasets: covtypeNorm (DS 5) , letter (DS 12), poker (DS 14), and segment (DS 15), where J48 gets better results than VFDT. For binary classes, VFDT behaves slightly better than J48. We think that the source of this problem is that VFDT, although incremental, can not move a node once it is created [60], when updating the model. An extension addressing this problem, known as CVFDT [41] will be evaluated in future work.

With respect to the number of training instances used to learn, Figure 6.1 (right) shows that all Windowing-based strategies achieve an important reduction in the number of used instances. The parallel round counter strategy seems to obtain the best reduction, but this is payed with its lower accuracies. The centralized, centralizing, and meta-learning strategies, use always all the available data in the system, so that there is no variability for them. Overall, the Windowing-based strategies obtained accuracies similar to those obtained by the centralized strategies, while reducing significantly the number of instances used to learn.

The results also suggest a negative correlation between the ratio of training instances used to obtain a model and its accuracy, independent of the chosen learning algorithm. This is depicted in Figure 6.2, where the series are strategies and the points are datasets. Each point has then the accuracy obtained given a strategy; and a ratio of used training instances (total number of instances/used instances). If a good accuracy can be achieved, then the percentage of used training instances decreases, and *vice versa*. Considering all the data points, the Pearson correlation coefficient [9] between accuracy and percentage of training examples used is −0.8175845 which establishes a negative correlation. The correlation plot is depicted in figure 6.3. From this observation, a future work, described in section 9.1, is planned.

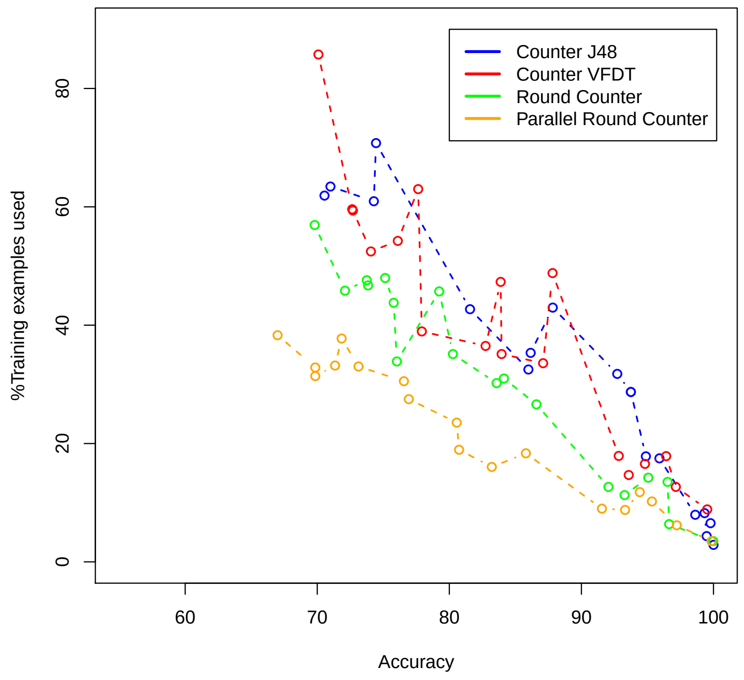


Figure 6.2.: Suggested correlation between accuracy and number of examples used. Each point plotted represents the result for a dataset with the particular strategy.

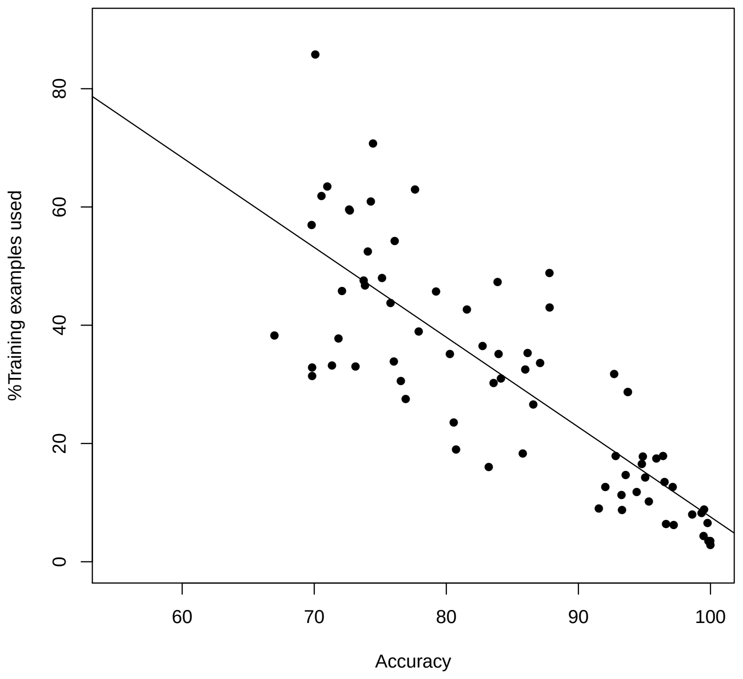


Figure 6.3.: Correlation plot of accuracy and number of examples used, considering all data points. The Pearson correlation coefficient is −0.8175845.

Regarding time, Figure 6.4 (left) shows that the centralized strategies are faster than the Windowing-based, and centralizing strategies, as expected, since they do not transmit any data. Although having data transmission, distributed Bagging is comparable in time terms with its centralized counterpart, since each distributed node uses only 1/8 of the data, the induction processes speedsup for the J48 learning algorithm, this is more apparent in datasets with a large number of instances: covtypeNorm (DS 5), imdb-D (DS 10), poker (DS 14). The round strategy, i.e., moving the model instead of the instances, is a little bit slower than the centralizing strategy. Speed depends also on the adopted learning algorithm, VFDT is much faster than J48.

Windowing-based strategies are slower in general, but this is more notorious with small datasets where gathering counter examples does not pay in time. However, observe (table 6.3) that for a big dataset as poker (DS 14), counter J48 is almost twice faster than centralized J48. The results for imdb-D (DS 10) seems counterintuitive in this sense, since counter J48 did not finished its process, but this is due to the number of attributes of this dataset (1002) and the fact that J48 is not an incremental algorithm, i.e., it has to build a new tree from scratch, when considering new counter examples. Also, the Centralized Bagging strategy using J48 did not finished its process for the

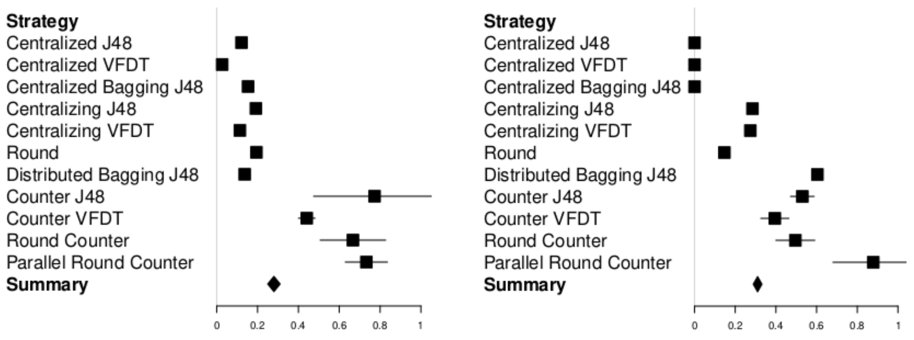


Figure 6.4.: Results for the ratio of process time (left), and ratio of generated traffic (right). For time and traffic, lower values are better. Summary is the overall value, commonly added in this kind of plots.

imdb-D (DS 10) dataset, due to a lack of available memory. Surprisingly, round counter is slower than counter; and even when, in general, round counter performs slightly better than parallel round counter, on big datasets (for example, DS 14) the contrary happens. Counter VFDT has the best overall time of the Windowing-based strategies.

Regarding data transmissions, Figure 6.4 (right), shows that the Windowing-based strategies transmit more data than the centralizing strategies, despite the fact that they used considerably fewer examples to learn. After all, the instances are not the only data transmitted in the strategies. This is clear in the imdb-D (DS 10) dataset, where the centralizing VFDT traffic is significantly smaller that the centralizing J48 one. The traffic for round counter is in the majority of cases greater than counter, because models are shared constantly between nodes as part of the process, and these models grow bigger as the process progress. Finally, the parallel round counter is the worst in terms of traffic. As expected, the round strategy reduces the traffic with respect to the centralizing strategy, but this is not reflected in speed. The distributed Bagging strategy also has a considerable amount of data transmission, since each of the eight distributed nodes builds a Bagging meta-classifier consisting of 32 internal models which are transmitted, with the exception of the centralization node that does not transmits its models.

For the strictly distributed scenarios, round seems to be the strategy with the best overall results. Recall, that it consists in moving the model through the available nodes, and updating it with all the available data. Round maintains an accuracy as good as the centralized strategy, but reduces time and traffic significantly when compared with the majority of the other strategies. However, since it is based on VFDT, it has problems to keep high accuracies when facing multivalued class datasets.

Most of the times, there is a Windowing-based strategy that approximates the accuracy of the centralized strategies, but using significantly less training examples. It is worth noting that more substantial efforts could be done to reduce data transmission on the Windowing-based strategies, but our primary interest was to validate if such strategies could compete in accuracy with centralized approaches while reducing the number of examples used. We believe that the reduction of training examples can be exploited to improve the performance of the learning process in terms of time and data transmission. While the search of counter examples is expensive, this process can be greatly optimized since it is parallelizable, i.e.; it could be approached through GPU computing, expecting that at some point the cost of the process is negligible when compared with the cost of treating with all the data available. The results shed some light regarding how the Windowing-based strategies could be improved, and gave us valuable information to create our GPU-based learning strategies, as presented in section 4.4.4, that optimize time and data transmission while preserving accuracy. Especially, from the results analysis, it became apparent that the window size in each iteration could be reduced in order to speedup the induction process, as explained in section 4.4.4.

7. GPU and large datasets

The interest of this chapter is to test large and distributed datasets, in order to asses how good our proposed Windowing-based approach is in this kind of scenarios. As we already learned from the experiments of chapter 6, non GPU windowing-based strategies may not deal properly with large datasets as they take a considerable amount of time to converge (Counter J48, and Round Counter), or they do not obtain a good accuracy (Counter VFDT and Parallel Round Counter). For this reason, the GPU windowing-based strategies, presented in section 4.4.4, were created, taking as a basis Counter J48, which always obtains a good accuracy, but it is slow due that the Windowing process is not optimized. In this way, the GPU windowing-based strategies optimize the Windowing process, emphasizing accuracy and speed, and making them appropriate for large datasets, as tested in this chapter.

7.1. Methodology

Two experimental settings were adopted for testing the applicability of the GPU strategies, one based on known datasets and another based on a pattern recognition case study: pixel-based segmentation of images for the detection of possible precancerous cervical lesions.

All the experiments evaluate and compare the accuracy of the obtained models, as well as the number of training examples used in the process, the run-time measured in seconds, the complexity of the models (number of leaves and tree size), and the resulting confusion matrix. All the experiments were executed on a cluster of three nodes with the same characteristics:

- Two Xeon processors at 2.40 GHz with four cores, and two threads each.
- 24 GB of RAM.
- One CUDA enabled Nvidia Tesla C2050 GPU with 448 cores and 6 GB of memory.

The Parallel Counter GPU Extra strategy [49] is compared with other strategies with different purposes. It is compared with: Parallel Counter GPU [48] to evaluate the effects of the extra filtering of counter examples; Weka Centralized for comparison with the traditional use of the data mining tool; Centralizing VFDT for comparison with a full centralization approach based on incremental induction; and Bagging and Random Forest for comparison with meta-learning approaches. It is worth noting that all the strategies were implemented using JaCa-DDM.

Centralizing VFDT gathers all the training examples scattered in the distributed nodes; and then induce a decision tree using the VDFT algorithm, as implemented in MOA [11]. The Bagging strategy is based on J48 pruned models, as provided by Weka. In each distributed node, 16 models are created in parallel from bootstrap samples of the local data. The Random Forest strategy runs on the same basis as Bagging, using $log_2(|attributes|) + 1$ as the K value for randomly selecting attributes in each node split.

A Wilcoxon signed rank test, as presented in section 5.2, with a significance evidence at $\alpha = 0.05$ and three possible outcomes (Won, Lost, Tie), is adopted to compare the accuracy of Counter GPU Extra strategy against the considered strategies.

7.1.1. Case study 1: known datasets

Four representative datasets from the UCI repository [47], as adapted by the MOA [11] and TunedIT [92] projects, were selected:

- airlines. A dataset to predict flight delays from the information of the scheduled departure.
- covtypeNorm. A dataset containing the forest cover type for 30 x 30 meter cells obtained from US Forest Service.
- KDDCup99. A dataset to built a predictive model capable of distinguishing between bad connections, intrusions or attacks, and good normal connections.
- poker-lsn. A dataset containing examples of poker and no poker hands drawn from a standard deck of 52 cards.

The datasets properties are shown in Table 7.1. Parameters (π) and agent distribution (δ) are configured as shown in Table 7.2, for all the experiments. A 10-fold stratified cross validation is adopted, i.e., preserving class value ratios in each partition. The training examples were stratified and split evenly among the three nodes, in order to have a fair comparison among the various methods.

Table 7.1.: The properties of the adopted datasets.

DS	**Dataset**	**#Instances**	**#Attributes**	**#Classes**
1	airlines	539383	8	2
2	covtypeNorm	581012	55	7
3	KDDCup99	4898431	42	23
4	poker-lsn	829201	11	10

7.1.2. Case study 2: pixel-based segmentation

Computer Vision (CV) is a discipline which attempts to emulate the perceptual interpretation of images developed by the human eye and brain. CV implies the acquisition

Table 7.2.: Strategy configuration.

Parameters (π)	**Agent distribution (δ)**
Pruning = true	contactPerson, node_1, 1
WindowInitPerc = 0.15	roundController, node_1, 1
StopTestSetPerc = 0.15	worker, node_1, 1
AccuracyThr = 0.004	worker, node_2, 1
MaxRounds = 10	worker, node_3, 1
$\mu = 0.15$	
$\Delta = 0.15$	
$\gamma = 10$	

and analysis of digital images represented as 2D arrays which contain color-spatial features that create different complex patterns. In order to create high level representations from those patterns, machine learning techniques are used. One of the most important challenges in CV is the segmentation of images with classification purposes. Although CV has multiple potential applications, the computationally demanding nature of the state-of-the-art algorithms makes them difficult to apply and ask for more efficient data analysis schemes.

A pixel-based segmentation problem consists on extracting features from labeled pixels to create a training dataset, which in turn is used to construct a model to predict the class of unseen pixels, and in this way achieve image segmentation [87]. As each training example is a pixel, the training set becomes restrictively big and parallel methods are necessary.

This case study deals with sequences of colposcopic images presenting possible precancerous cervical lesions [1]. A colposcopy test consists in the visualization of the cervix through a low power microscope called colposcope. During visualization time, a solution of acetic acid is spread over the cervix surface in order to induce a coagulation process on those cells that are in transformation process due to the cancerous lesion. For the purpose of the case study reported here, a sequence of images was taken during visualization time, in order to capture the temporal color changes on the tissue.

The image sequences were extracted from 38 women patients, ages ranging from 22 to 35 years old. All of them gave informed written consent.

For each patient a range between 310 and 630 aligned images were obtained. The difference on the number of images per patient is due to the fact that some patients where recorded for a longer time than others. A medical expert labeled some of the pixels from the images of each patient (Fig. 7.1 shows an example). Taking into account that the images are aligned, a mask for each patient can be drawn, marking six classes: normal tissue, immature metaplasia, mature metaplasia, ectopy, low grade lesion, and high grade lesion. From these classes, only the last two represent possible precancerous lesion. We are only concerned in recognizing two classes: possible precancerous lesion (+) and the opposite (−), but the six classes are considered in order to exploit class decomposition to deal with class imbalance [83].

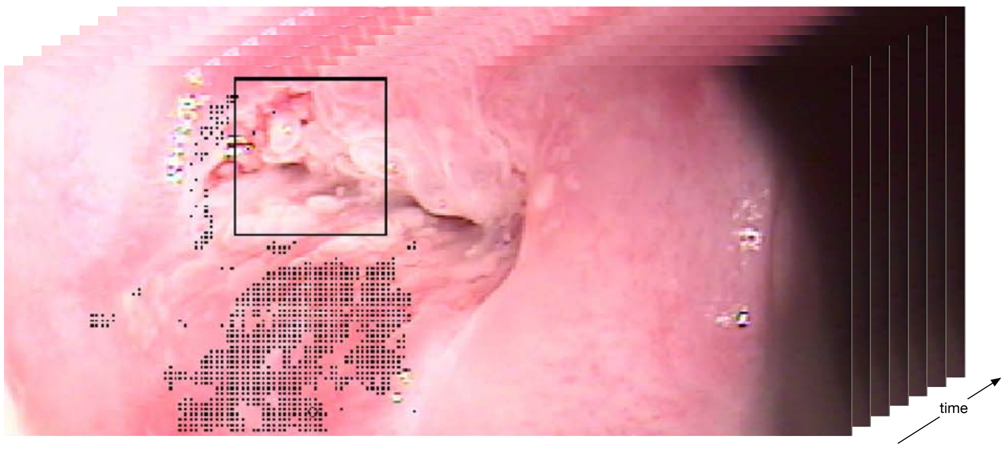


Figure 7.1.: Example of a sequence of colposcopic images. Black dots represent pixels labeled by a medical expert.

As shown in Table 7.3, observations are imbalanced, having substantially more possible precancerous lesion observations (+). Classes are balanced by varying the number of images from the patients, depending on the number of observations of the class. Less observed classes needed more images from a patient with observations of that class. As the minimum number of images in the series is 310, this value is adopted as the maximum number of images that could be considered for a patient. The minimum number of observations of a class is 771, multiplying that value by 310 yields to 239010, which is the maximum number of instances per class that can be obtained in a balanced set. So the total number of instances is $239010 * 6 = 1434060$.

Table 7.3.: The six classes of the case study 2 and the associated number of patients and observations for each one.

Class	**#Patients**	**#Observations**
Normal tissue (−)	11	4294
Immature metaplasia (−)	7	771
Mature metaplasia (−)	5	1264
Ectopy (−)	2	802
Low grade lesion (+)	26	23897
High grade lesion (+)	3	2908

Observe that this process still results in an imbalance of negative and positive classes. The resulting imbalance is intended as a mean to mitigate the problem of fewer patients presenting observations of the negative class, which affects the results when using a leave-one-out evaluation per patient, as the evaluation for patients with negative observations tend to draw a significant percentage of negative examples for testing, creating an important imbalance on the training data, favoring the positive class.

Using FIJI [78], 30 numeric attributes were extracted from the pixels of interest, by applying the following texture filters: Variance, Mean, Minimum, Maximum, and Median. Each filter used a maximum number of 16 neighbors (maximum sigma parameter in FIJI), table 7.4 illustrates the characteristics of the dataset.

Table 7.4.: The properties of the case study dataset.

#Instances	**#Attributes**	**#Classes**
1434060	30	6

Results are evaluated using the leave-one-out method, in order to properly assess accuracy in this case. The test dataset in each case is extracted from a single patient, which means that 38 iterations are performed. For each tested strategy, with the exception of the Centralized one, the training data was stratified and evenly split in the available nodes. Parallel Counter GPU Extra configuration is shown in Table 7.2.

7.2. Results & discussion

Obtained results for both case studies along with its corresponding discussion is addressed next.

7.2.1. Case study 1: known datasets

Table 7.5.: Results for the case study 1: known datasets. In Wilcoxon column, the accuracy of Parallel Counter GPU Extra is compared against each method, a check mark means win, a cross lose, a - a tie, and ! means that the comparison is not possible.

DS	**Strategy**	**Accuracy**	**%Instances**	**Time (seconds)**	**Wilcoxon**
1	**Parallel Counter GPU Extra**	65.36 ± 0.25	**51.21** ± 0.03	435.04 ± 106.28	!
1	Weka Centralized	66.34 ± 0.11	100.00 ± 0.00	1164.66 ± 211.76	✗
1	Parallel Counter GPU	66.26 ± 0.12	94.95 ± 0.01	1810.78 ± 446.47	✗
1	Centralizing VFDT	65.24 ± 0.27	100.00 ± 0.00	**4.67** ± 0.53	-
1	Bagging	66.45 ± 0.13	100.00 ± 0.00	144.67 ± 4.47	✗
1	Random Forest	**66.76** ± 0.11	100.00 ± 0.00	123.82 ± 3.89	✗
2	**Parallel Counter GPU Extra**	92.17 ± 0.52	**43.28** ± 0.04	817.30 ± 253.27	!
2	Weka Centralized	94.59 ± 0.04	100.00 ± 0.00	855.41 ± 97.88	✗
2	Parallel Counter GPU	93.10 ± 0.34	48.44 ± 0.01	1089.03 ± 277.06	✗
2	Centralizing VFDT	76.83 ± 0.35	100.00 ± 0.00	**8.96** ± 0.56	✓
2	Bagging	**94.99** ± 0.10	100.00 ± 0.00	149.35 ± 5.37	✗
2	Random Forest	78.34 ± 0.39	100.00 ± 0.00	44.47 ± 4.38	✓
3	**Parallel Counter GPU Extra**	99.98 ± 0.01	**4.29** ± 0.01	93.23 ± 6.671	!
3	Weka Centralized	**99.99** ± 0.01	100.00 ± 0.00	1688.91 ± 363.89	-
3	Parallel Counter GPU	99.96 ± 0.01	9.28 ± 0.01	199.72 ± 45.62	✓
3	Centralizing VFDT	99.97 ± 0.01	100.00 ± 0.00	**56.17** ± 1.307	✓
3	Bagging	**99.99** ± 0.01	100.00 ± 0.00	371.51 ± 19.39	-

Continues on next page

Table 7.5 – continuation from previous page

DS	Strategy	Accuracy	%Instances	Time (seconds)	Wilcoxon
3	Random Forest	99.97 ± 0.01	100.00 ± 0.00	132.43 ± 21.23	✓
4	**Parallel Counter GPU Extra**	99.53 ± 0.59	**8.80** ± 0.01	22.93 ± 3.51	!
4	Weka Centralized	99.78 ± 0.10	100.00 ± 0.00	174.26 ± 28.55	-
4	Parallel Counter GPU	98.67 ± 0.46	9.56 ± 0.01	24.90 ± 8.05	✓
4	Centralizing VFDT	87.78 ± 1.92	100.00 ± 0.00	**4.25** ± 0.47	✓
4	Bagging	**99.71** ± 0.10	100.00 ± 0.00	64.09 ± 5.49	-
4	Random Forest	96.73 ± 0.25	100.00 ± 0.00	236.34 ± 14.37	✓

Table 7.6.: Results for the case study 1: known datasets (continuation). Model complexity in terms of number of leaves and tree size, i.e.; deepest tree level.

DS	Strategy	#Leaves	Tree Size
1	**Parallel Counter GPU Extra**	91710	94528
1	Weka Centralized	137470	142081
1	Parallel Counter GPU	132767	137210
1	Centralizing VFDT	2553	2711
1	Bagging	53375	54367
1	Random Forest	115822	120367
2	**Parallel Counter GPU Extra**	9519	19038
2	Weka Centralized	14158	28314
2	Parallel Counter GPU	12679	25265
2	Centralizing VFDT	206	427
2	Bagging	75225	15449
2	Random Forest	1007	2014
3	**Parallel Counter GPU Extra**	701	827
3	Weka Centralized	968	1147
3	Parallel Counter GPU	667	855
3	Centralizing VFDT	247	379
3	Bagging	530	652
3	Random Forest	570	693
4	**Parallel Counter GPU Extra**	1929	3793
4	Weka Centralized	2212	4408
4	Parallel Counter GPU	1831	3552
4	Centralizing VFDT	457	877
4	Bagging	2208	4207
4	Random Forest	5917	10232

Table 7.5, and table 7.6 show the results for the case study 1. The accuracy of Parallel Counter GPU Extra is comparable with that of the other methods in all datasets, although being slightly lower for the airlines and covtypeNorm datasets.

Observe that, while preserving accuracy, our strategy reduces the number of examples used for training, up to a 95% (around 49% in the worst case). This results in an important improvement in speed terms, when compared with our previous work, Parallel Counter GPU. Such improvement is due exclusively to the extra filtering proposed, independently of the GPU optimization shared by both strategies. When compared with the rest of the strategies, our proposal is consistently faster than the Weka Centralized approach and even faster than Bagging and Random Forest in some cases. Centralizing VFDT, given its incremental nature, is by far the fastest of the considered strategies, but this is payed with a poor accuracy, even when all the available examples are used to induce the model.

Bagging always maintains a similar accuracy as the Weka Centralized approach, showing a consistent good time performance, which results of working with one third of the data in each distributed node. Bagging seems to overcome the possible bias imposed by data distribution, even if such bias is low in this case, as the data was stratified for distribution. It would be interesting to test this method with truly biased distributed datasets.

The decay of accuracy and time performance shown by Random Forest in some datasets, is likely due to the random selection of attributes at each splitting point, i.e., the most informative attributes could not be selected, tending to create bigger and less accurate trees. Also, being 48 the maximum number of trees in the forest enabled in our setting, there was not improvement in accuracy neither. Furthermore, the resulting bigger trees tended to be slower to induce that those obtained by Bagging.

Due to space limitations, the detailed results about confusion matrix are omitted. The number of leaves and tree size are an indicator of the complexity of the induced trees, but no significant differences were found, with the exception of Random Forest, as explained above; and VFDT that always creates much smaller trees. No significant differences were found for the confusion matrixes, neither. We expected to improve accuracy for the minority classes in the KDDCup99 dataset, but a more refined sampling method, such as the one described in [29], seems to be required.

Observe that the proposed Windowing based method could be used in conjunction with assembly techniques, such as Bagging and Random Forest, i.e., building forests of trees induced using Parallel Counter GPU Extra, which is consistently faster than the Weka Centralized approach traditionally adopted.

7.2.2. Case study 2: pixel-based segmentation

Table 7.7.: Results for case study 2 (n/a means not available). In Wilcoxon column, the accuracy of Parallel Counter GPU Extra is compared against each method, a check mark means win, a cross lose, a - a tie, and ! means that the comparison is not possible.

Strategy	Accuracy	%Instances	Time (seconds)	Wilcoxon	Sen	Spe
Parallel Counter GPU Extra	67.61 ± 19.32	37.00 ± 3.52	3782.26 ± 1094.21	!	60.96	64.83
Weka Centralized	63.68 ± 18.44	100.00 ± 0.00	6436.64 ± 923.16	✓	60.80	61.60
Centralizing VFDT	53.34 ± 20.58	100.00 ± 0.00	32.03 ± 2.61	✓	53.10	58.51
Bagging	64.25 ± 21.78	100.00 ± 0.00	1138.83 ± 108.83	✓	65.40	59.16
Random Forest	58.88 ± 23.71	100.00 ± 0.00	1817.10 ± 179.18	✓	68.78	49.34

Continues on next page

Table 7.7 – continuation from previous page

Strategy	Accuracy	%Instances	Time (seconds)		Wilcoxon	Sen	Spe
Original results [1]	67.00 ± n/a	n/a	n/a	n/a	!	71.00	59.00

Table 7.8.: Results for case study 2 (continuation) (n/a means not available). Model complexity in terms of number of leaves and tree size, i.e.; deepest tree level.

Strategy	#Leaves	Tree Size
Parallel Counter GPU Extra	25085	50169
Weka Centralized	41678	83355
Centralizing VFDT	600	1199
Bagging	18016	36031
Random Forest	28782	57563
Original results [1]	n/a	n/a

Table 7.7, and 7.8, show the results for the case study 2. The original case study results, reported in [1], are also included in comparisons. That work uses a time series approach based on the intensity value of each labeled pixel over time. Each training example represents a spatial positioned signal, i.e., a spatio-temporal pattern, that it is smoothed using a polynomial model. A k-Nearest neighbor approach, with $k = 20$ and Euclidean distance as similarity function, is used for classification. The leave-one-out method is also adopted for evaluation. On the contrary, Parallel Counter GPU is not included, given that the new extra strategy already showed to be consistently faster, while preserving accuracy. Even though, results for that strategy, using a different preprocessing setting, are reported in [48].

As usual, sensibility (*Sen*) is the rate of true positive observations (precancerous lesion) against the sum of true positive plus false negatives, and the specificity (*Spe*) is the rate of true negative (no precancerous lesion) observations against the sum of true negative plus false positives.

When compared to the other considered strategies, Parallel Counter GPU Extra obtains significantly the highest accuracy, based on the Wilcoxon signed rank test computed as before. This may be a consequence of a better integration of examples from the negative classes, thus improving specificity. These examples, although numerous, thanks to the class balance, do not appear in patients as frequently as examples from the positive class, as shown in Table 7.3. This in turn affects the accuracy evaluation in a leave-one-out setting, favoring the positive class, but possibly the Windowing technique helped in this case. It is also interesting to note how our method, while not directly comparable, approaches the results from [1], having the same accuracy, but differing on sensibility and specificity, which it is expected since other preprocessing and techniques were applied.

The wide standard deviation of the accuracy results is an indicator of how different is the accuracy for each patient. For some patients the accuracy is over 90% while for others is about 15%, this happens because of the nature of the data, which it is a common place in medical applications. For some patients there are few observations, and also some classes happen in few patients, being a problem when adopting a leave-one-out approach for testing.

Our method is considerably faster than the Centralized approach, while slower than Bagging and Random Forest, but with significant better accuracy. Centralized VFDT stands out as the faster strategy, but it also has the worst accuracy. Also, the tendency in the model complexity from the case study 1 can be seen in this case as well.

Part III.

Conclusions and future work

This part closes our work giving insights about the most relevant conclusions that can be yielded from this document, taking into account the different ideas and related discussions presented throughout the work. Also, related future work is highlighted, specially addressing current developments in the form of analysis the Windowing technique to asses its applicability as a subsampling technique; a new distributed environment for the JaCa model, which directly impacts on JaCa-DDM; and a related Web interface for JaCa-DDM which development is still in progress.

8. Conclusions

Considering the hypothesis and objectives raised at the beginning of this work, we corroborate that the agents & artifacts paradigm enables the definition of a flexible framework for DDM. This was demonstrated by construction by means of JaCa-DDM, which allows the definition of traditional agent-based approaches, e.g.; centralizing and meta-learning; new approaches, such as the Windowing-based approach introduced and discussed throughout this document; and even traditional Data Mining centralized approaches. JaCa-DDM is supported by a model to implement such approaches in the form of encapsulated learning strategies, that can be deployed in a distributed environment and tested in a standard fashion, thus allowing the comparison of learning strategies, enforcing the research on agent-based DDM methods. Also, the JaCa-DDM model is flexible enough to allow the integration of technologies like GPU computing, which was used to enhance our proposed Windowing-based approach for DDM to deal with large datasets, obtaining an overall good results in our case studies.

In the conclusions that follow, a more in depth discussion of what JaCa-DDM, and the agents & artifacts paradigm in general, mean to agent-based DDM is presented. The discussion has three main points:

- Arguably, the core problem in agent-based DDM is not learning but collaboration [37]. From the best of our knowledge, the opposite is assumed; agent based DDM systems [84, 42, 2, 4, 98] are conceived as deployment systems for distributed learning algorithms. Thus focusing on learning, not in collaboration. In our opinion, the resulting interactions among agents in the cited systems are simpler than the ones promoted by the JaCa-DDM strategies. An exception to this learning centered approach is the formal adoption of workflows in agent based DDM[57, 58]. The agents and artifacts approach used in JaCa-DDM goes in the same direction, enabling to focus on collaborative issues with more appropriate tools.

- JaCa-DDM enforces the aggregation of three independent components: Distributed Data managing, Data Mining algorithms, and Collaborative workflows. Such aggregation is due to the adoption of the agents & artifacts paradigm. Agents are concerned with workflows; the learning algorithms and data managing issues are artifact affairs, which it is not entailed by any other of the discussed agent-based DDM approaches. All this goes beyond of mere division of labour, it is about identifying that some composants of the workflows can be seen as agents, while others are tools for them. Even when this conception of the workflows seems natural, it is far from being the usual approach to agent based DDM. Usually agents are seen as the Data Mining algorithms themselves [84, 53, 42, 5, 2], blurring the distinction between the agents and the tools they can use to learn. Such distinction is relevant, since the theoretical and practical foundations for this composants differ: JaCa-DDM workflows are founded on the BDI constructors provided by Jason, e.g., beliefs, goals, intentions, speech acts messages, plans, actions, events, etc.; While data managing and learning algorithms are supported by the artifacts constructors, e.g, observable properties, signals, operations, linked operations. We are convinced that the agents and artifacts approach adopted in JaCa-DDM promotes clearer designs, making easier the implementation, configuration and deployment of agent based DDM systems.

- In JaCa-DDM, Workflows are clearly defined in terms of interactions between agents and artifacts, i.e.; learning strategies, and so do their deployment. Such definitions are easy to implement, because the JaCa programming tools are based on the agents and artifacts paradigm. Artifacts solve the "agentification" of the WEKA and MOA algorithms and representations, enabling our agents to exploit them. Almost every aspect of JaCa-DDM beyond the basic definitions, can be extended or enhanced. JaCa-DDM has the means to go beyond traditional agent based DDM, where the MAS is merely used as a distribution tool; in order to explore full-fledged agent based contributions for DDM.

With regard of the learning strategies,and JaCa-DDM technology presented in this work, some conclusions can be drawn:

- The JaCa-DDM strategies presented in this work were conceived as a proof of concept, also testing large and distributed datasets, to promote the use of our system for designing, implementing, and testing DDM based workflows. Different approaches such as centralizing, meta-learning, Windowing, and even the coupling with GPU technology are well adapted to the JaCa-DDM model. In particular, Parallel Counter GPU Extra strategy showed to be well suited to DDM scenarios, involving large amounts of data scattered in different sites. The proposed enhancements overcome the time performance issues associated to Windowing based strategies, while achieving similar accuracy results to the centralized approach. The strategy seems also very promissory for real applications such as pixel-based segmentation problems, when combined with a careful image preprocessing. Distributing the data in such cases, improves time performance and reduces memory loads, when comparing to centralized approaches, keeping also a decent overall performance when compared to meta-learning methods, e.g., Bagging and Random Forest.

- JaCa-DDM strategies are encapsulated, with respect to their deployment system. This novel idea is of great value for evaluating the strategies when deployed in different contexts, and also makes possible to implement a learning strategy for a given concern, such as data privacy, or model accuracy. Every learning strategy can have strong and weak points, instead of covering all possibilities. For the learning strategies presented here, table 8.1, taken from section 4.5, shows such strong and weak points, which in turn are of great value to decide what learning strategy to use given a specific scenario. Such is the nature of the encapsulation of learning strategies.

Table 8.1.: Strong and weak points of learning strategies taking as a basis the individual performance of each strategy.

Strategy	**Best quality**	**Worst quality**
Centralized J48	Accuracy	Speed
Centralized VFDT	Speed	Accuracy
Centralized Bagging	Accuracy	Speed
Centralizing J48	Accuracy	Traffic
Centralizing VFDT	Speed	Traffic
Round	Traffic	Accuracy
Distributed Bagging	Accuracy	Traffic
Distributed Random Forest	Speed	Accuracy
Counter J48	Instances used	Speed
Counter VFDT	Speed	Accuracy
Round Counter	Instances used	Speed
Parallel Round Counter	Instances used	Traffic
Parallel Counter GPU	Accuracy	Speed
Parallel Counter GPU extra	Instances used	Accuracy

9. Future work

The experiments and developments discussed in this work, gave us various insights of how to create new learning strategies that are interesting from the agent-based DDM researching point of view. Some ideas in that regard are presented in this section. Furthermore, JaCa-DDM as a technology has various points of improvement, which are also discussed in this section. Finally, future work in progress is presented in corresponding subsections, including an analysis of Windowing as a subsampling technique, an improvement of distributed transparency for the JaCa model, and an enhanced Web-based GUI for JaCa-DDM.

With regard of the proposed Windowing-based strategies, they perform a centralized induction exploiting a distributed search for training examples. While this allows for a single model with a global scope of the data, it also creates a potential bottleneck. An alternative approach would be to distribute the inductive process as well. Indeed, this is the way adopted by meta-learning methods [25], e.g., Bagging and Random Forest, combining the results of a number of separate learning processes in an intelligent fashion. Voting, arbitrating, and combining techniques can be used [69] with this purpose. Nevertheless, a major issue of meta-learning is obtaining models that represent a good generalization of all the data, considering that they have been build from incomplete local data [80]. Adopting Windowing as a subsampling process in meta-learning methods, to see if it can improve the accuracy of meta-learners, as it does not have a random nature in contrast to traditional bootstrap, is a very interesting line of research, further discussed in the next subsection.

Following the previous idea, the distributed Bagging strategy gave us an insight about how the data bias created by equally segmenting a dataset, can be overcame in a Bagging schema by inducing a large amount of internal models. A strategy that combines Windowing and Bagging to further speedup the learning process for large datasets seems to be the next step. This strategy could heavily segment the data while sharing counter examples to further reduce data bias and improve individual models. The mentioned strategy serves to see how JaCa-DDM can be used to further investigate and improve DDM approaches.

As mentioned, at the beginning of this work, in the scope an limitations, our experimental settings produce stratified partitions of the original datasets, distributed in the available JaCa-DDM nodes. Testing the considered strategies when this is not the case would be of interest to evaluate their performance when facing local overrepresented classes. Windowing-based strategies are expected to degrade gracefully in such situations, but the exact impact of such bias need to be established in future experiments. Also, currently, JaCa-DDM is limited to classification Data Mining problems, but it could also be adapted for clustering, or online learning, which it is an envisaged future work. Support for more data sources, as currently only arff files are supported, is also desirable.

Data managing is an issue, an artifacts one. Linking artifacts in different nodes has some nuances and it is not very efficient. This can be faced by redesigning the artifacts deployment, so that linkable artifacts are always together; or enhancing CArtAgo itself, wich it is further discussed in section 9.2. For the sake of simplicity and reuse, artifacts just wrap WEKA/MOA objects and methods, e.g., you can induce a model and classify an instance with a classifier linked to an InstanceBase; you can add, delete and retrieve instances. Sometimes, particularly in the distributed scenarios, a price is payed in efficiency since these objects were not designed with distribution in mind. Nevertheless, it is possible to intervene at the programming level to solve this, e.g., improving instances representation for a faster counter examples gathering. The strategies can also be revisited to work in a less centralized-like terms. Signals and observable properties are not properly exploited in the JaCa-DDM strategies presented here; and interactions among agents are basically message based. More decoupled strategies exploiting awareness of the agents are possible, but they were beyond our goals for this work.

High expertise in MAS, particularly using Jason and CArtAgo, seems to be required to fully exploit JaCa-DDM. A description language, as well as an associated graphic tool for defining the strategies is planned to enhance the usability of the system. Improving the user experience is also desirable, e.g., a Web based interface.

In what follows, a more in depth presentation of ongoing future work is addressed.

9.1. Windowing as a subsampling technique

Understanding how the Windowing process work is key to determine the best ways to exploit it. An interesting find during the experiments from chapter 6, was with regard of our Windowing-based strategies and the reduction of training instances used to learn. The reduction seem to be greater as the accuracy obtained for a specific dataset improved. This negative correlation can be viewed in figure 9.1, which was taken from section 6.3.

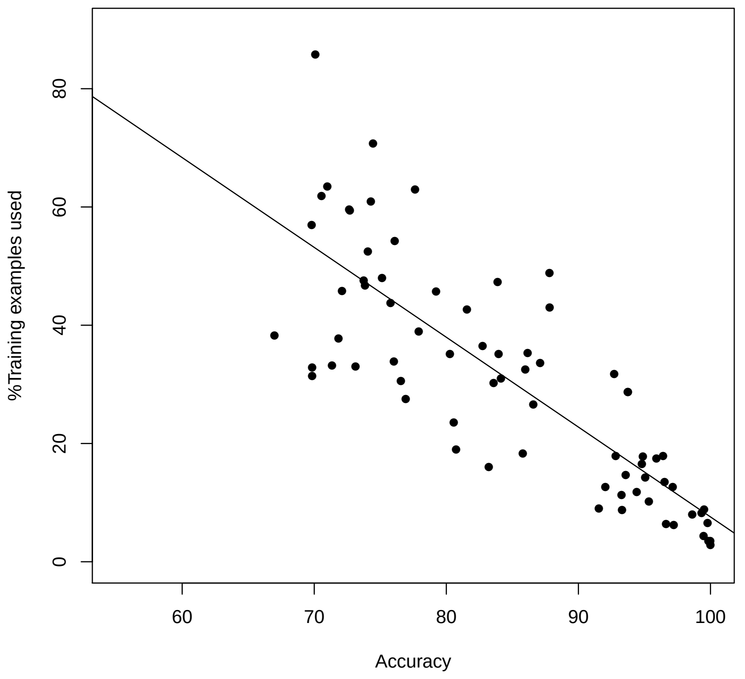


Figure 9.1.: Correlation plot of accuracy and number of examples used , considering all data points, from section 6.3. The Pearson correlation coefficient is -0.8175845.

Possibly, this negative correlation is due to consistent, redundant datasets; where Windowing gathers fewer counter examples, and these counter examples induce high accurate models. We plan to verify this hypothesis using consistent synthetic datasets with a variable amount of redundancy. High redundant synthetic datasets should produce high accurate models with fewer examples than the low redundant ones.

To meet the requirement of this experiment, some Windowing-based strategies, and also the API of JaCa-DDM have been modified in order to make available the final training set used for learning and also the dataset that is conformed from the instances not used. These datasets can also be analyzed to determine its properties, and make comparisons with subsampling techniques, such as bootstrap. From the previous analysis, it is possible to asses the viability of the Windowing as a subsampling technique that can be used, for example, in conjunction with Meta-learning techniques.

In this work, the Windowing-based strategies are coupled with decision trees, this is due that we wanted to concentrate on the learning strategies as a process more than on the actual classifiers, which also allowed us to make fair comparisons. But the technique appears not to be bound to a specific classifier, and we have already tested how the

Windowing technique can be extended to other classifiers such as Naive Bayes [39]. Interest with Naive Bayes is in its ability to endure noise in the data without decreasing accuracy. Also, as Windowing works by only considering not redundant instances, it can potentially be used as a way to mitigate data noise, as noise can not be redundant. The coupling with Naive Bayes and Windowing as a subsampling technique can give us valuable insights in the matter of data noise which we'll try to further investigate.

9.2. Extending the JaCa distributed capabilities

In the process of creating and working with JaCa-DDM, various shortcomings of the JaCa technology were found, these shortcomings are with regard to distributed settings, being the most important the fact that local and remote workspaces are defined and treated differently, which derives in the following problems: i) There is not distributed transparency for agents, being forced to directly manipulate network information, making network distinctions between workspaces. ii) The underlying environment topology is difficult to represent and exploit by the agents as it does not follow any structure or workspace relations beyond the sharing of the same node. All of these problems have the consequence of reducing the abstraction level in which agents work, impacting flexibility and environment exploitation as well. The impact for JaCa-DDM is in the way learning strategies are defined, as the programmer has sometimes to take directly care of IP addresses, which may be cumbersome, reducing flexibility.

Another problem is the lack of proper configuration facilities to allow the inclusion of remote workspaces information at deployment time, meaning that host information for remote workspace spawning need to be hard-coded on the agent programs or externally supported. To spawn a remote workspace, a CArtAgO node needs to be running on the destination host, and there is not any integrated facility to manage them automatically when needed. Furthermore, the current distributed implementation does not exhibit any degree of fault tolerance, this is specially important for possible network connection problems that may arise in a distributed system.

With the awareness of the mentioned problems, a synergy between the research group of JaCaMo [12] and JaCa-DDM was created. JaCaMo is the result of the orthogonal composition of three technologies for MAS: Jason [13] (taken as a proper name inspired by Greek mythology), CArtAgO [74] (Common ARTifact infrastructure for AGents Open environments), and MOISE [40] (Model of Organisation for multI-agent SystEms).

In this section, a proposal to solve the identified problems is presented. A separation between environment and infrastructure is suggested. The environment is represented as a hierarchical tree structure, which represents the topology. In this tree, each node is a workspace which actual physical placement on the distributed system is irrelevant. Workspaces may be deployed in different places, but for the agents point of view, it only matters their placement in the topology. A workspace may be the logical parent of another one, multiple workspaces can be in the same physical place, and there is no restriction about how the topology may be organized, e.g.; workspaces on the same physical place may be on different branches. This allows to organize environments as it is usually done in CArtAgO, but in a more structured way, also supporting remote workspaces transparently.

In a practical sense, each workspace in the tree is represented by a path starting at the root workspace, these paths brings the notion of logical placement that agents require to organize and exploit their environment. We adopt a Unix-like path format to represent this placement, but using a "." instead of a "/", following Jason syntax. These paths are used by the agents to execute environment related actions, such as creating new workspaces or joining one. From the proposed JaCaMo API, there is no difference between local and remote actions related to workspaces. For example for joining local and remote workspaces, which it is related to figure 2.6; with the proposal, a workspace topology would be created, a possibility is to have $workspace2$ and $workspace3$ as direct descendants of the root workspace *main*, with this setting the associated code snipped is as follows:

```
    joinWorkspace("main", WspId1);
    joinWorkspace("main.workspace2", WspId2);
```

As in current CArtAgO, agents may work in multiple workspaces at the same time, but the concept of current workspace is dropped since in actuality all the joined workspaces should be considered the current context of working. Nevertheless, agent may specify the target workspace for an action. A new introduced concept is the home workspace of an agent, which it is the workspace where the agent is initially deployed, serving as a relative reference to other places in the topology, providing a default place for the agent, and also serving as the default workspace to execute actions when a target workspace is not specified.

With regard to the infrastructure, a layer is added to manage distribution, this layer provides the required services for the agents to exploit their distributed environment. These services include: i) Workspace management, so agents can create, join, and quit workspaces no matter their physical placement; ii) Topology inspection, so agents can reason about the current topology organization and do searches concerning workspaces; iii) Workspace dynamics observation, so agents can know when other agents manage workspaces, or when workspaces disconnect and reconnect after a network problem; iv) Disconnection and fault tolerance to manage and recuperate from network problems, which it is currently left as future work. We believe that the set of mentioned services do not only bring distributed support, but also enhance the dynamics of MAS in general, extending its possibilities.

9.2.1. Formal description

JaCaMo assumes an endogeneous approach to MAS, i.e., the environment is an explicit part of the system:

Definition 9.2.1 *A MAS is composed by a set of agents (Ags), their environment (Env), and an infrastructure ($Infr$) running both of them:*

$$MAS = \{Ags, Infr, Env\}$$

The set of agents is composed by $n \geq 1$ agents:

$$Ags = \{a_1, \dots, a_n\}$$

Each agent, as usual, is composed by beliefs, actions, and other elements equal to:

$$a_i = \{bels, acts, \dots\}$$

By default, when created, an agent includes minimally:

$$a_i = \{joined(home)\}, \{join, quit, create\}, \dots\}$$

which means that every agent believes he has joined a home workspace, and has actions to join, quit, and create workspaces; and update the information about the environment.

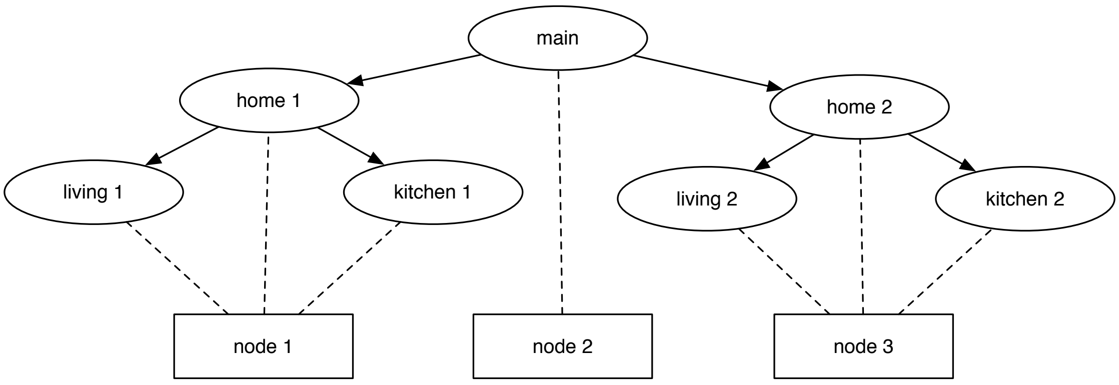


Figure 9.2.: The intented view of an endogeneous environment.

Figure 9.2 illustrates the intended view of the environnment in this proposal. First, the environment, properly speaking, is a tree of workspaces, expressing a kind of spatial relation among workspaces, e.g., the kitchen 1 is at the home 1. Second, nodes and hosts are not part of the environment, but are defined as part of the infrastructure of the MAS, nevertheless, workspaces keep information about its corresponding physical node.

The infrastructure is a layer hidden to the agents, that gives the low level support to distribution, formally defined as:

$$Infr = \{Nodes, Hosts\}$$

where:

- $Nodes = \{node_1, \ldots, node_k\}$ is a set of CArtAgO nodes, i.e.; processes, possibly remote, where workspaces can be created. Each $node_i$ is a tuple $\langle ni, SWsps, hi, port \rangle$, where ni is an unique identifier for the node; $SWsps \subseteq W$ is the set of spawned workspaces in the node, containing at least a default workspace for the node; hi is an identifier of the host computer where the node exists; and $port$ is the host port used by the node process.

- $Hosts = \{host_1, \ldots, host_p\}$ is the set of available computer devices on the distributed system. Each $host_i$ is a tuple $\langle hi, HNodes, ip \rangle$, where hi is a host unique identifier, $HNodes \subseteq Nodes$ is a set of hosted nodes, and ip is the IP address of the computer.

Formally, the environment Env is defined as a graph:

$$Env = \{W, E\}$$

where:

- $W = \{w_1, \ldots, w_i\}$ is a finite, non-empty set of $i \geq 1$ workspaces that contain artifacts. Each $w_i = \langle idW, name, ni \rangle$, where idW is an unique identifier for the workspace, $name$ is a logical name, and ni is a reference to the CArtAgO node in $Infr$ that contains w_i. The $node$ element establishes a connection between the environment and the infrastructure, in order to forward agent actions to the destined physical place.

- $E \subset W^2$ is a set of edges over the workspaces, such that Env is minimally connected, i.e., it is a rooted tree that represents the environment topology.

For instance, following Figure 9.2, $Env = \{W, E\}$, and considering for simplicity only the name of each w_i, such that:

- $W = \{main, home1, home2, living1, kitchen1, living2, kitchen2\}$
- $E = \{\{main, home1\}, \{main, home2\}, \{home1, living1\}, \ldots\}$

The expression $w_1.w_2 \ldots .w_n$ denotes a path on Env, if:

- $w_i \in W$ for $i = 1, \ldots, n$;
- $\{w_{i-1}, w_i\} \in E$ for $i = 2, \ldots, n$.

Abusing a little bit of the notation, we can write $w_1. \ldots .w_n \in Env$. For instance, $main.home1.living1 \in Env$. Some useful operations over paths, include:

- $last(w_1.w_2. \ldots .w_n) = w_n$

- $butlast(w_1.w_2.\dots.w_{n-1}.w_n) = w_1.w_2.\dots.w_{n-1}$
- $add(w, w_1.w_2.\dots.w_n, Env) = w_1.w_2.\dots.w_n.w$. This involves adding w to W, and $\{w_n, w\}$ to E in Env.
- $del(w, w_1.w_2.\dots.w_n.w, Env) = w_1.w_1.\dots.w_n$. This involves deleting w from W, and $\{w_n, w\}$ from E in Env.

In what follows, the transition rules related to environment agent actions are described, workspaces denote paths in the environment.

Joining a workspace

An agent can ask himself about the workspaces he has currently joined: $ag_{bels} \models joined(w)$, if and only if, w is a workspace currently joined by the agent. Recall that by default $ag_{bels} \models joined(home)$. An agent can join different worspaces concurrently, so that $ag_{bels} \models joined(Ws)$ unifies Ws with a list of the workspaces joined by the agent. Two transtion rules define the behavior of the action *join*. First, an agent can join a worspace w, if and only if w is a path in the environment Env and it is not believed to be already joined:

$$(\mathbf{join_1}) \qquad \frac{join(w) \mid w \in Env \wedge ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag', Env \rangle}$$
$$s.t. \quad ag'_{bels} = ag_{bels} \cup \{joined(w)\}$$

Second, nothing happens if an agent tries to join a previously joined worspace:

$$(\mathbf{join_2}) \qquad \frac{join(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

Any other use of *join* fails.

9.2.2. Quiting workspaces

An agent can quit the workspace w if he believes he had joined w. The agent forgets such belief.

$$(\mathbf{quit_1}) \qquad \frac{quit(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag', Env \rangle}$$
$$s.t. \quad ag'_{bels} = ag_{bels} \setminus \{joined(w)\}$$

If the agent tries to quit a workspace he has not joined yet, nothing happens:

$$(\mathbf{quit_2}) \qquad \frac{quit(w) \mid ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

Creating workspaces

An agent can create a workspace w, if it is not a path in the environment, but $butlast(w)$ is one:

$$(\mathbf{create_1}) \qquad \frac{create(w) \mid w \notin Env \wedge butlast(w) \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env'\rangle}$$
$$s.t. \quad Env' = add(last(w), butlast(w), Env)$$

Observe that the result of creating a workspace must be propagated to the rest of the agents in the MAS. This could be done by the infrastructure, or broadcasting the *add* operation. The actual node where the workspace is going to be created is decided by the infrastructure following a policy, by default the infrastructure spawns the workspace on the same node where its parent workspace is running.

Trying to create an existing workspace does nothing:

$$(\mathbf{create_2}) \qquad \frac{create(w) \mid w \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env\rangle}$$

9.2.3. Implementation

The model introduced on the previous section is open enough to allow different implementations. This section presents a practical possibility, intended to be integrated with JaCaMo. The core implementation and main design choices are related to the general architecture, and configuration and deployment.

General architecture

The architecture to support agent services is based on the concept of Node, which refers to the $Nodes$ element in $Infr$. Nodes represent independent CArtAgO processes, possibly remote, running on a given host ($Hosts$ element in $Infr$), and associated to a port. Nodes are the main abstraction to manage workspaces (W element in Env), and as such, they provide all the necessary tools to create, join, and quit workspaces, as well as the means to communicate with other nodes in order to maintain a consistent workspace topology, and properly dispatch topology related events. The workspace topology corresponds to the E element in Env. A NodeArtifact is the gateway used by an agent to interact with the node services and to observe the distributed environment. There is a NodeArtifact in each workspace, and every agent has access to one of them, which one depends on its *home* workspace, which in turn it is intended to be on the same node as the agent process.

Nodes communicate between each other following a centralized approach: one node is designated as the central node, this is usually the node deployed by default by JaCaMo, so every change on the topology is inspected and approved by a single node, and the associated actions and events are dispatched from there. This centralized approach makes possible to maintain a consistent topology, avoiding run conditions. To exemplify node communication, the workflow for creating a new workspace is the following:

- An agent that wants to create a workspace issues the action to its corresponding NodeArtifact, passing a tree path.
- The artifact checks if the tree path is consistent with the topology tree, if it is, it sends a request to the central node.
- The central node issues a request to the end node where the workspace is actually going to exist. By default, it chooses as end node the same as the parent workspace from the path given.
- The end node creates the workspace and returns control to the central node.
- The central node makes the corresponding changes to the workspace topology and communicates the success to the original requesting node. It also dispatches a create and tree change event to the other nodes.

As the node model is centralized, there exists the concern of a single point of failure, that is why all nodes maintain redundant information about the topology, so it is possible to recuperate from a central node dropping, as any node can take the role of central node. The topology structure is also lightweight, which speeds up the tree synchronization among nodes, this synchronization is only required when there is a change in the topology. This redundancy also allows to boost the efficiency of operations such as joining and quitting workspaces, since those operations only need to read from the topology, so the local copy is used in those cases. Communication with the central node is only required in cases where a change in the topology is required. We believe that in traditional environment management, it is more common for the agents to join and quit workspaces than to create new ones.

MAS configuration and deployment

To ease the deployment of the distributed infrastructure is a goal of our overall proposal, this means to be able to configure and launch the desired hosts, nodes, and workspaces that will take part in the MAS from the start. It is also possible to manually add new nodes after deployment. The idea is to extend the deployment of JaCaMo, where only workspaces are considered. JaCaMo uses a text file known as the JCM file to configure the deployment of the MAS. The intention is to further include fields in this file to also configure host, and nodes for distributed systems; and add the facilities to automatically launch CArtAgO nodes in remote machines through a daemon service.

The changes to the JCM file include:

- Host configuration: assign a logical name and IP address to each host.

- Node configuration: assign a logical name for the node, i.e.; the name of the default workspace; the related host name; and optionally a port number.
- Workspaces configuration: relate each workspace to a specific node.
- Lib file: the path to a jar file containing all the necessary files to launch CArtAgO nodes. This includes custom artifacts binaries, third party libraries, custom classes binaries, and any configuration file. This file is intended to be shared among all nodes.

9.3. JaCa-DDM Web-based GUI

A work is in progress to implement a new Web-based GUI for JaCa-DDM, this new system has the goal to provide a better user experience, making a distinction between configuration and usage, and providing facilities to share learning strategies, artifacts, and datasets, to build a local community for agent-based DDM research. Also, facilities for the execution of batch experiments is added, and the possibility to administer them. The configuration is done as "packages" of experiments.

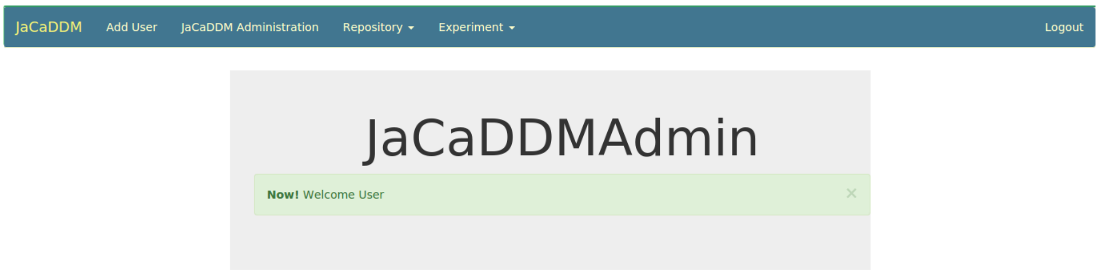
In the system there are tree kinds of user roles:

- Administrator: its major concern is to make available new computer nodes for experiments, experiments and user administration.
- Researcher: this kind of user can share strategies, artifacts, and datasets, as well as create and configure experiments.
- General public: this kind of user is for demonstration purposes, she can only configure and execute some experiments.

The navigation structure of the system is as follows:

- Login screen: to identify the role of the user. Only the administrator can register new users. A general public user does not need to login.
- Configuration: administrator only. To administer computer nodes, experiments, users, and repositories of contributions.
- Repository: to share and download learning strategies, artifacts, and datasets. An user can also update new versions of shared material.
- Experiments: allows to create, configure, save, load, and administer packages of experiments, as well as to access the experiment history with associated partial and final results.

Figure 9.3 shows the main screen for the administrator, which has every option enabled.



Now! Welcome User

Figure 9.3.: Main screen of the new Web-based GUI for JaCa-DDM.

The new system does not change any related JaCa-DDM model, it is just a tool to generate XML configuration files as explained in section 3.2.3. Also, the new GUI is on top of the JaCa-DDM core system, so its development is independent. As an example of some screens, figure 9.4 shows the general wizard to configure an experiment, while figure 9.5 shows the history results from the experiments of an user.

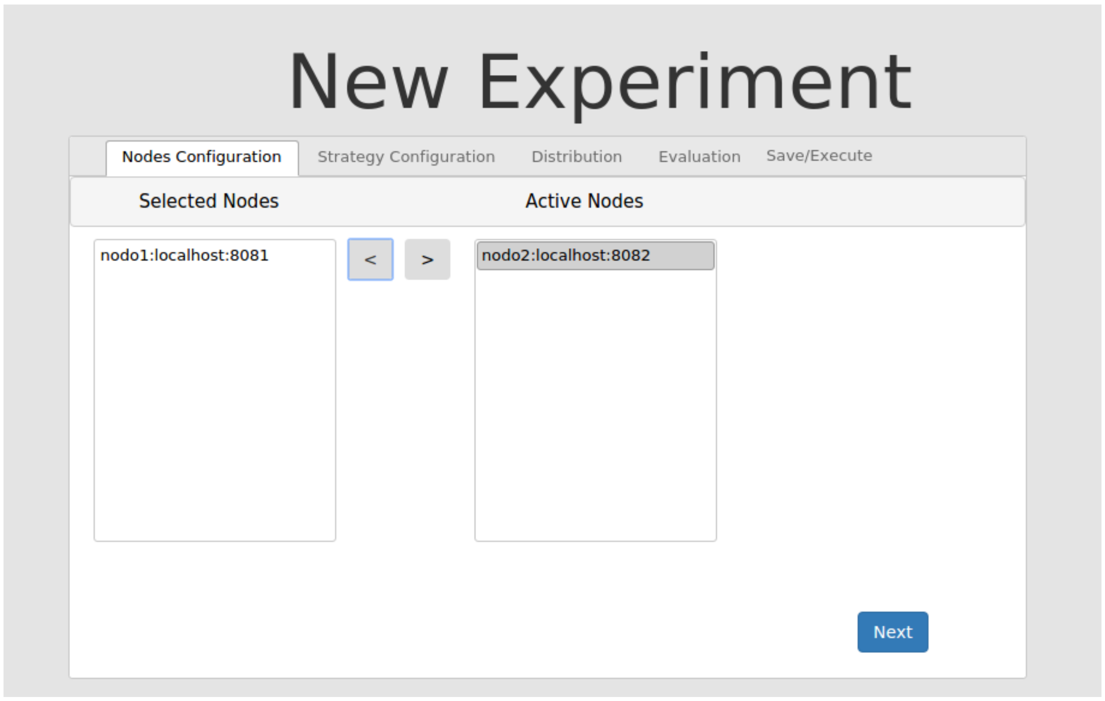
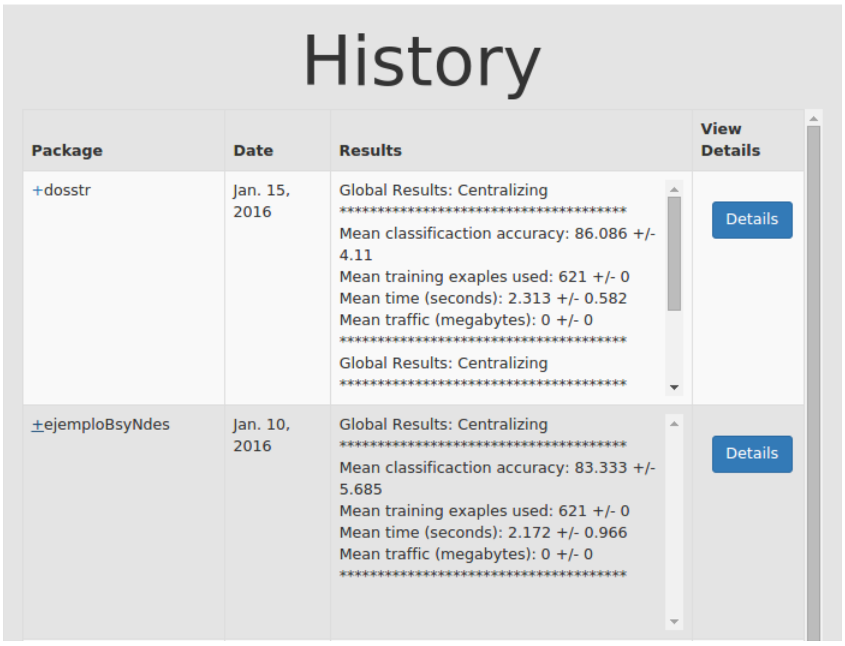


Figure 9.4.: Experiment configuration wizard.

An interesting part of the implementation is the contribution of artifacts. In order for artifacts to work, they may need a third-party library, Java classes, or other kind of files (for example, to work with GPUs kernel files are needed). The distribution of such artifact is on the form of a zip file with an specific folder structure:

- artifact: contains the artifact class.
- lib: contains third party libraries needed.
- classes: contains needed Java classes.
- extra: contains any extra file.

Once an artifact is uploaded, the file is extracted and added to the code base of JaCa-DDM making the artifact available to use by other users in their learning strategies. This code base is then distributed along the various nodes in the system though ssh connections (ssh credential can be configured).



History

Package	Date	Results	View Details
+dosstr	Jan. 15, 2016	Global Results: Centralizing *************************************** Mean classificaction accuracy: 86.086 +/- 4.11 Mean training exaples used: 621 +/- 0 Mean time (seconds): 2.313 +/- 0.582 Mean traffic (megabytes): 0 +/- 0 *************************************** Global Results: Centralizing ***************************************	Details
+ejemploBsyNdes	Jan. 10, 2016	Global Results: Centralizing *************************************** Mean classificaction accuracy: 83.333 +/- 5.685 Mean training exaples used: 621 +/- 0 Mean time (seconds): 2.172 +/- 0.966 Mean traffic (megabytes): 0 +/- 0 ***************************************	Details

Figure 9.5.: History results screen.

Bibliography

[1] Héctor-Gabriel Acosta-Mesa, Nicandro Cruz-Ramírez, and Rodolfo Hernández-Jiménez. Aceto-white temporal pattern classification using k-nn to identify precancerous cervical lesion in colposcopic images. *Computers in biology and medicine*, 39(9):778–784, 2009.

[2] Kamal Ali Albashiri and Frans Coenen. Agent-enriched data mining using an extendable framework. In *Agents and Data Mining Interaction*, pages 53–68. Springer, 2009.

[3] Kamal Ali Albashiri, Frans Coenen, and Paul Leng. *An investigation into the issues of Multi-Agent Data Mining.* University of Liverpool, 2010.

[4] Sung Wook Baik, Jerzy Bala, and Ju Sang Cho. Agent based distributed data mining. In *Parallel and Distributed Computing: Applications and Technologies*, pages 42–45. Springer, 2005.

[5] Stuart Bailey, Robert Grossman, Harimath Sivakumar, and A Turinsky. Papyrus: a system for data mining over local and wide area clusters and super-clusters. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 63. ACM, 1999.

[6] Omar Baqueiro, Yanbo J Wang, Peter McBurney, and Frans Coenen. Integrating data mining and agent based modeling and simulation. In *Industrial Conference on Data Mining*, pages 220–231. Springer, 2009.

[7] Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.

[8] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, Ltd, England, 2007.

[9] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[10] Michael J Berry and Gordon Linoff. *Data mining techniques: for marketing, sales, and customer support.* John Wiley & Sons, Inc., 1997.

[11] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[12] Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.

[13] Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason.* John Wiley & Sons Ltd, 2007.

[14] Ivan Bratko. *Prolog: programming for artificial intelligence.* Addison-Wesley, 2001.

[15] Michael E Bratman. Intention, plans, and practical reason. 1987.

[16] Michael E Bratman, David J Israel, and Martha E Pollack. Plans and resource-bounded practical reasoning. *Computational intelligence*, 4(3):349–355, 1988.

[17] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[18] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] Sarah E Brockwell and Ian R Gordon. A comparison of statistical methods for meta-analysis. *Statistics in medicine*, 20(6):825–840, 2001.

[20] Giovanni Caire, Elena Quarantotto, and Giovanna Sacchi. Wade: an open source platform for workflows and agents. In *MALLOW*, 2009.

[21] L. Cao, G. Weiss, and S. Y. Philip. A brief introduction to agent mining. *Autonomous Agents and Multi-Agent Systems*, 25(3):419–424, 2012.

[22] Longbing Cao. *Data Mining and Multi-agent Integration.* Springer, Berlin Heidelberg New York London, 2009.

[23] Longbing Cao, Ana L. C. Bazzan, Vladimir Gorodetsky, Pericles A. Mitkas, Gerhard Weiss, and Philip S. Yu. *Agents and Data Mining Interaction: 6th ADMI 2010, Toronto, ON, Canada*, volume 5980 of *Lecture Notes in Artificial Intelligence.* Springer Verlag, Berlin Heidelberg, 2010.

[24] Longbing Cao, Vladimir Gorodetsky, Jiming Liu, Gerhard Weiss, and Philip S. Yu. *Agents and Data Mining Interaction: 4th ADMI, Budapes, Hungary*, volume 5680 of *Lecture Notes in Artificial Intelligence.* Springer Verlag, Berlin Heidelberg New York, 2009.

[25] P. K. Chan and S. J. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8(1):5–28, 1997.

[26] William F Clocksin and Christopher S Mellish. *Programming in PROLOG.* Springer Verlag, 2003.

[27] Geoff Cumming. *Understanding the new statistics: Effect sizes, confidence intervals, and meta-analysis.* Routledge, 2012.

[28] Josenildo C Da Silva, Chris Giannella, Ruchita Bhargava, Hillol Kargupta, and Matthias Klusch. Distributed data mining and agents. *Engineering Applications of Artificial Intelligence*, 18(7):791–807, 2005.

[29] Annarita D'Addabbo and Rosalia Maglietta. Parallel selective sampling method for imbalanced and large data classification. *Pattern Recognition Letters*, 62:61–67, 2015.

[30] Fatheme Daneshfar and Hassan Bevrani. Load–frequency control: a ga-based multi-agent reinforcement learning. *IET generation, transmission & distribution*, 4(1):13–26, 2010.

[31] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[32] Hamdi Ellouzi, Hela Ltifi, and Mounir Ben Ayed. An intelligent agent based architecture for visual data mining. *International Journal of Advanced Computer Science & Applications*, 1(7):151–157, 2016.

[33] T. Finin et al. An overview of KQML: A knowledge query and manipulation language. Technical report, University of Maryland, CS Department, 1992.

[34] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[35] Johannes Fürnkranz. Integrative windowing. *Journal of Artificial Intelligence Research*, 8:129–164, 1998.

[36] Robin Genuer, Jean-Michel Poggi, Christine Tuleau-Malot, and Nathalie Villa-Vialaneix. Random forests for big data. *arXiv preprint arXiv:1511.08327*, 2015.

[37] Vladimir Gorodetsky, Oleg Karsaeyv, and Vladimir Samoilov. Multi-agent technology for distributed data mining and classification. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 438–441. IEEE, 2003.

[38] Y Guo and J Sutiwaraphun. Knowledge probing in distributed data mining. In *Working Notes of the KDD-97 Workshop on Distributed Data Mining*, pages 61–69, 1998.

[39] David Heckerman, Dan Geiger, and David M Chickering. Learning bayesian networks: The combination of knowledge and statistical data. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 293–301. Morgan Kaufmann Publishers Inc., 1994.

[40] Jomi F Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.

[41] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM, 2001.

[42] Hillol Kargupta, Daryl Hershberger Byung-Hoon, and Erik Johnson. Collective data mining: A new perspective toward distributed data analysis. In *Advances in Distributed and Parallel Knowledge Discovery*. Citeseer, 1999.

[43] Matthias Klusch, Stefano Lodi, and Gianluca Moro. Agent-based distributed data mining: The kdec scheme. In *Intelligent information agents*, pages 104–122. Springer, 2003.

[44] Matthias Klusch, Stefano Lodi, and Gianluca Moro. Issues of agent-based distributed data mining. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1034–1035. ACM, 2003.

[45] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International joint Conference on artificial intelligence*, volume 14, pages 1137–1145. Lawrence Erlbaum Associates Ltd, 1995.

[46] Steff Lewis and Mike Clarke. Forest plots: trying to see the wood and the trees. *BMJ: British Medical Journal*, 322(7300):1479, 2001.

[47] M. Lichman. UCI machine learning repository, 2013.

[48] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramırez, Héctor-Gabriel Acosta-Mesa, and Francisco Grimaldo. A windowing based gpu optimized strategy for the induction of decision trees in jaca-ddm. In *Artificial Intelligence Research and Development: Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence*, volume 277, page 100. IOS Press, 2015.

[49] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, Héctor-Gabriel Acosta-Mesa, and Francisco Grimaldo. A windowing strategy for distributed data mining optimized through gpus. *Pattern Recognition Letters*, 2016.

[50] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, and Francisco Grimaldo. An agents & artifacts approach to distributed data mining. In F. Castro, Alexander Gelbukh, and M. G Mendoza, editors, *11th MICAI*, volume 8266 of *LNAI*, pages 338–349, Berlin Heidelberg, 2013. Springer Verlag.

[51] John Wylie Lloyd. *Foundations of logic programming*, volume 2. Springer-verlag Berlin, 1984.

[52] Win-Tsung Lo, Yue-Shan Chang, Ruey-Kai Sheu, Chun-Chieh Chiu, and Shyan-Ming Yuan. Cudt: a cuda based decision tree algorithm. *The Scientific World Journal*, 2014, 2014.

[53] P. Luo, Q. He, R. Huang, F. Lin, and Z. Shi. Execution engine of meta-learning system for kdd in multi-agent environment. In *AIS-ADM*, volume 3505 of *LNAI*, pages 149–160, Berlin Heidelberg, 2005. Springer-Verlag.

[54] Wenjing Ma and Gagan Agrawal. A translation system for enabling data mining applications on gpus. In *Proceedings of the 23rd international conference on Supercomputing*, pages 400–409. ACM, 2009.

[55] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety.* Springer Science & Business Media, 2012.

[56] Chayapol Moemeng, Vladimir Gorodetsky, Ziye Zuo, Yong Yang, and Chengqi Zhang. Agent-based distributed data mining: A survey. In *Data mining and multi-agent integration*, pages 47–58. Springer, 2009.

[57] Chayapol Moemeng, Xinhua Zhu, and Longbing Cao. Integrating workflow into agent-based distributed data mining systems. In *Agents and Data Mining Interaction*, pages 4–15. Springer, 2010.

[58] Chayapol Moemeng, Xinhua Zhu, Longbing Cao, and Chen Jiahang. i-analyst: An agent-based distributed data mining platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1404–1406. IEEE, 2010.

[59] Álvaro F Moreira, Renata Vieira, Rafael H Bordini, et al. Extending the operational semantics of a bdi agent-oriented programming language for introducing speech-act based communication. *Lecture notes in computer science*, pages 135–154, 2004.

[60] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and Information Systems*, pages 1–35, 2014.

[61] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[62] James J Odell, H Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer, 2002.

[63] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[64] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.

[65] CA Oyeka. An introduction to applied statistical methods. *Enugu: Nobern Avocation Publishing Co*, pages 218–84, 1996.

[66] Ikewelugo Cyprian Anaene Oyeka and Godday Uwawunkonye Ebuh. Modified wilcoxon signed-rank test. *Open Journal of Statistics*, 2(02):172, 2012.

[67] Byung-Hoon Park and Hillol Kargupta. Distributed data mining: Algorithms, systems, and applications. pages 341–358, 2002.

[68] REECHA B Prajapati and SUMITRA Menaria. Multi agent-based distributed data mining. *Int. J. Adv. Res. Comput. Eng. Technol.(IJARCET)*, 1(10):76, 2012.

[69] A. Prodromidis, P. Chan, and S. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed and parallel knowledge discovery*, 3, 2000.

[70] John Ross Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993.

[71] Adrian E Raftery, David Madigan, and Jennifer A Hoeting. Bayesian model averaging for linear regression models. *Journal of the American Statistical Association*, 92(437):179–191, 1997.

[72] A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

[73] V. S. Rao. Multi agent-based distributed data mining: An overview. *International Journal of Reviews in Computing*, 3:83–92, 2009.

[74] A. Ricci, M. Viroli, and A. Omicini. Construenda est cartago: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574, 2006.

[75] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[76] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[77] Stuart Russell. Artificial intelligence: A modern approach author: Stuart russell, peter norvig, publisher: Prentice hall pa. 2009.

[78] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012.

[79] John R Searle. Meaning and speech acts. *The Philosophical Review*, 71(4):423–432, 1962.

[80] J. Secretan. *An Architecture for High-Performance Privacy-Preserving and Distributed Data Mining*. PhD thesis, University of Central Florida Orlando, Florida, Orlando, FL., USA, 2009.

[81] Toby Sharp. Implementing decision trees and forests on a gpu. In *Computer Vision–ECCV 2008*, pages 595–608. Springer, 2008.

[82] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[83] Jerzy Stefanowski. Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data. In *Emerging paradigms in machine learning*, pages 277–306. Springer, 2013.

[84] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. Jam: Java agents for meta-learning over distributed databases. In *KDD*, volume 97, pages 74–81, 1997.

[85] Andreas L Symeonidis, Kyriakos C Chatzidimitriou, Ioannis N Athanasiadis, and Pericles A Mitkas. Data mining for agent reasoning: A synergy for training intelligent agents. *Engineering Applications of Artificial Intelligence*, 20(8):1097–1111, 2007.

[86] Nam-Luc Tran, Quentin Dugauthier, and Sabri Skhiri. A distributed data mining framework accelerated with graphics processing units. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 366–372. IEEE, 2013.

[87] Xiang-Yang Wang, Xian-Jin Zhang, Hong-Ying Yang, and Juan Bu. A pixel-based color image segmentation using support vector machine and fuzzy c-means. *Neural Networks*, 33:148–159, 2012.

[88] Merrill Warkentin, Vijayan Sugumaran, and Robert Sainsbury. The role of intelligent agents and data mining in electronic partnership management. *Expert Systems with Applications*, 39(18):13277–13288, 2012.

[89] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.

[90] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[91] Ian H. Witten and Eibe Frank. *Data mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, CA., USA, second edition, 2005.

[92] Marcin Wojnarski, Sebastian Stawicki, and Piotr Wojnarowski. TunedIT.org: System for automated evaluation of algorithms in repeatable experiments. In *Rough Sets and Current Trends in Computing (RSCTC)*, volume 6086 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 20–29. Springer, 2010.

[93] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[94] Michael Wooldridge, Nicholas R Jennings, et al. Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152, 1995.

[95] Junyi Xu, Li Yao, Le Li, and Yifan Chen. Sampling based multi-agent joint learning for association rule mining. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1469–1470. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[96] Lei Xu and Michael I Jordan. Em learning on a generalized finite mixture model for combining multiple classifiers. In *Proceedings of the World Congress on Neural Networks*, volume 4, pages 227–230, 1993.

[97] L. Zeng, L. Li, L. Duan, K. Lu, Z. Shi, M. Wang, W. Wu, and P. Luo. Distributed data mining: a survey. *Information Technology and Management*, 13(4):403–409, 2012.

[98] Ning Zhong, Yasuaki Matsui, Tomohiro Okuno, and Chunnian Liu. Framework of a multi-agent kdd system. In *Intelligent Data Engineering and Automated Learning—IDEAL 2002*, pages 337–346. Springer, 2002.

Part IV.

Appendix: accepted papers

An Agents & Artifacts approach to Distributed Data Mining

Xavier Limón[1], Alejandro Guerra-Hernández[1], Nicandro Cruz-Ramírez[1], Francisco Grimaldo[2]

[1] Universidad Veracruzana, Departamento de Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000
`xavier120@hotmail.com, aguerra@uv.mx, ncruz@uv.mx`
[2] Universitat de València, Departament d'Informàtica, Avigunda de la Universitat, s/n, Burjassot-València, España 46100
`francisco.grimaldo@uv.es`

Abstract. This paper proposes a novel Distributed Data Mining (DDM) approach based on the Agents and Artifacts paradigm, as implemented in CArtAgO [9], where artifacts encapsulate data mining tools, inherited from Weka, that agents can use while engaged in collaborative, distributed learning processes. Target hypothesis are currently constrained to decision trees built with J48, but the approach is flexible enough to allow different kinds of learning models. The twofold contribution of this work includes: i) JaCA-DDM: an extensible tool implemented in the agent oriented programming language Jason [2] and CArtAgO [10, 9] to experiment DDM agent-based approaches on different, well known training sets. And ii) A collaborative protocol where an agent builds an initial decision tree, and then enhances this initial hypothesis using instances from other agents that are not covered yet (counter examples); reducing in this way the number of instances communicated, while preserving accuracy when compared to full centralized approaches.

Keywords: Multi-Agent System, Distributed Data Mining, CArtAgO, Jason, Collaborative Learning

1 Introduction

As the amount of data produced by the everyday systems grows and distribute, the problems faced by Data Mining also grows. Being this the case, Data Mining as a research field needs to keep the pace. Distributed Data Mining (DDM) addresses the problem of mining huge amounts of distributed (even geographically) data. From the point of view of software engineering, DDM systems need to exhibit various desirable characteristics, such as scalability, configuration flexibility and reusability [7]. Multi-Agent Systems (MAS) are inherently decentralized and also distributed systems, being a good option to implement DDM systems that cope with the requirements. Nowadays, agent-based DDM is growing in popularity [14].

In this work we present JaCA-DDM, a novel approach of DDM based on the Agents and Artifacts paradigm, as implemented in CArtAgO [9]. Agents in the system are implemented in the well known agent oriented programming language Jason [2]. In JaCA-DDM, CArtAgO artifacts play a big role, being the basis, as will be explained later, for a modular, scalable, distributed Java based architecture, easy to design, implement and maintain. We also present a distributed learning strategy that borrows ideas from collaborative concept learning in MAS [3]. This strategy is an incremental collaborative protocol that tries to enhance a model created with few instances, by means of contradictory instances provided by the agents on the system. In this way, it is possible to reduce the number of instances communicated, while at the same time maintaining the accuracy of a centralized approach.

This paper presents a work in progress aimed to develop an agents & artifacts competitive approach for DDM. To this end, we created an experimental setting using the agent-artifact architecture discussed here, and we designed a series of experiments with the aim to test the differences in accuracy and actual training examples used between our strategy and a traditional centralized strategy. Accuracy and number of examples being our main concern, we put aside many efficiency aspects for the moment, but the main strategy and system architecture is open enough to allow further efficiency enhancement in the future.

This paper is organized as follows. Section 2 introduces the background for this paper, this includes: DDM, agent based DDM, and CArtAgO environments. Section 3 introduces JaCA-DDM and describes the generalities of our leaning strategy. In section 4 the experimental setting and results obtained are addressed. Finally, section 5 closes with a conclusion and future work.

2 Background

Knowledge Discovery in Databases, or data mining, is a discipline that merges a wide variety of techniques intended for the exploration and analysis of huge amounts of data, with the goal of finding patterns and rules somewhat hidden in it [13]. Since data mining is about data, it is important to know the origin and distribution of this data in order to exploit it efficiently. A traditional way of doing data mining is using a centralized schema, in this way, all the data and learned models are on the same site. With the ubiquitous presence of computational networks, it is common to find data belonging to the same system spreaded in various sites, even in sites that are geographically far away from each other. From the data mining point of view, some questions may arise in these distributed scenarios: Which is the best strategy for constructing learning models that take into account the data from all the sites?, What is the best way to face heterogeneous databases?, How can the communication of the data and the data mining process be optimized?, How can the privacy of the data be preserved?, Is there some efficient way to treat cases where the data changes and grows constantly?

A lot of systems devoted to DDM have been created. According to the strategy that they implement, those systems can be classified into two major categories [7]: centralized learning strategies, and meta-learning strategies. In the centralized strategies, all the data is moved to a central site, and when all data is merged together, data mining algorithms are applied sequentially to produce a single learning model. In general, the centralized strategy is expensive and inapplicable in a lot of cases because of the cost of data transmission. Meta-Learning refers to a general strategy that seeks to learn how to combine a number of separate learning processes in an intelligent fashion [4]. The idea behind Meta-Learning is to collect locally generated classification models and from there generate or simulate a single learning model. To accomplish this, it is necessary to collect the prediction of the local classification models on a validation data set, and then create a meta-level training data set from the predictions of the locally generated classification models. To generate the final meta-level classification model from the meta-level training data set, voting, arbitrating and combining techniques can be used [6]. Meta-learning is an efficient and scalable way to do DDM since the data transmitted is limited and the process is parallel, with a good load balance. Nevertheless, it is not as efficient as its centralized counterpart when a new instance needs to be classified. This is because the classification process is not completely direct. The classification query has to traverse a decision process that maybe has various classification models involved. Centralized learning is also simpler, to setup a meta-learning system can be more difficult. Another disadvantage of distributed meta-learning is that, because classifiers are not built globally on data, the model's performance may suffer as a result of incomplete information [11].

The learning strategy that we propose in the present is another approach for DDM. This strategy is inspired by SMILE[3]. SMILE is a collaborative setting for concept learning in MAS. A concept learning problem deals with keeping consistent a hypothesis about a target concept. The hypothesis has to be consistent with respect to a set of examples that can be received from the environment or from other agents. The hypothesis is kept consistent through a series of incremental revisions. In this way, the hypothesis is incrementally enhanced through a process that involves the use of counter examples (examples not covered by the current hypothesis). We took this idea and translate it to DDM terms, having instead of a hypothesis a learning model that is incrementally enhanced through the use of contradictory instances.

Data mining is applied to a variety of domains that have their own particularities and special cases, making it difficult to come with a general way of treating all scenarios. Multi-agent systems are a straight and flexible way to implement DDM systems since they are already decentralized distributed systems. In this way, the agents are in charge of the details of the problem. Each agent can do various tasks concurrently and it can be seen as an independent process. The location of the agents is in some degree irrelevant and transparent. The communication between agents is done in a high abstract level, that makes it easy to implement sophisticated protocols and behaviors. Some agent-based

DDM systems had been done in the past with successful results, JAM [12] is one of the most influential. JAM is a distributed, scalable and portable agent-based data mining system that employs Meta-Learning. For more information about agent-based DDM, [8] discusses the challenges of agent based DDM, and [14] is a good survey of DDM that includes agent-based DDM.

A fundamental part of a MAS is the environment where it is deployed. It is important to adequately model the environment such that the agents can be able to perceive it, modify it, and exploit it. CArtAgO is an infrastructure and architecture based on artifacts used for modeling computational environments in multi-agent systems. With CArtAgO the concept of environment is simplified, the environment is a set or artifacts.

An artifact is a first order abstraction used for modeling computational environments in MAS. Each artifact represents an entity of the environment, offering services and data structures to the agents in order for them to improve their activities, especially the social ones. Artifacts are also of great value in the design and reutilization of multi-agent systems since their structure is modular, based on object-oriented concepts. Artifacts are conceived as function-oriented computational devices, functionality that the agents may exploit in order to fulfill their individual and social activities. The vision proposed by CArtAgO impacts directly in the agent theories about interaction and activity. Under this vision a MAS is conceived as a set of agents that develop their activities in three distinct ways: computing, interacting with other agents and using shared artifacts that compose the computational environment.

Artifacts can be the objective of the agent activity as well as the tool the agents use to fulfill their activities, reducing the complexity of their tasks. Since the environment is composed by artifacts, the state of each artifact can be perceived by the interested agents. The infrastructure of CArtAgO was designed having in mind distributed environments. It is possible to define work-spaces to determine the context where an artifact exists and can be perceived and used. The distributed environment is transparent for the agent, the later is one of the most valuable characteristics of CArtAgO. In this work, CArtAgO plays an important role, and is one of the base technologies used to support JaCA-DDM.

3 JaCA-DDM: A Jason Multi-Agent System for DDM

`JaCA-DDM` (Jason & CArtAgO for DDM) is a Multi-Agent System implemented in Jason and situated in a CArtAgO environment. JaCA-DDM is used to create and run distributed learning experiments that are based on the collaborative learning strategy explained later. Currently it supports J48 decision trees, but it can easily be extended to support other classification learning approaches. The artifacts provided by this environment encapsulate data mining tools as implemented by WEKA[5]. In what follows, the artifacts, agents, and the workflow are described in detail.

The MAS is composed by a coordinator agent and n workers. There are three main types of artifacts used by the agents: `Oracle`, `InstancesBase` and

ClassifierJ48. The coordinator uses the `Oracle` to extract information about the learning set and split it among the workers and itself. Each agent stores its training examples in an `InstancesBase` artifact. Instances distribution is shown in figure 2 (page 7). The coordinator induces an initial model with its instances using `ClassifierJ48`. Then it asks for contradictory instances as shown in figure 3 (page 8). The interactions amongst the artifacts are shown in figure 1. In what follows, a more detailed account for each artifact is presented.

Since the main goal of `JaCA-DDM` is to experiment distributed learning scenarios, we are interested in partitioning existing training data sets in a controlled way to enable comparisons with centralized scenarios. The single `Oracle` artifact creates random stratified data partitions and distributes them among the agents. An agent can use the `Oracle` artifact to: obtain the attributes information, as described in the ARFF file; restart the artifact for a new running of the system; recreate the artifact to run a new round in the cross-validation process; get the number of instances stored in the artifact; and reinitialize the artifact with a new training set. The `port1` is used to get other artifacts linked with this one. Usually `InstancesBase` artifacts are linked via this port to get set of instances.

`ClassifierJ48` is a single artifact in charge managing and creating the learning model. An agent can execute a set of operations (○) on an instance of this artifact to: add a new training instance to the artifact; build a J48 classifier with the instances stored in the artifact; print the tree representation of the computed classifier; classify an instance; and restart the artifact for running a new experiment. An agent can also link other artifacts to this one, so that the linked artifacts can execute linked operations (◇) on the `Classifier48` for: getting the J48 classifier; and classifying an instance. Observe that the artifact is used to classify instances in two ways: i) An agent executes `classifyInstance` over a string representing the instance to be classified, obtaining an integer representing the instance class as defined in WEKA; ii) Another artifact executes `classifyInstance` to classify an instance stored in that artifact. The `port1` is used to link other artifacts linked to this one. Usually `InstancesBase` artifacts are linked via this port to classify instances.

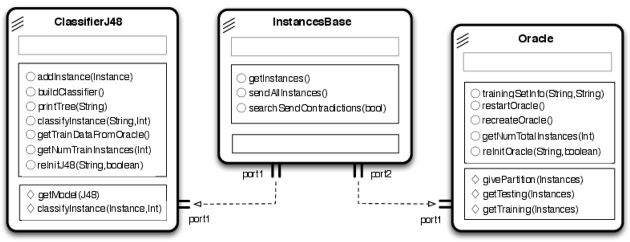


Fig. 1. The main artifacts used in JaCA-DDM

`InstancesBase` is an artifact class implementing local repositories of instances for the agents, so each agent has control of a InstancesBase artifact. Such an artifact can be linked with an `Oracle` artifact, via `port2`, in order to execute the linked operation `givePartition` to obtain a set of instances. It can be also linked to a `ClassifierJ48`, via `port1`, in order to search for a contradiction in the local repository and add it to the `ClassifierJ48`. A contradiction is an instance wrongly classified by the current model.

Other artifact related to the experimental setting provided by `JaCA-DDM` include: the `GUI` artifact is a front end for launching experiments and setting the different parameters for the experiment; the `Evaluator` artifact performs statistical operations with the results gathered, this operations include standard deviation, medias and paired T-test.

The collaborative learning strategy proposed has the following characteristics: there exists a central site, in this site the data is controlled by a special agent known as the coordinator. In this central site a base model is induced using the instances of the site, this base model serves as the first model that presumably needs enhancement since it maybe was induced with few instances. The base model can be shared between the different sites. The coordinator agent also is in charge of the experiment control, initialization of artifacts, control of the learning process, and managing the results. In each of the other sites, a worker agent resides, this agent manages the data of its corresponding site and runs a process with the purpose of finding contradictions between the base model and the instances of the site. A contradiction exists when the model does not predict the instance class correctly. The contradictory instances are sent to the central site enhancing the base model in a posterior induction. The process repeats itself until no contradictions are found.

To run the experiments we used a single computer to simulate different distributed scenarios consisting of various sites, the number of sites is configurable. Despite using a single computer for the experiments, the system architecture is flexible, it can also be applied in a true distributed environment without any major change.

Before an experiment begins, the parameters for the experiment are set through the GUI, those parameters include: database path, number of worker agents (in this manner simulating various sites) and type of model evaluation (hold-out or cross validation with its respective parameters). An experiment has the following general workflow: first, the coordinator determines which agents are going to participate in the experiment (currently all the agents participate). Then the coordinator creates the artifacts needed passing the relevant parameters. From there, each agent sends a request to `Oracle`, asking for its data partition. The coordinator sends all its examples to ClassifierJ48 in order to create the base model. Next, the coordinator begins the social process, asking to each worker, one by one, if it has contradictory examples. If a worker finds a contradiction, the contradictory example is sent to ClassifierJ48. When a worker finishes sending all the contradictions, the coordinator may issue an induction request to ClassifierJ48, the frequency of this induction request can be tuned in order to increase efficiency. This process continues until no more contradictions are found.

The interaction diagrams presented in figures 2 and 3 (page 8) resume the most important parts of the workflow described earlier. These figures omit the InstancesBase artifacts to simplify the diagrams and improve readability. Remembering that each agent has an InstancesBase associated for the storage and administration of its instances. In figure 2 it's shown the process of data distribution, and in figure 3 shows the learning process.

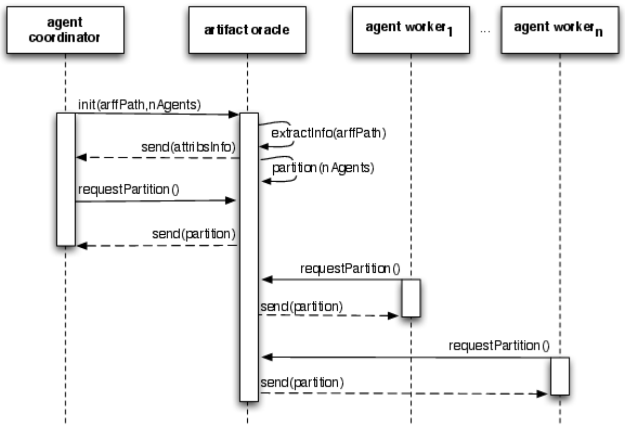


Fig. 2. Interaction diagram, data distribution

Our current learning strategy is linear in the sense that only one worker agent at a time searches and sends contradictions. In this aspect, we are not yet exploiting the concurrent and parallel facilities that the architecture of JaCa-DDM provides. This is likely to change in future revisions as we enhance our learning strategy.

4 Experiments and Results

JaCa-DDM was used to create a series of experiments to compare our collaborative learning strategy and a traditional centralized strategy. We choose a comparison against the centralized strategy because it offers a good benchmark for comparing accuracy and number of training examples used for training the

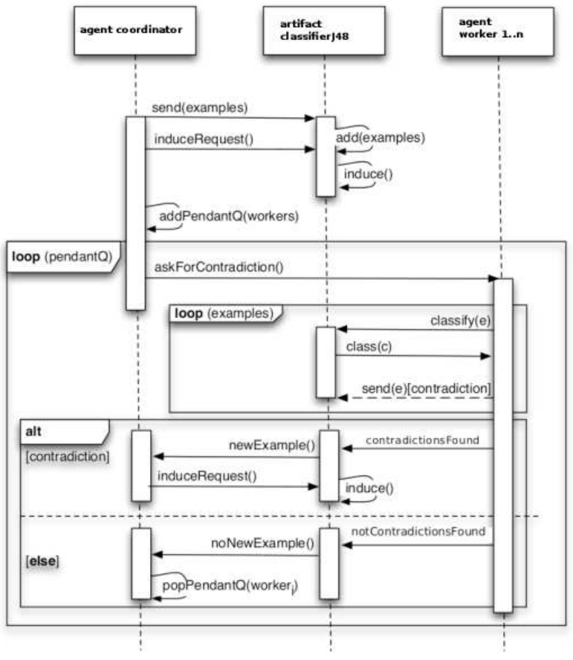


Fig. 3. Interaction diagram, learning strategy

learning model. This comparison takes into account the number of examples used for training, classification accuracy and time. Since we ran the experiments in a single computer and not in a distributed system, the time results may not be fair because the cost of data transmission is not present, nevertheless, for the sake of completeness we also show time results. A set of databases of the UCI repository [1] was used (table 1). To distribute the data between the agents a randomized stratified policy was used, the stratification ensures that each data partition conserves the ratio of class distribution. Stratified cross validation with 10 folds and 10 repetitions was applied. For each database, experiments were done with 1, 10, 30, 50 and 100 worker agents. To do the comparison, the same data partitions were used for both strategies. Results of two tailed paired T test with 0.05 degrees of significance are presented to test if there are significant differences between the two strategies, the results of this test are presented in a versus fashion, confronting the collaborative model against the centralized one (CollvsCen column of thable 3) and the collaborative model against the base model (CollvsBas column of table 3) . Where 0 means no significant difference (a tie), 1 means that the first strategy paired won against the second one, and -1 means that the first strategy lost against the second one. We used J48 algorithm with pruning activated and the rest of the WEKA options set to default.

Table 1. Data Sets

Data Set	Instances	Attributes	Classes
adult	48842	15	2
german	1000	21	2
letter	20000	17	26
poker	829201	11	10
waveform	5000	41	3

In table 2 the number of examples used in the centralized model, the base model (the first model induced with the instances of the central site), and collaborative model are presented. In the case of the collaborative model, the standard deviation is also shown, this is because there are variations in each experiment (the standard deviation is for 100 experiments). This table shows that our strategy definitely reduces the number of training examples used to induce the model. This can be seen for example in the results for the adult database (except for 1 worker agent) where only about 35% of the instances where used for training in our strategy. The standard deviation results show that our strategy is stable enough.

Table 3 shows the accuracy for the centralized model, base model, and collaborative model. This table also shows the results for paired T-test. There are results of standard deviation in the accuracy because each experiment may vary since the data is randomized each time. This table shows that our approach maintains a similar accuracy when compared with the centralized strategy. Even in the cases where, according to the paired T-test, our approach loses against the centralized strategy, the difference is not big (for example the accuracy results for the poker database), there are significant differences in those cases because the standard deviation of the centralized strategy is small.

Finally, table 4 shows the mean time in milliseconds of model creation for the central model and for the collaborative model. The time of the collaborative strategy includes the time for the base model. From this results it is obvious that our strategy has its process overhead, this is more noticeable in small databases like german, nevertheless, as the database grows, the advantages of our strategy begin to show up, this is specially true for the poker database, where the time efficiency actually improves. This boost in the time efficiency occurs because as the data grows it is more efficient to do multiple inductions with a small amount of data than doing a single induction with a big amount of data. Since our strategy pretends to be applied in scenarios where the amount of data is big, the results are promising.

As we continue to develop our work, we hope to do a more in depth analysis about the results and consequences of our collaborative learning strategy. In this case we limited our analysis to the most noticeable facts.

Table 2. Number of instances used to learn

Data Set	**Wks**	**Total**	**Centralized**	**Base**	**Collab**
adult	1	48842	43957	21978	27468.85 ± 107.53
adult	10	48842	43957	3996	16121.10 ± 147.73
adult	30	48842	43957	1417	15162.07 ± 142.62
adult	50	48842	43957	861	15403.52 ± 163.40
adult	100	48842	43957	435	16063.00 ± 221.61
german	1	1000	900	450	698.20 ± 15.68
german	10	1000	900	81	613.64 ± 13.94
german	30	1000	900	29	614.03 ± 16.20
german	50	1000	900	17	618.74 ± 15.94
german	100	1000	900	8	629.59 ± 16.10
letter	1	20000	18000	9000	11803.68 ± 164.30
letter	10	20000	18000	1636	8349.14 ± 217.90
letter	30	20000	18000	580	8259.17 ± 238.86
letter	50	20000	18000	352	8389.38 ± 227.43
letter	100	20000	18000	178	8628.10 ± 284.24
poker	1	829201	746280	373140	374100.00 ± 14.24
poker	10	829201	746280	67843	71419.50 ± 150.61
poker	30	829201	746280	24073	38988.50 ± 1750.09
poker	50	829201	746280	14632	48773.50 ± 994.89
poker	100	829201	746280	7388	81041.50 ± 1141.97
waveform	1	5000	4500	2250	3836.38 ± 33.40
waveform	10	5000	4500	409	3534.60 ± 34.54
waveform	30	5000	4500	145	3523.18 ± 34.12
waveform	50	5000	4500	88	3543.13 ± 34.50
waveform	100	5000	4500	44	3561.68 ± 37.20

5 Conclusions and Future Work

In this paper we presented JaCa-DDM, an extensible tool that we created and used to run a series of experiments of DDM. The principles entailed by JaCa-DDM make it easy to extend and improve it. This is due to the modular nature of the system and the fact that agents and artifacts raise the level of abstraction, so we can think naturally in terms of shared services, communication and workflow.

As the results in the previous section show, our learning strategy is promissory. Our initial expectation of reducing the number of training instances used to

Table 3. Accuracy results

Data Set	**Wks**	**Centralized**	**Base**	**Collab**	**CollvsCen**	**CollvsBas**
adult	1	86.00 ± 0.44	85.78 ± 0.48	86.32 ± 0.45	1	1
adult	10	85.97 ± 0.44	84.75 ± 0.57	86.22 ± 0.56	1	1
adult	30	85.99 ± 0.43	83.84 ± 0.73	86.25 ± 0.57	1	1
adult	50	86.02 ± 0.44	83.54 ± 0.89	86.28 ± 0.51	1	1
adult	100	85.98 ± 0.43	82.20 ± 1.58	86.30 ± 0.52	1	1
german	1	72.05 ± 3.73	71.33 ± 4.05	71.82 ± 4.02	0	0
german	10	71.57 ± 3.74	68.38 ± 3.81	71.73 ± 3.78	0	1
german	30	71.83 ± 4.11	68.14 ± 3.89	71.18 ± 4.00	0	1
german	50	71.75 ± 4.0	66.56 ± 5.56	71.51 ± 3.96	0	1
german	100	72.50 ± 3.73	65.36 ± 7.94	71.79 ± 4.09	-1	1
letter	1	87.98 ± 0.76	83.74 ± 0.87	88.18 ± 0.74	1	1
letter	10	88.07 ± 0.70	69.28 ± 1.35	88.26 ± 0.70	1	1
letter	30	87.96 ± 0.80	57.86 ± 1.69	88.23 ± 0.84	1	1
letter	50	88.09 ± 0.73	51.26 ± 2.23	88.26 ± 0.80	1	1
letter	100	88.02 ± 0.67	40.35 ± 3.11	88.26 ± 0.76	1	1
poker	1	99.78 ± 0.01	99.76 ± 0.010	99.79 ± 0.01	0	0
poker	10	99.78 ± 0.01	99.06 ± 0.11	99.74 ± 0.01	-1	0
poker	30	99.79 ± 0.01	96.47 ± 0.25	99.76 ± 0.01	-1	0
poker	50	99.79 ± 0.01	92.22 ± 1.36	99.33 ± 0.02	-1	0
poker	100	99.79 ± 0.01	87.99 ± 0.40	98.99 ± 0.79	-1	1
waveform	1	75.35 ± 1.87	74.77 ± 2.03	75.24 ± 1.75	0	1
waveform	10	75.36 ± 1.99	70.89 ± 2.22	75.08 ± 1.88	0	1
waveform	30	75.35 ± 1.90	67.52 ± 3.03	74.69 ± 2.09	-1	1
waveform	50	75.05 ± 1.74	65.44 ± 3.26	74.85 ± 1.94	0	1
waveform	100	75.21 ± 1.99	62.76 ± 4.54	74.99 ± 2.04	0	1

train the model while conserving the accuracy of a traditional centralized strategy was fulfilled. Now we have the challenge to improve the learning strategy to enhance efficiency as well as to do a more in depth analysis of the benefits and consequences of this approach. This analysis has to take into account more databases with a wide range of characteristics as well as more classification techniques, and not only J48. As it was mentioned earlier, we ran the experiments in a single computer, simulating various distributed sites. In the future, we hope to do experiments in a true distributed setting. In this way, we can have a better account of time results that will help us to move forward in the efficiency enhancements that we want to implement.

References

1. K. Bache and M. Lichman. UCI machine learning repository, 2013.
2. Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley-Interscience, 2007.
3. Gauvain Bourgne, Amal El Fallah Segrouchni, and Henry Soldano. SMILE: Sound multi-agent incremental learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 38. ACM, 2007.

Table 4. Processing time in milliseconds

Data Set	Wks	Centralized	Collab	Data Set	Wks	Centralized	Collab
adult	1	1393.97	5913.78	letter	1	795.76	5435.10
adult	10	1419.80	14191.26	letter	10	816.49	17850.55
adult	30	1435.85	15167.84	letter	30	813.13	18120.53
adult	50	1441.34	12626.48	letter	50	826.68	14723.98
adult	100	1465.65	9720.67	letter	100	848.36	11643.36
german	1	10.14	68.16	poker	1	143236.00	180256.00
german	10	7.70	264.52	poker	10	147610.00	120582.00
german	30	6.70	385.76	poker	30	145595.00	53229.00
german	50	6.73	402.97	poker	50	148476.00	54364.50
german	100	6.89	546.88	poker	100	147646.00	54837.00
waveform	1	372.84	3330.27				
waveform	10	370.02	9056.13				
waveform	30	377.05	9371.32				
waveform	50	390.79	6669.84				
waveform	100	399.83	6933.90				

4. Philip K Chan and Salvatore J Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8(1):5–28, 1997.
5. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
6. Andreas Prodromidis, Philip Chan, and Salvatore Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed and parallel knowledge discovery*, 3, 2000.
7. Vuda Sreenivasa Rao. Multi agent-based distributed data mining: An overview. *International Journal of Reviews in Computing*, 3:83–92, 2009.
8. Vuda Sreenivasa Rao, S Vidyavathi, and G Ramaswamy. Distributed data mining and agent mining interaction and integration: A novel approach, 2010.
9. Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.
10. Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Construenda est CArtAgO: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574, 2006.
11. Jimmy Secretan. *An Architecture for High-Performance Privacy-Preserving and Distributed Data Mining.* PhD thesis, University of Central Florida Orlando, Florida, 2009.
12. Salvatore Stolfo, Andreas L Prodromidis, Shelley Tselepis, Wenke Lee, Dave W Fan, and Philip K Chan. Jam: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.
13. Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.
14. Li Zeng, Ling Li, Lian Duan, Kevin Lu, Zhongzhi Shi, Maoguang Wang, Wenjuan Wu, and Ping Luo. Distributed data mining: a survey. *Information Technology and Management*, 13(4):403–409, 2012.

A Windowing based GPU optimized strategy for the induction of Decision Trees in JaCa-DDM

Xavier Limón [a], Alejandro Guerra-Hernández [a], Nicandro Cruz-Ramírez [a], Héctor-Gabriel Acosta-Mesa [a], and Francisco Grimaldo [b]

[a] *Universidad Veracruzana, Centro de Investigación en Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000*

[b] *Universitat de València, Departament d'Informàtica, Avigunda de la Universitat, s/n, Burjassot-València, España 46100*

Abstract. When inducing Decision Trees, Windowing consists in selecting a random subset of the available training instances (the window) to induce a tree, and then enhance it by adding counter examples, i.e., instances not covered by the tree, to the window for inducing a new tree. The process iterates until all instances are well classified or no accuracy is gained. In favorable domains, the technique is known to speed up the induction process, and to enhance the accuracy of the induced tree; while reducing the number of training instances used. In this paper, a Windowing based strategy exploiting an optimized search of counter examples through the use of GPUs is introduced to cope with Distributed Data Mining (DDM) scenarios. The strategy is defined and implemented in JaCa-DDM, a novel system founded on the Agents & Artifacts paradigm. Our approach is well suited for DDM problems generating large amounts of training instances. Some experiments in diverse domains compare our strategy with the traditional centralized approach, including an exploratory case study on pixel-based segmentation for the detection of precancerous cervical lesions on colposcopic images.

Keywords. Windowing, Decision Trees, GPU computation, Multi-Agent Systems, Distributed Data Mining

1. Introduction

The Windowing technique was originally designed to cope with memory limitations in the C4.5 [8] system. It consists in inducing a tree from a small random subset of the available training instances (the window). The tree is then used to classify the remaining training instances, searching for counter examples, i.e., instances not covered by the current tree. The window is extended with the counter examples found and a new tree is induced. The process iterates until a stop criterion is met, e.g., all examples are covered; or the accuracy of the new tree does not enhance anymore.

Windowing is expected to obtain an accuracy similar to that obtained using all the available training instances, while reducing considerably the number of examples used to induce a tree. In favorable domains, i.e., free of noise and indeterminism, it is also expected to speed up the inductive process; but in the general case, it slows down the process since convergence requires many iterations. Windowing based strategies [4] for Distributed Data Mining (DDM) seems to inherit these properties: The accuracy of the induced trees is close to, or even slightly better than that obtained without windowing; The number of examples used to induce the tree is reduce up to 60%; But the processing time is much worse when using Windowing, 90 times slower in the worst case.

Searching for counter examples seems to be in part responsible for the poor time performance of the Windowing based strategies. In this work, CUDA [7] enabled GPUs are used to improve the gathering of counter example, seeking a performance improvement of the overall induction processes. Although some frameworks have been proposed to boost time efficiency of data mining process through GPUs [12,6], including the induction of Decision Trees [11,5], our work focuses on DDM scenarios, using JaCa-DDM [4] to further enhance the performance of the processes and to overcome GPU memory limitations.

JaCa-DDM is an Agents & Artifacts [9] based DDM system, conceived to design, implement, deploy and evaluate distributed learning strategies. A strategy is a description of the interactions among a set of agents, exploiting artifacts deployed in a distributed system, that provide data mining tools. The proposed strategy concerns the induction of Decision Trees [8], using the J48 algorithm provided by Weka [14].

The organization of the paper is as follows: Section 2 presents a brief description of JaCa-DDM, introducing the notions of strategy and deployment system. Section 3 describes the implementation of the Windowing based strategy proposed in this paper, detailing the GPU based optimizations. Section 4 defines the experimental methodology to evaluate the proposed strategy. The results and discussion of the experiments are presented in section 5. Finally, this paper closes with some conclusions and insights of future work in section 6.

2. JaCa-DDM

JaCa-DDM is a system based on the Agents & Artifacts paradigm as implemented by Jason [3], the well known agent oriented programming language, and CArtAgO [9], an agent infrastructure to define environments based on the concept of artifacts. The main interest of JaCa-DDM is to provide a platform to execute and test data mining processes over a distributed environment. A novelty of JaCa-DDM is the way in which the DDM processes are conceived as strategies.

Strategies are descriptions of workflows in terms of agents and artifacts interactions, allowing the implementation of truly sophisticated processes that can exploit BDI reasoning and representations, as well as speech acts based communications; while using already existing data mining tools provided by Weka, wrapped in the form of artifacts. Strategies are by definition encapsulated, allowing standarized ways to define, configure, deploy, and test them.

The JaCa-DDM model is built on the concepts of strategy and its deployment. While a strategy defines a DDM workflow, its deployment deals with configuration issues.

Definition 2.1 *A tuple $\langle Ags, Arts, Params, ag_1\rangle$ denotes a JaCa-DDM strategy, where:*

- *$Ags = \{ag_1, \dots, ag_n\}$ is the set of user defined Jason agent programs.*
- *$Arts = \{art_1, \dots, art_m\}$ is the set of user defined artifact types.*
- *$Params = \{param_1 : type_1, \dots, param_k : type_k,\}$ is a set of paramenters and their associated data types, where $type_{1,\dots,k} \in \{int, bool, double, string\}$.*
- *$ag_1 \in Ags$ is a special agent program playing the role of contact person between the agents in Ags and the deployment system. This agent launches and finishes the strategy, and can be programmed to do any other task.*

Definition 2.2 *A tuple $\langle Nodes, DS, Arts, Strat, Config, ag_0\rangle$ is a JaCa-DDM deployment system, where:*

- *$Nodes = \{node_0, node_1 \dots, node_j\}$, is a set of computational nodes, usually, but not necessarily, distributed in a network, where: $node_0$ is running Jason and CArtAgO, while $node_{1,\dots,j}$ are running only CArtAgO. Each node defines a single CArtAgO workspace, where artifacts are to be created, but all agents run in $node_0$. Each node is denoted by a pair $\langle nodeName, IPaddress : port\rangle$.*
- *$DS = \{ds_1, \dots, ds_j\}$ is a set of data sources associated to each node, excepting $node_0$. Data sources can be created dinamically at run time; or be statically defined in each node.*
- *$Arts = \{art_1, \dots, art_i\}$ is a set of artifact types, used to deploy the system.*
- *Strat is a learning strategy as stated in Definition 2.1.*
- *$Config = \langle \delta, \pi\rangle$ is a configuration for a strategy deployment. It has two components:*

  ∗ *$\delta = \{(ag, node, i), \dots\}$, is a set of allocations, i.e., an agent distribution specifying how many copies of a given agent program will be focusing on a given node. Where $ag \in Strat_{Ags}$ is an agent program in the strategy, that will be cloned $i \geq 1$ times, and assigned to focus on $node \in Nodes \backslash \{node_0\}$.*

  ∗ *$\pi = \{(p, v), \dots\}$ is a set of pairs: strategy parameter, initialization value; where for all $param : type \in Strat_{Params}$, p is a parameter of the strategy and v is its value of type t.*

- *ag_0 is the agent setting up the deployment system.*

3. Implementation

The implementation of the proposed Windowing based GPU optimized strategy comprehends a GPU based counter examples filtering processes, and the Parallel Counter GPU strategy itself.

3.1. GPU based counter examples filtering processes

The windowing process can be split into two main subprocess repeated iteratively: counter examples filtering and model induction. In this work we implement the filtering of counter examples on GPUs, trying to achieve a negligible time cost for this subprocess. As mentioned, we do not deal with the induction subprocess, which allows for other methods to be applied, for example ensemble techniques, or GPU based induction algorithms.

A decision tree is a directed acyclic graph, where each node has at most one father. There are two kinds of nodes: internal nodes, and leaf nodes. Internal nodes represent a given attribute, and leaf nodes class values. Arcs contain a boolean function over the node attribute values, and each function is mutually exclusive. There are three kinds of arc functions, each one bound to a boolean operator: $\leq$, $>$, $=$. The first two operators are for numerical attributes, and the last one for nominal ones. Given a Decision Tree, and an unclassified instance, a classification process consist on traversing arcs yielding true values on its function, from the root node to a leaf node.

On GPUs, it is a good practice to avoid irregular and complex data structures, in order to improve performance. Scattered memory access is not efficient, and affects the performance of the GPU cache memories. It is better to read large blocks of memory in order to exploit coalesced memory access (combining multiple memory accesses into a single transaction). With these ideas in mind, a plain representation based on one dimensional arrays was adopted for the GPU Decision Trees. The structure consists on various properties, some of them are related to node and arc information:

- int NUM_NODES : how many nodes (including leaves) has the tree.
- int MAX_NUM_ARCS: each node can have a variable number of arcs, but a constant value is necessary to reserve memory.
- int attributes[NUM_NODES]: contains the attribute index for each node. On the case of a leaf node, it contains the index of the class value.
- int isLeaf[NUM_NODES]: a leaf node contains the value 1, otherwise 0.
- int numbersOfArcs [NUM_NODES]: the actual number of arcs of each node.
- int NUM_ARCS: the sum of all the arcs of all nodes.
- int evalTypes[NUM_ARCS]: the evaluation type of the arc: $\leq$, $>$, $=$.
- float vals[NUM_ARCS]: evaluation value of the arc.
- int nodeIndices[NUM_ARCS]: the index of the node pointed by the arc.

A method that takes a Weka J48 Tree and transforms it to a GPU Decision Tree was implemented in the J48 artifact. A kernel is a function that executes on a device (GPU). Kernels are intended to be executed in parallel, receiving parameters to define the number of threads to be used. The implemented kernels include:

- classify : Return the index value of the predicted class of an instance.
- searchCounter: Classifies each instance within the instances set in GPU, and if the predicted class is different from the actual class, then it saves the index of the instance in an array as big as the instances set. Each thread

receives the number of instances that will process. At the end of its work, each thread also saves the number of counter examples found.

- countSize: Computes a sum over each thread result of the searchCounter kernel to yield the total number of counter examples found.
- genResult: "Shrinks" the array that contains the counter examples indices found by the searchCounter kernel, saving the indices on a new array that is the exact size of the counter examples found.
- filterParallel: Filters the counter examples from the dataset in the GPU.

The workflow of the counter examples filtering process requires to load the instances set into the GPU at the beginning of the general process. A copy of the instances set is held in the CPU. It is also necessary to determine the number of multi-processors, and the maximum number of parallel threads of each multi-processor, in order to define an ideal number of working threads. The filtering process, from the host's (CPU) point of view, can be summarized in the following steps:

1. Transform the J48 Tree into a GPU Decision Tree.
2. Load the GPU Decision Tree in the GPU.
3. Invoke the searchCounter kernel on the ideal number of threads.
4. Invoke the countSize kernel on one thread.
5. Use the result from countSize to reserve enough memory on the GPU to save all the counter example indices on an array.
6. Invoke genResult on one thread to fill the array created previously.
7. Invoke filterParallel on the ideal number of threads to erase the counter examples found from the instances set in the GPU.
8. Use the array with counter examples indices, and filter all the counter examples in the CPU to obtain a counter examples instance set.
9. Free the memory not needed anymore on the GPU.

Note that the search process on the GPU only finds index values, the actual filtering is done on the CPU. This design choice was made to reduce data transmission between the host and the devices.

3.2. Parallel Counter GPU strategy

The proposed strategy consists on an agent bulding an initial Decision Tree with a subset of its training instances, in a central classifier artifact. Asking to other agents in the system to gather all the counter examples found in the deployment system, and sending them to the central classifier artifact for trying to enhance the Decision Tree. The processes iterates for a number of rounds. Following definition 2.1, its components are:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - ∗ *contactPerson* controls the rounds and induces the learned model.
  - ∗ *worker* gathers counter examples.
  - ∗ *roundController* determines the stop condition.
- $Arts = \{ClassifierJ48, InstancesBase, Evaluator\}$, where:

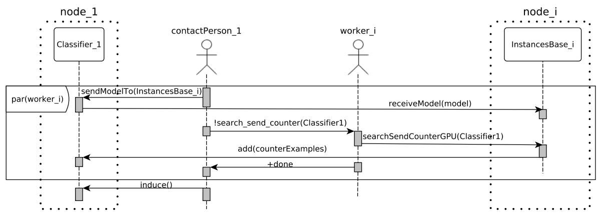


Figure 1. Parallel Counter GPU strategy sequence diagram for counter examples filtering workflow. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i). $contacPerson_1$ sends the current model to the $InstancesBase_i$ artifact of each $worker_i$. Then it asks all the workers to search for counter examples in their $InstancesBase_i$ using the GPU, and send them to $Classifier_1$, where a new model is induced.

  - ∗ $ClassifierJ48$. Induces models.
  - ∗ $InstancesBase$. Used to store and manipulate the learning examples. The GPU search is launched on this artifact.
  - ∗ $Evaluator$. Used to compute the accuracy of a model given a testing set, for the auto-adjust stop procedure.

- $Params$ include:

  - ∗ $Prunning : Bool$ if true, forces the J48 to use post pruning.
  - ∗ $InitPercentage : Double$ defines the size of the initial training window.
  - ∗ $TestPercentageForRounds : Double$ defines the size of the testing set for the auto-adjust stop procedure.
  - ∗ $ChangeStep : Double$ defines a threshold of minimum change between two consecutive rounds. Used by the auto-adjusted stop procedure.
  - ∗ $MaxRounds : Int$ defines the maximum number of rounds.

Figure 1 shows the workflow for one round of the Parallel Counter GPU strategy. The stop criterion computing is not show for the sake of clarity, but at the end of every round, the induced Decision Tree is tested to obtain its accuracy and decide if the process continues or not.

4. Experimental Methodology

A set of datasets were selected to compare the perfomance of the proposed Parallel Counter GPU strategy with the usual centralized approach. The measured parameters for all the experiments are the following: accuracy, percentage of training examples used, time in seconds, number of leaves, and tree size. The experiments were executed on a cluster consisting of three computers with the same characteristics: Two Xeon processors at 2.40 GHz with four cores, and two threads each; 24 GB of RAM; Two GPU Nvidia Tesla C2050.

Table 1 shows some datasets used for this purpose. They were selected from the MOA [2] and TunedIT [15] projects, because they vary in the number of instances, attributes, and classes. Evaluation was done with a 10-fold stratified cross validation, and also the training instances were stratified and split evenly among the 3 computer nodes.

Table 1. Used MOA/TunedIT datasets.

DS	#Instances	#Attributes	#Classes
airlines	539383	8	2
covtypeNorm	581012	55	7
KDDCup99	4898431	42	23
poker-lsn	829201	11	10

Strategy distribution δ was $\{(contactPerson, node_1, 1), (roundController, node_1, 1), (worker, node_1, 1), (worker, node_2, 1), (worker, node_3, 1)\}$. The parameters initialization π for all datasets was $\{(Pruning, true), (InitPercentage, 0.15), (TestPercentageForRounds, 0.15), (ChangeStep, 0.004), (MaxRounds, 10)\}$. The InitPercentage and TestPercentageForRounds parameters only take data from one node, and each node has one third of the training data, thus the parameter value considers this fact. On the centralized case, pruning was also active.

An exploratory case study for pixel-based segmentation was also considered. Pixel-based segmentation consists on extracting features from labeled pixels to create a training dataset, which is used to construct a model to predict the class of unseen pixels, and in this way achieve image segmentation [13]. Our case study deals with sequences of colposcopic images presenting possible precancerous cervical lesions. The image data was extracted from 38 patients, for each patient a range between 300 and 600 images were obtained. A medical expert labeled some of the pixels of each image (figure 2 shows an example), from which two classes can be drawn: precancerous lesion, and no precancerous lesion. For a more detailed account of this case study see [1]. From the images for each patient, we selected 30 images evenly spread in the series. Using FIJI [10] we extracted the pixels of interest from the images to create a Weka ARFF file, selecting the default pixel parameters: gaussian blur, sobel filter, hessian, membrane projections, and difference of gaussians. The obtained dataset has the following characteristics: Total number of examples: 1016600; No precancerous lesions: 213819; Precancerous lesions: 802781; Number of attributes: 80. We compare our strategy with the centralized approach using leave-one-out, where the test data in each case is the extracted pixels from a single patient. For our strategy, the training data was stratified and evenly split in our 3 computers available. The distribution δ and parameters π of the Parallel Counter GPU was the same as in the general experiments. It is worth mentioning that we did not apply any preprossessing to the dataset, being an exploratory experiment, we are more interested in testing the behavior of our approach in this kind of setting.

5. Results and discussion

Table 2 shows the results obtained by our strategy and the centralized approach for the MOA/TunedIT datasets. As expected, accuracy is similar in all cases, and

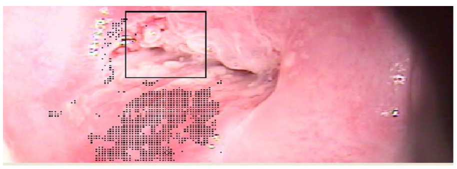


Figure 2. Example of colposcopic image. Black dots represent pixels labeled by a medical expert.

our strategy reduced the number of examples used for training up to a 90%. The number of leaves and tree size are also reduced by our strategy in all cases. GPU based counter examples filtering speed up the whole process. For KDDCup99 and poker-lsn, our approach is up to 8 times faster than the centralized one. For airlines and covtypeNorm, in the wort case, our approach is 0.78 times slower than the centralized. This enhances considerably the results of the Windowing based strategies previously reported by Limón [4], where such strategies were up to 200 times slower than the centralized approach.

The rate of instances used to induce a Decision Tree is indicative of how difficult is for the strategy to converge. Higher rates suggest that the strategy iterates more times, inducing more trees, and using more instances before attaining convergence. Interestingly, in these and previous experiments, the rate of used instances seems to correlate with the accuracy: Low accuracy demands more instances, while high accuracies demands much fewer instances. In these cases, the GPU based counter example fintering is not responsible of the decreased time performance, but the number of iterations executed by the strategy. The airlines dataset shows an extreme case in this regard, the overall problem is too difficult for the J48 algorithm.

In the observation of the evolution of the learning process of covtypeNorm we found that the majority of examples used for learning were found during the fist search of counter examples. This means that the initial model was too simple, and adding all the counter examples found could not be the best choice in the long run. Possibly, increasing the size of the initial training set, and/or further filtering counter examples on initial phases of the process, would help to obtain a better time performance.

Table 2. Results for the MOA/TunedIT datasets.

DS	Strategy	Accuracy	Used instances	Time (seconds)	#Leaves	Tree Size
airlines	Centralized	66.34 ± 0.11	100.00 ± 0.00	1164.66 ± 211.76	137470	142081
airlines	Parallel Counter	66.26 ± 0.12	94.95 ± 0.01	1810.78 ± 446.47	132767	137210
covtypeNorm	Centralized	94.59 ± 0.04	100.00 ± 0.00	855.41 ± 97.88	14158	28314
covtypeNorm	Parallel Counter	93.10 ± 0.34	48.44 ± 0.01	1089.03 ± 277.06	12679	25265
KDDCup99	Centralized	99.99 ± 0.01	100.00 ± 0.00	1688.91 ± 363.89	968	1147
KDDCup99	Parallel Counter	99.96 ± 0.01	9.28 ± 0.01	199.72 ± 45.62	667	855
poker-lsn	Centralized	99.78 ± 0.01	100.00 ± 0.00	174.26 ± 28.55	2212	4408
poker-lsn	Parallel Counter	98.67 ± 0.46	9.56 ± 0.01	24.90 ± 8.05	1831	3552

For the pixel-based segmentation case study, Table 3 shows a comparison of the results obtained by our strategy, the centralized approach, and the results reported in [1]. The sensibility (*Sen*) is the rate of true positive observations (precancerous lesion) against the sum of true positive plus false negatives, and the specificity (*Spe*) is the rate of true negative (no precancerous lesion) observations against the sum of true negative plus false positives.

Observe that the results are similar to those obtained for the covtypenorm dataset, with similar explanations. Anyway, observe that the proposed approach obtained a similar accuracy and sensibility than the results reported by Acosta-Mesa et al. [1], where a time series approach was adopted, and other normalizations were applied, that yielded a more balanced dataset, which in turns explains the difference in specificity that we obtained. Our experiment was done with a unbalanced dataset with no preprocessing applied (as this is an exploratory case study), favoring the precancerous lesion class, which also may explain our improvement in sensibility. Being identified the probable cause of no time performance improvement in our strategy (i.e too many counter examples are added on the first round), and given the results obtained, we are optimistic that with a proper preprossessing of the dataset, and with an enhancement of our current strategy, we can achieve much better results.

Table 3. Results for the case study

Strategy	Accuracy	Used instances	Time (seconds)	#Leaves	Tree Size	Sen	Spe
Centralized	66.32 ± 29.50	100.00 ± 0.00	4806.50 ± 455.28	32436	64871	79.04	18.53
Parallel Counter	65.87 ± 26.89	46.80 ± 0.02	6317.36 ± 605.46	30633	61265	77.73	21.37
Results from [1]	67.00	n/a	n/a	n/a	n/a	71.00	59.00

6. Conclusions and future work

The Windowing based GPU optimized strategy proposed in this work demonstrates that Windowing based approaches can be applied to large datasets, having clear time performance improvements in some cases, in comparison with the centralized approach, while preserving a similar accuracy. Even on no favorable cases, our strategy was acceptably slower, and always reduced the number of leaves and tree size by using less counter examples. We believe that the time performance that our strategy can achieve is bound by two factors: i) The complexity of the problem represented by the dataset; and ii) The dataset redundancy. The first factor is an open problem, and for the time being, we do not plan to direct our efforts toward it, while the second factor is of interest for our future work. In practical terms, redundancy means that some examples from the dataset are not needed for the learning process, and that discarding such examples is paramount to improve time performance. We plan to so research on the nature of redundancy on tree model induction to create an improved version of the Windowing algorithm that further filters redundant examples.

References

[1] Héctor-Gabriel Acosta-Mesa, Nicandro Cruz-Ramírez, and Rodolfo Hernández-Jiménez. Aceto-white temporal pattern classification using k-nn to identify precancerous cervical lesion in colposcopic images. *Computers in biology and medicine*, 39(9):778–784, 2009.

[2] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[3] Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason.* John Wiley & Sons Ltd, 2007.

[4] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, and Francisco Grimaldo. An agents & artifacts approach to distributed data mining. In F. Castro, Alexander Gelbukh, and M. G Mendoza, editors, *11th MICAI*, volume 8266 of *LNAI*, pages 338–349, Berlin Heidelberg, 2013. Springer Verlag.

[5] Win-Tsung Lo, Yue-Shan Chang, Ruey-Kai Sheu, Chun-Chieh Chiu, and Shyan-Ming Yuan. Cudt: a cuda based decision tree algorithm. *The Scientific World Journal*, 2014, 2014.

[6] Wenjing Ma and Gagan Agrawal. A translation system for enabling data mining applications on gpus. In *Proceedings of the 23rd international conference on Supercomputing*, pages 400–409. ACM, 2009.

[7] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[8] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[9] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[10] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012.

[11] Toby Sharp. Implementing decision trees and forests on a gpu. In *Computer Vision–ECCV 2008*, pages 595–608. Springer, 2008.

[12] Nam-Luc Tran, Quentin Dugauthier, and Sabri Skhiri. A distributed data mining framework accelerated with graphics processing units. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 366–372. IEEE, 2013.

[13] Xiang-Yang Wang, Xian-Jin Zhang, Hong-Ying Yang, and Juan Bu. A pixel-based color image segmentation using support vector machine and fuzzy c-means. *Neural Networks*, 33:148–159, 2012.

[14] Ian H. Witten and Eibe Frank. *Data mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, CA., USA, second edition, 2005.

[15] Marcin Wojnarski, Sebastian Stawicki, and Piotr Wojnarowski. TunedIT.org: System for automated evaluation of algorithms in repeatable experiments. In *Rough Sets and Current Trends in Computing (RSCTC)*, volume 6086 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 20–29. Springer, 2010.

Pattern Recognition Letters
journal homepage: www.elsevier.com

A Windowing strategy for Distributed Data Mining optimized through GPUs

Xavier Limón[a,**], Alejandro Guerra-Hernández[a], Nicandro Cruz-Ramírez[a], Héctor-Gabriel Acosta-Mesa[a], Francisco Grimaldo[b]

[a]*Universidad Veracruzana, Centro de Investigación en Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000*
[b]*Universitat de València, Departament d'Informàtica, Avigunda de la Universitat, s/n, Burjassot-València, España 46100*

ABSTRACT

This paper introduces an optimized Windowing based strategy for inducing decision trees in Distributed Data Mining scenarios. Windowing consists in selecting a sample of the available training examples (the window) to induce a decision tree with an usual algorithm, e.g., J48; finding instances not covered by this tree (counter examples) in the remaining training examples, adding them to the window to induce a new tree; and repeating until a termination criterion is met. In this way, the number of training examples required to induce the tree is reduced considerably, while maintaining the expected accuracy levels; which is paid in terms of time performance. Our proposed enhancements solve this by searching for counter examples on GPUs and further reducing their number in the window. The resulting strategy is implemented in JaCa-DDM, our agents & artifacts tool for Distributed Data Mining, keeping the benefits of Windowing, while distributing the process and being faster than the traditional centralized approach, even performing similarly to Bagging and Random Forests in some cases. Experiments in data mining tasks are addressed, including a case study on pixel-based segmentation for the detection of precancerous cervical lesions on medical images.

1. Introduction

Distributed Data Mining (DDM) scenarios involve large amounts of data scattered in different sites, e.g., internet based databases and data streams for meteorology, oceanography, economy, etc; geographically distributed information systems; sensor networks; and grids. While traditional Data Mining approaches require collecting all data in a single site, this is usually inefficient or infeasible due to storage, communication, and computational costs, as well as privacy issues (Tsoumakas and Vlahavas, 2009). Agent Mining (Cao et al., 2012) has aroused as an option to cope with such scenarios, because of the distributed nature and inherent advantages of Multi-Agent Systems (MAS): autonomy, flexibility, robustness, and scalability (Moemeng et al., 2009). Numerous Agent Mining systems and framework have been proposed, including: JAM (Stolfo et al., 1997), BODHI (Kargupta et al., 1999), Papyrus (Bailey et al., 1999), GLS (Zhong et al., 2002), EMADS (Albashiri and Coenen, 2009), i-Analyst (Moemeng et al., 2010), SMAJL (Xu et al., 2014).

This work introduces an optimized Agent Mining method, based on Windowing (Quinlan, 1993), to induce decision trees. Windowing was proposed to cope with memory limitations when using the inductive algorithm C4.5. It consists in selecting a sample of the available training examples (the window) to induce a decision tree; finding instances not covered by this tree in the remaining training examples (counter examples); adding them to the window to induce a new tree; and then, repeating the process until a stop criterion is met. Windowing yields models with an accuracy similar to C4.5, using all the available training examples in a single site; but reducing significantly their number, since consistent examples are filtered out. However, given its iterative nature, Windowing did not show good time performance. Two enhancements are proposed to solve this: i) Optimizing the search for counter examples through the use of GPUs, and ii) Reducing the size of the window even further.

The proposed method, called Parallel Counter GPU Extra, is modeled and implemented as a strategy for JaCa-DDM, our own Agent Mining system founded on the Agents & Artifacts paradigm (Omicini et al., 2008). A strategy in this context, is a workflow described in terms of the interactions among agents using standard Data Mining tools, distributed in a computer network as artifacts. JaCa-DDM consolidates the Agent Mining

[**]Corresponding author: Tel.: +52-228-817-2957;
e-mail: xavier120@hotmail.com (Xavier Limón)

concept by adopting an Agent & Artifacts approach that provides the right level of abstraction and flexibility for modeling, implementing, and deploying DDM workflows, while reusing standard Data Mining tools for this.

Agent Mining apart, different enhancements of the inductive algorithms themselves have been proposed for learning decision trees from large and distributed datasets, including:

- Bagging (Breiman, 1996) is an ensemble technique, where a set of learning models of the same type are induced from separate training datasets for the same problem. These models are gathered at a single place, and used for classification, following a majority vote scheme. Bagging uses a bootstrap sampling, where a single dataset of size n is used to produce m datasets of size n' by applying random sampling with replacement, obtaining then m models. For large n, when $n = n'$, it is expected that each dataset keeps $\approx 63.2\%$ of unique training examples. Generally, the more training datasets, the better the results, since this reduces the variance of the method (Witten and Frank, 2005).

- Random Forest (Breiman, 2001) is a variant of Bagging, resulting in a set of decision trees, induced from bootstrap samples. Each tree induction is boosted by randomly sampling the attribute vector for each node split, based on a parameter K, determining the number of attributes taken into account at each node split. This makes possible to process datasets with a large numbers of attributes. Small K values are recommended, e.g., $log_2(|attributes|)+1$. Map-Reduce extensions (Genuer et al., 2015) apply a careful subsampling, in order to avoid sampling bias, but this is difficult in distributed scenarios where sites are naturally biased. Windowing may be useful as a sampling method, purposefully trying to skew the training example distribution, by considering only the counter examples found while learning (Fürnkranz, 1998). Since Windowing samples while the model is learned, no extra sampling step is needed.

- Incremental decision trees, such as VFDT (Domingos and Hulten, 2000), ICE (Yoon et al., 1999), and BOAT (Gehrke et al., 1999) are able to update the learned model in the presence of new training examples. Model updating is often very fast and memory usage is also optimized, since only a small sample of training examples needs to be loaded. Some of them, e.g., VFDT, are sensitive to the order of arrival of the training examples, since nodes can not be completely rearranged once created (Nguyen et al., 2014). They enable DDM strategies that exchange models instead of training examples.

- Inducing decision trees from distributed heterogeneous data (Chawla et al., 2003; Mehenni and Moussaoui, 2012), i.e., data exhibiting vertical partitions. Although the discussed strategies do not apply for vertical partitioned data, there is evidence (Melgoza-Gutiérrez et al., 2014) that JaCa-DDM can be used in such cases, when inducing decision trees.

- Related to GPUs, there are efforts to induce decision trees using GPUs (Lo et al., 2014; Sharp, 2008); and different frameworks that try to boost time efficiency of Data Mining process through GPUs have been proposed (Ma and Agrawal, 2009; Tran et al., 2013), but distributed settings have not been considered. Using JaCa-DDM further enhances the performance of the processes and helps to overcome GPU memory limitations.

In what follows, the adopted methods and tools are described in detail in Section 2. Two experimental settings are used to evaluate the proposed strategy, comparing it with the centralized use of the inductive algorithm, a previously proposed strategy (Parallel Counter GPU), and JaCa-DDM strategies based on VFDT, Bagging, and Random Forest. The first setting uses some datasets from well known repositories; the second one is a pattern recognition case study, based on pixel-based image segmentation for the identification of precancerous cervical lesions on colposcopy images. These settings are introduced in Section 3. Results are analyzed and discussed in Section 4. Conclusions are drawn in Section 5.

2. Methods and tools

Although the methods adopted in this work can be implemented in different ways, all of them have been actually conceived as strategies for JaCa-DDM (Limón et al., 2013, 2015). Our Agent Mining system is briefly introduced for those interested in exploring the discussed strategies, as included in the JaCa-DDM [1] distribution. Then, the Windowing method and the proposed enhancements are described in detail. Finally, the definition of the resulting Parallel Counter GPU Extra strategy is introduced.

2.1. JaCa-DDM

JaCa-DDM is an Agent Mining system founded on the Agents & Artifacts paradigm, conceived to design, implement, deploy, and evaluate DDM strategies. Jason (Bordini et al., 2007), a known Agent Oriented Programming language, is used to implement different strategies to cope with distributed computer environments in terms of rational agents and their interactions. CArtAgO (Ricci et al., 2011) artifacts encapsulate Weka (Witten and Frank, 2005) learning algorithms, data sources, evaluation tools, and other resources usually employed by the agents in such tasks.

Strategies make use of the Belief-Desire-Intention (BDI) reasoning and representation provided by Jason, as well as the agent communication based on speech-acts. Agents can create, perceive, and use the Weka artifacts in their environment; coordinating themselves in a workflow. A JaCa-DDM model, based on the concepts of strategy and deployment, is proposed to allow canonical ways to create and deploy different strategies. A tuple $\langle Ags, Arts, Params, ag_1 \rangle$ denotes the elements of a strategy, where:

- $Ags = \{ag_1, \ldots, ag_n\}$ is a set of user defined agent types.

[1] Available at `http://jacaddm.sourceforge.net`

- $Arts = \{art_1, \ldots, art_m\}$ is a set of user defined artifact types, used to support Data Mining related tasks.
- $Params = \{param_1 : type_1, \ldots, param_k : type_k, \}$ is a set of parameters and their associated data types, where $type_{1,\ldots,k} \in \{int, bool, double, string\}$.
- $ag_1 \in Ags$ is a special agent program playing the role of contact person between the agents in Ags and the deployment system. This agent launches and finishes the strategy, and can be programmed to do any other task.

Observe that these elements are required for all the strategies defined in JaCa-DDM. The specific workflow contained in a strategy, i.e., the way the agents learn together using their artifacts, is encapsulated in the agent programs. The workflow for the proposed strategy is described with the help of UML-like interaction diagrams in Section 2.3. A tuple $\langle Nodes, DS, Arts, Strat, Config, ag_0\rangle$ denotes a JaCa-DDM deployment specification, where:

- $Nodes = \{node_0, node_1 \ldots, node_j\}$, is a set of computational nodes, usually, but not necessarily, distributed in a network. Each node defines a single CArtAgO workspace, where artifacts are to be executed, but all agents run in $node_0$. Each node is identified by a pair $\langle node_{i=0,\ldots,j}, IPaddress : Port\rangle$.
- $DS = \{ds_1, \ldots, ds_j\}$ is a set of data sources associated to each node, not including $node_0$. Data sources can be created dinamically at run time; or be statically defined in each node.
- $Arts = \{art_1, \ldots, art_i\}$ is a set of artifact types, used to deploy the system.
- $Strat$ is a learning strategy as introduced before.
- $Config = \langle \delta, \pi\rangle$ is a configuration for a strategy deployment. It has two components:
  - $\delta = \{(ag, node, i), \ldots\}$, is a set of allocations, denoting that i copies of the agent $ag \in Strat_{Ags}$ will focus on a given $node \in Nodes$. Focusing here, means an agent can perceive and act on artifacts via Java RMI.
  - $\pi = \{(param, val), \ldots\}$ is a set of parameters initialization for all $param : type \in Strat_{Params}$, where val is a specific value of the given $type$.
- ag_0 is the agent setting up the deployment.

Given XML definitions of a strategy and its deployment specification, JaCa-DDM executes the associated workflow in the available computational infrastructure. The decision of running all the agents in $node_0$ optimizes communication costs, while the demanding computational processes in the workflow, as inducing models, classifying instances, searching for counter examples, etc., are truly distributed using the artifacts.

2.2. Enhancing the Windowing method

The Algorithm 2.1 describes the basic Windowing method. Although the stopping condition may vary, the traditional criterion is to stop when no more counter examples are found.

Algorithm 2.1. *The basic Windowing algorithm.*

```
1:  function WINDOWING(Exs)
2:      Window ← sample(Exs)
3:      Exs ← Exs − Window
4:      repeat
5:          stopCond ← true
6:          model ← induce(Window)
7:          for ex ∈ Exs do
8:              if classify(model, ex) ≠ class(ex) then
9:                  Window ← Window ∪ {ex}
10:                 Exs ← Exs − {ex}
11:                 stopCond ← false
12:     until stopCond
13:     return model
```

Windowing was criticized because the learned models were not only unable to significantly outperform the traditional centralized approaches, but an extra computational cost resulted of the search for counter examples. Nevertheless, the method can achieve significant run-time gains in noise-free domains (Fürnkranz, 1998) and reduces significantly the number of training examples used to induce the models.

The Algorithm 2.1, involves two main subprocesses repeated iteratively: the model induction (Line 6); and the search for counter examples (Lines 7–11). We have found that, when large amounts of data are involved, reducing the number of examples used in the induction can potentially boost time performance, even if the process is repeated iteratively; but, for this to happen, the searching for counter examples must also be accelerated using GPUs.

2.2.1. Enhancing the inductive process

There are two ways of improving the time performance of the inductive process: altering the inductive algorithm itself as discussed in the introduction, e.g., using parallel computing, incremental computing, GPU boosting, etc.; and keeping the size of the window as small as possible. The second approach is adopted here, while the first one is considered for future work.

The proposed enhancement exploits the fact that some counter examples seem redundant in the following sense: suppose a decision tree is computed with a given window and the remaining set of training examples are classified as shown in Fig. 1. In order to enhance such a tree, Windowing adds all the counter examples to the window to execute a new inductive process. This happens in all the leaves of the tree. Now, if two counter examples reach the same leaf when classified, they are alike in the sense that they were misclassified for similar reasons, i.e., their attributes values are similar.

Since smaller windows mean faster inductions, we hypothesize that it is unnecessary to add all the alike counter examples to the window at once, in order to obtain the desired accuracy levels, faster. Three parameters are proposed to control the

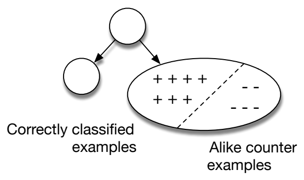


Fig. 1. Alike counter examples (-) are those that reached the same leaf when classified. Correctly classified examples (+) are not considered in the iteration, while some alike counter examples are added to the window in each iteration.

number of alike counter examples being added to the window in each iteration:

- μ defines the percentage of randomly selected counter examples per leaf to be added to the window;
- Δ defines a percentage increment of μ, applied at each iteration and;
- γ defines the minimun number of examples per leaf, required to apply any filtering at all.

With these parameters, the function to know how many counter examples are sampled for each tree node in any given iteration is the following:

$$keep(C,i) = \begin{cases} |C| & \text{If } |C| < \gamma \ \vee \ \mu + incr(\mu) \geq 1 \\ |C| \times (\mu + incr(\mu)) & \text{Otherwise} \end{cases}$$

Where: C is a set of counter examples in a given node; i the current iteration starting at 0; and $incr(\mu) = i \times \Delta$. On the first rounds of Windowing, the sets of alike counter examples tend to be big; as the model improves and more leaves are created, these sets become smaller. Given these observations, parameters are set to discard more counter examples at the beginning of the process, and discard less counter examples as Windowing progresses. The method is implemented in GPUs as a part of the parallel search of counter examples.

2.2.2. *Enhancing the searching for counter examples process*

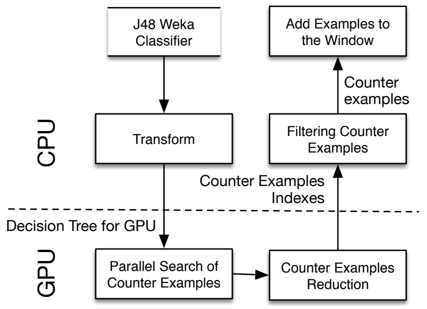
The searching for counter examples is accelerated using GPUs, in order to achieve a negligible time cost for this process. This enhancement requires representing the decision trees and the training examples in data structures well suited for CUDA (Nickolls et al., 2008), as well as implementing the corresponding classification and filtering algorithms as kernels.

Decision trees have two kinds of nodes: internal and leaf nodes. Internal nodes represent attributes and leaf nodes, class values. Arcs represent a boolean function over the attribute values, and each function over the same node is mutually exclusive. There are three kinds of arc functions, each of them bound to the boolean operators: $\leq$, $>$, $=$. The first two are for numerical attributes, and the last for nominal ones. Given a decision tree, and an unclassified instance, a classification process consist of traversing arcs yielding true values on its function, from the root to a leaf.

When using GPUs, it is good practice to avoid irregular and complex data structures. Scattered memory access is not efficient and affects the performance of the GPU cache memories. It is better to read large blocks of memory in order to exploit coalesced memory access, i.e., combining multiple memory accesses into a single transaction. With these ideas in mind, a plain representation based on one dimensional arrays was adopted. The structure consists on various arrays, related to node and arc information, that make possible to traverse the tree in order to do classifications in an efficient way.

A variety of CUDA kernels were implemented to support different tasks such as classification, counter examples filtering, counter examples reduction, etc.

Training examples are represented in the GPU as numeric arrays of size n, where each index $0, \ldots, n-1$ represents the value of the attribute with the same index. The last element of the array represents the class value. The searching for counter examples process requires to load the training examples into the GPU at the beginning of the Windowing process. A copy of them is held in the CPU. It is also necessary to determine the number of multi-processors, and the maximum number of parallel threads of each multi-processor, in order to define an ideal number of working threads. The filtering process is summarized in Fig. 2.



J48 Weka Classifier
Add Examples to the Window
Counter examples
CPU
Transform
Filtering Counter Examples
Counter Examples Indexes
Decision Tree for GPU
GPU
Parallel Search of Counter Examples
Counter Examples Reduction

Fig. 2. Counter examples searching executed at each Windowing iteration.

Note that the search process on the GPU only finds index values, the actual filtering is done on the CPU. This design choice was made to reduce data transmission between the CPU and the GPU, and thus improve performance for large datasets. The enhancements to the inductive process and the searching for counter examples are implemented as part of the Parallel Counter GPU Extra strategy, defined as follows.

2.3. *Parallel Counter GPU Extra strategy*

The *extra* in the proposed strategy is due to the control in the number of counter examples aggregated to the window, additional to the GPU search for counter examples previously proposed as Parallel Counter GPU (Limón et al., 2015). Following

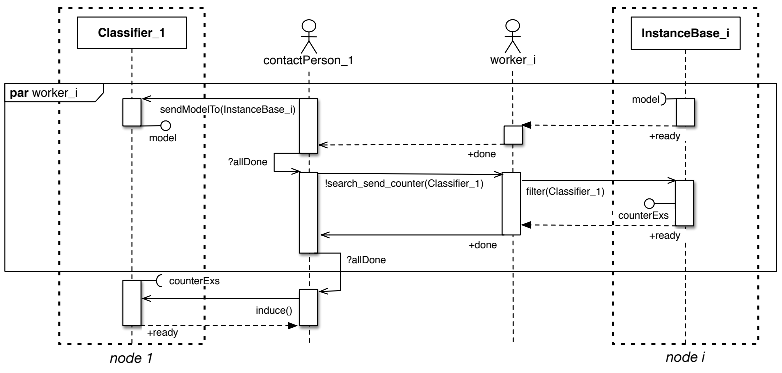


Fig. 3. Parallel Counter GPU Extra strategy sequence diagram for the counter examples filtering workflow.

the JaCa-DDM model, the proposed strategy has the following components:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the process and induces the learned model.
  - *worker* gathers counter examples in each distributed node.
  - *roundController* evaluates the termination criterion.
- $Arts = \{Classifier, InstancesBase, Evaluator\}$, where:
  - $Classifier$. It is a Weka J48 classifier, extended with GPU capabilities. It is used for inducing decision trees.
  - $InstancesBase$. It is a Weka instances base used to store and manipulate training examples. It is also extended with GPU capabilities.
  - $Evaluator$. It used to compute the accuracy of a given model, with a set of reserved training examples.
- *Params* include:
  - *Prunning* : *Bool* defines if post pruning is to be used.
  - *WindowInitPerc* : *Double* defines the size of the initial window, as a percentage of the available training examples.
  - $StopTestSetPerc$: *Double* defines the size of the validation set used for computing the termination criterion, as a percentage of the available training examples.
  - *AccuracyThr* : *Double* defines a threshold of minimum acceptable accuracy change between two consecutive rounds. Used by the termination criterion.
  - *MaxRounds* : *Int* defines the maximum number of rounds of the process. It subsumes the termination criterion.
  - μ : *Double* defines the initial percentage of counter examples to be collected per leaf, as described before.
  - Δ : *Double* defines an increment percentage for μ applied at each iteration, as described before.
  - γ : *Int* defines the minimum number of counter examples needed in a leaf to apply filtering, as describe before.

The resulting workflow for one iteration of the process is shown in Fig. 3, using a UML-like notation. Dotted boxes represent different nodes in the system. Regular boxes represent artifacts. An arrow from an agent to an artifact denotes an operation being executed by the agent on that artifact. An arrow from an agent to an agent denotes a speech act, e.g., achieve something (!) or tell something (+).

The agent *contactPerson*_1 builds a decision tree using $Classifier$_1 with a subset of the available training examples, i.e., the initial window, and sends the resulting model to the instances base artifacts of each worker. Once this is done, *contactPerson*_1 asks the workers to search for counter examples and send them to the $Classifier$_1 artifact. This searching process is GPU optimized as described before. Once the counter examples are collected, each worker agent sends them to the classifier artifact, in order to enhance the current decision tree with a new induction over the extended window. The process iterates until the termination criterion is met: determining, for a pair of rounds, if the obtained accuracy computed over a validation set, is better enough in the second round. For the sake of clarity, some parts of the process are not shown, including this halt condition, and the initial model induction.

3. Experiments

Two experimental settings were adopted for testing the applicability of the proposed strategy, one based on known datasets and another based on a pattern recognition case study: pixel-based segmentation of images for the detection of possible precancerous cervical lesions.

All the experiments evaluate and compare the accuracy of the obtained models, as well as the number of training examples used in the process, the run-time measured in seconds, the complexity of the models (number of leaves and tree size), and the resulting confusion matrix. All the experiments were executed on a cluster of three nodes with the same characteristics:

- Two Xeon processors at 2.40 GHz with four cores, and two threads each.
- 24 GB of RAM.
- One CUDA enabled Nvidia Tesla C2050 GPU with 448 cores and 6 GB of memory.

The Parallel Counter GPU Extra strategy is compared with other strategies with different purposes. It is compared with: Parallel Counter GPU (Limón et al., 2015) to evaluate the effects of the extra filtering of counter examples; Weka Centralized for comparison with the traditional use of the data mining tool; Centralizing VFDT for comparison with a full centralization approach based on incremental induction; and Bagging and Random Forest for comparison with meta-learning approaches. It is worth noting that all the strategies were implemented using JaCa-DDM.

Centralizing VFDT gathers all the training examples scattered in the distributed nodes; and then induce a decision tree using the VDFT algorithm, as implemented in MOA (Bifet et al., 2010). The Bagging strategy is based on J48 pruned models, as provided by Weka. In each distributed node, 16 models are created in parallel from bootstrap samples of the local data. The Random Forest strategy runs on the same basis as Bagging, using $log_2(|attributes|)+1$ as the K value for randomly selecting attributes in each node split.

3.1. Case study 1: known datasets

Four representative datasets from the UCI repository (Lichman, 2013), as adapted by the MOA (Bifet et al., 2010) and TunedIT (Wojnarski et al., 2010) projects, were selected:

- airlines. A dataset to predict flight delays from the information of the scheduled departure.
- covtypeNorm. A dataset containing the forest cover type for 30 x 30 meter cells obtained from US Forest Service.
- KDDCup99. A dataset to built a predictive model capable of distinguishing between bad connections, intrusions or attacks, and good normal connections.
- poker-lsn. A dataset containing examples of poker and no poker hands drawn from a standard deck of 52 cards.

The datasets properties are shown in Table 1. Parameters (π) and agent distribution (δ) are configured as shown in Table 2, for all the experiments. A 10-fold stratified cross validation is adopted, i.e., preserving class value ratios in each partition. The training examples were stratified and split evenly among the three nodes, in order to have a fair comparison among the various methods.

Table 1. The properties of the adopted datasets.

Dataset	#Instances	#Attributes	#Classes
airlines	539383	8	2
covtypeNorm	581012	55	7
KDDCup99	4898431	42	23
poker-lsn	829201	11	10

Table 2. Strategy configuration.

Parameters (π)	Agent distribution (δ)
Pruning = true	contactPerson, node_1, 1
WindowInitPerc = 0.15	roundController, node_1, 1
StopTestSetPerc = 0.15	worker, node_1, 1
AccuracyThr = 0.004	worker, node_2, 1
MaxRounds = 10	worker, node_3, 1
$\mu = 0.15$	
$\Delta = 0.15$	
$\gamma = 10$	

3.2. Case study 2: pixel-based segmentation

Computer Vision (CV) is a discipline which attempts to emulate the perceptual interpretation of images developed by the human eye and brain. CV implies the acquisition and analysis of digital images represented as 2D arrays which contain color-spatial features that create different complex patterns. In order to create high level representations from those patterns, machine learning techniques are used. One of the most important challenges in CV is the segmentation of images with classification purposes. Although CV has multiple potential applications, the computationally demanding nature of the state-of-the-art algorithms makes them difficult to apply and ask for more efficient data analysis schemes.

A pixel-based segmentation problem consists on extracting features from labeled pixels to create a training dataset, which in turn is used to construct a model to predict the class of unseen pixels, and in this way achieve image segmentation (Wang et al., 2012). As each training example is a pixel, the training set becomes restrictively big and parallel methods are necessary.

This case study deals with sequences of colposcopic images presenting possible precancerous cervical lesions (Acosta-Mesa et al., 2009). A colposcopy test consists in the visualization of the cervix through a low power microscope called colposcope. During visualization time, a solution of acetic acid is spread over the cervix surface in order to induce a coagulation process on those cells that are in transformation process due to the cancerous lesion. For the purpose of the case study reported here, a sequence of images was taken during visualization time, in order to capture the temporal color changes on the tissue.

The image sequences were extracted from 38 women patients, ages ranging from 22 to 35 years old. All of them gave informed written consent.

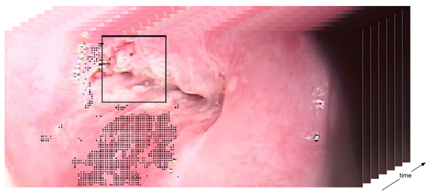


Fig. 4. Example of a sequence of colposcopic images. Black dots represent pixels labeled by a medical expert.

For each patient a range between 310 and 630 aligned images were obtained. The difference on the number of images per patient is due to the fact that some patients where recorded for a longer time than others. A medical expert labeled some of the pixels from the images of each patient (Fig. 4 shows an example). Taking into account that the images are aligned, a mask for each patient can be drawn, marking six classes: normal tissue, immature metaplasia, mature metaplasia, ectopy, low grade lesion, and high grade lesion. From these classes, only the last two represent possible precancerous lesion. We are only concerned in recognizing two classes: possible precancerous lesion (+) and the opposite (−), but the six classes are considered in order to exploit class decomposition to deal with class imbalance (Stefanowski, 2013).

As shown in Table 3, observations are imbalanced, having substantially more possible precancerous lesion observations (+). Classes are balanced by varying the number of images from the patients, depending on the number of observations of the class. Less observed classes needed more images from a patient with observations of that class. As the minimum number of images in the series is 310, this value is adopted as the maximum number of images that could be considered for a patient. The minimum number of observations of a class is 771, multiplying that value by 310 yields to 239010, which is the maximum number of instances per class that can be obtained in a balanced set. So the total number of instances is $239010 * 6 = 1434060$.

Table 3. The six classes of the case study 2 and the associated number of patients and observations for each one.

Class	#Patients	#Observations
Normal tissue (−)	11	4294
Immature metaplasia (−)	7	771
Mature metaplasia (−)	5	1264
Ectopy (−)	2	802
Low grade lesion (+)	26	23897
High grade lesion (+)	3	2908

Observe that this process still results in an imbalance of negative and positive classes. The resulting imbalance is intended as a mean to mitigate the problem of fewer patients presenting observations of the negative class, which affects the results when using a leave-one-out evaluation per patient, as the evaluation for patients with negative observations tend to draw a significant percentage of negative examples for testing, creating an important imbalance on the training data, favoring the positive class.

Using FIJI (Schindelin et al., 2012), 30 numeric attributes were extracted from the pixels of interest, by applying the following texture filters: Variance, Mean, Minimum, Maximum, and Median. Each filter used a maximum number of 16 neighbors (maximum sigma parameter in FIJI).

Results are evaluated using the leave-one-out method, in order to properly assess accuracy in this case. The test dataset in each case is extracted from a single patient, which means that 38 iterations are performed. For each tested strategy, with the exception of the Centralized one, the training data was stratified and evenly split in the available nodes. Parallel Counter GPU Extra configuration is shown in Table 2.

4. Results and discussion

Table 4 shows the results for the case study 1. A Wilcoxon signed rank test with a significant evidence at $\alpha = 0.05$ and three possible outcomes (Won, Lost, Tie), is adopted to compare the accuracy of the considered strategies. The accuracy of Parallel Counter GPU Extra is comparable with that of the other methods in all datasets, although being slightly lower for the airlines and covtypeNorm datasets.

Observe that, while preserving accuracy, our strategy reduces the number of examples used for training, up to a 95% (around 49% in the worst case). This results in an important improvement in speed terms, when compared with our previous work, Parallel Counter GPU. Such improvement is due exclusively to the extra filtering proposed, independently of the GPU optimization shared by both strategies. When compared with the rest of the strategies, our proposal is consistently faster than the Weka Centralized approach and even faster than Bagging and Random Forest in some cases. Centralizing VFDT, given its incremental nature, is by far the fastest of the considered strategies, but this is payed with a poor accuracy, even when all the available examples are used to induce the model.

Bagging always maintains a similar accuracy as the Weka Centralized approach, showing a consistent good time performance, which results of working with one third of the data in each distributed node. Bagging seems to overcome the possible bias imposed by data distribution, even if such bias is low in this case, as the data was stratified for distribution. It would be interesting to test this method with truly biased distributed datasets.

The decay of accuracy and time performance shown by Random Forest in some datasets, is likely due to the random selection of attributes at each splitting point, i.e., the most informative attributes could not be selected, tending to create bigger and less accurate trees. Also, being 48 the maximum number of trees in the forest enabled in our setting, there was not improvement in accuracy neither. Furthermore, the resulting bigger trees tended to be slower to induce that those obtained by Bagging.

Table 4. Results for the case study 1: known datasets. In Wilcoxon column the accuracy of each method is compared against Parallel Counter GPU Extra.

DS	Strategy	Accuracy	%Instances	Time (seconds)	Wilcoxon test
airlines	**Parallel Counter GPU Extra**	65.36 ± 0.25	51.21 ± 0.03	435.04 ± 106.28	–
airlines	Weka Centralized	66.34 ± 0.11	100.00 ± 0.00	1164.66 ± 211.76	Won
airlines	Parallel Counter GPU	66.26 ± 0.12	94.95 ± 0.01	1810.78 ± 446.47	Won
airlines	Centralizing VFDT	65.24 ± 0.27	100.00 ± 0.00	4.67 ± 0.53	Tie
airlines	Bagging	66.45 ± 0.13	100.00 ± 0.00	144.67 ± 4.47	Won
airlines	Random Forest	66.76 ± 0.11	100.00 ± 0.00	123.82 ± 3.89	Won
covtypeNorm	**Parallel Counter GPU Extra**	92.17 ± 0.52	43.28 ± 0.04	817.30 ± 253.27	–
covtypeNorm	Weka Centralized	94.59 ± 0.04	100.00 ± 0.00	855.41 ± 97.88	Won
covtypeNorm	Parallel Counter GPU	93.10 ± 0.34	48.44 ± 0.01	1089.03 ± 277.06	Won
covtypeNorm	Centralizing VFDT	76.83 ± 0.35	100.00 ± 0.00	8.96 ± 0.56	Lost
covtypeNorm	Bagging	94.99 ± 0.10	100.00 ± 0.00	149.35 ± 5.37	Won
covtypeNorm	Random Forest	78.34 ± 0.39	100.00 ± 0.00	44.47 ± 4.38	Lost
KDDCup99	**Parallel Counter GPU Extra**	99.98 ± 0.01	4.29 ± 0.01	93.23 ± 6.671	–
KDDCup99	Weka Centralized	99.99 ± 0.01	100.00 ± 0.00	1688.91 ± 363.89	Tie
KDDCup99	Parallel Counter GPU	99.96 ± 0.01	9.28 ± 0.01	199.72 ± 45.62	Lost
KDDCup99	Centralizing VFDT	99.97 ± 0.01	100.00 ± 0.00	56.17 ± 1.307	Lost
KDDCup99	Bagging	99.99 ± 0.01	100.00 ± 0.00	371.51 ± 19.39	Tie
KDDCup99	Random Forest	99.97 ± 0.01	100.00 ± 0.00	132.43 ± 21.23	Lost
poker-lsn	**Parallel Counter GPU Extra**	99.53 ± 0.59	8.80 ± 0.01	22.93 ± 3.51	–
poker-lsn	Weka Centralized	99.78 ± 0.10	100.00 ± 0.00	174.26 ± 28.55	Tie
poker-lsn	Parallel Counter GPU	98.67 ± 0.46	9.56 ± 0.01	24.90 ± 8.05	Lost
poker-lsn	Centralizing VFDT	87.78 ± 1.92	100.00 ± 0.00	4.25 ± 0.47	Lost
poker-lsn	Bagging	99.71 ± 0.10	100.00 ± 0.00	64.09 ± 5.49	Tie
poker-lsn	Random Forest	96.73 ± 0.25	100.00 ± 0.00	236.34 ± 14.37	Lost

Due to space limitations, the detailed results about number of leaves, tree size, and confusion matrix are omitted. The first two of them are an indicator of the complexity of the induced trees, but no significant differences were found, with the exception of Random Forest, as explained above; and VFDT that always creates much smaller trees. No significant differences were found for the confusion matrixes, neither. We expected to improve accuracy for the minority classes in the KDDCup99 dataset, but a more refined sampling method, such as the one described in (D'Addabbo and Maglietta, 2015), seems to be required.

Observe that the proposed Windowing based method could be used in conjunction with assembly techniques, such as Bagging and Random Forest, i.e., building forests of trees induced using Parallel Counter GPU Extra, which is consistently faster than the Weka Centralized approach traditionally adopted.

Table 5 shows the results for the case study 2. The original case study results, reported in (Acosta-Mesa et al., 2009), are also included in comparisons. That work uses a time series approach based on the intensity value of each labeled pixel over time. Each training example represents a spatial positioned signal, i.e., a spatio-temporal pattern, that it is smoothed using a polynomial model. A k-Nearest neighbor approach, with $k = 20$ and Euclidean distance as similarity function, is used for classification. The leave-one-out method is also adopted for evaluation. On the contrary, Parallel Counter GPU is not included, given that the new extra strategy already showed to be consistently faster, while preserving accuracy. Even though, results for that strategy, using a different preprocessing setting, are reported in (Limón et al., 2015).

As usual, sensibility (Sen) is the rate of true positive observations (precancerous lesion) against the sum of true positive plus false negatives, and the specificity (Spe) is the rate of true negative (no precancerous lesion) observations against the sum of true negative plus false positives.

When compared to the other considered strategies, Parallel Counter GPU Extra obtains significantly the highest accuracy, based on the Wilcoxon signed rank test computed as before. This may be a consequence of a better integration of examples from the negative classes, thus improving specificity. These examples, although numerous, thanks to the class balance, do not appear in patients as frequently as examples from the positive class, as shown in Table 3. This in turn affects the accuracy evaluation in a leave-one-out setting, favoring the positive class, but possibly the Windowing technique helped in this case. It is also interesting to note how our method, while not directly comparable, approaches the results from (Acosta-Mesa et al., 2009), having the same accuracy, but differing on sensibility and specificity, which it is expected since other preprocessing and techniques were applied.

The wide standard deviation of the accuracy results is an indicator of how different is the accuracy for each patient. For some patients the accuracy is over 90% while for others is about 15%, this happens because of the nature of the data, which it is a com-

Table 5. Results for case study 2 (n/a = not available). In Wilcoxon column each method is compared against Parallel Counter GPU Extra, a – means that the comparison is not possible.

Strategy	Accuracy	%Instances	Time (seconds)	Wilcoxon test	Sen	Spe
Parallel Counter GPU Extra	67.61 ± 19.32	37.00 ± 3.52	3782.26 ± 1094.21	–	60.96	64.83
Weka Centralized	63.68 ± 18.44	100.00 ± 0.00	6436.64 ± 923.16	Lost	60.80	61.60
Centralizing VFDT	53.34 ± 20.58	100.00 ± 0.00	32.03 ± 2.61	Lost	53.10	58.51
Bagging	64.25 ± 21.78	100.00 ± 0.00	1138.83 ± 108.83	Lost	65.40	59.16
Random Forest	58.88 ± 23.71	100.00 ± 0.00	1817.10 ± 179.18	Lost	68.78	49.34
Original results (Acosta-Mesa et al., 2009)	67.00 ± n/a	n/a	n/a n/a	–	71.00	59.00

mon place in medical applications. For some patients there are few observations, and also some classes happen in few patients, being a problem when adopting a leave-one-out approach for testing.

Our method is considerably faster than the Centralized approach, while slower than Bagging and Random Forest, but with significant better accuracy. Centralized VFDT stands out as the faster strategy, but it also has the worst accuracy.

5. Conclusions and future work

Parallel Counter GPU Extra strategy showed to be well suited to DDM scenarios, involving large amounts of data scattered in different sites. The proposed enhancements overcome the time performance issues associated to Windowing based strategies, while achieving similar accuracy results to the centralized approach. The strategy seems also very promissory for pixel-based segmentation problems, when combined with a careful image preprocessing. Distributing the data in such cases, improves time performance and reduces memory loads, when comparing to centralized approaches, keeping also a decent overall performance when compared to meta-learning methods, e.g., Bagging and Random Forest.

Our experimental settings produce stratified partitions of the original datasets, distributed in the available JaCa-DDM nodes. Testing the considered strategies when this is not the case would be of interest to evaluate their performance when facing local overrepresented classes. Parallel Counter GPU Extra is expected to degrade gracefully in such situations, but the exact impact of such bias need to be established in future experiments.

Parallel Counter GPU Extra performs a centralized induction exploiting a distributed search for training examples. While this allows for a single model with a global scope of the data, it also creates a potential bottleneck. An alternative approach would be to distribute the inductive process as well. Indeed, this is the way adopted by meta-learning methods (Chan and Stolfo, 1997), e.g., Bagging and Random Forest, combining the results of a number of separate learning processes in an intelligent fashion. Voting, arbitrating, and combining techniques can be used (Prodromidis et al., 2000) with this purpose. Nevertheless, a major issue of meta-learning is obtaining models that represent a good generalization of all the data, considering that they have been build from incomplete local data (Secretan, 2009). Adopting Windowing as a subsampling process in meta-learning methods, or using it as a mean to boost time performance on the induction of individual trees in a forest, are very interesting lines of research.

JaCa-DDM was very useful for implementing and deploying the strategies discussed in this paper. Upcoming improvements to this system include usability issues as: adding more primitive Weka/MOA based models to be used in user defined strategies; providing a web based interface to easily configure and launch experiments at distance; and a description language with an associated graphic tool for designing strategies.

Beyond the technical results, the discussed JaCa-DDM strategies, and the tool itself, are intended to bolster the interest on the alternative Agent Mining approach to DDM, supported by the current advances on Agent Oriented Software Engineering, e.g., Jason and CArtAgO Artifacts. We expect that the promising results showed by Parallel Counter GPU Extra, promote the use of JaCa-DDM to model and implement better strategies, exploiting the social dimension of Multi-Agent Systems.

Acknowledgments

First author was funded by Conacyt scholarship 362384. This work has been partly supported by the Spanish Ministry of Science and Innovation project TIN2015-66972-C5-5-R.

References

Acosta-Mesa, H.G., Cruz-Ramírez, N., Hernández-Jiménez, R., 2009. Aceto-white temporal pattern classification using k-nn to identify precancerous cervical lesion in colposcopic images. Computers in biology and medicine 39, 778–784.

Albashiri, K.A., Coenen, F., 2009. Agent-enriched data mining using an extendable framework, in: Agents and Data Mining Interaction. Springer, pp. 53–68.

Bailey, S., Grossman, R., Sivakumar, H., Turinsky, A., 1999. Papyrus: a system for data mining over local and wide area clusters and super-clusters, in: Proceedings of the 1999 ACM/IEEE conference on Supercomputing, ACM. p. 63.

Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B., 2010. Moa: Massive online analysis. The Journal of Machine Learning Research 11, 1601–1604.

Bordini, R.H., Hübner, J.F., Wooldridge, M., 2007. Programming Multi-Agent Systems in Agent-Speak using Jason. John Wiley & Sons Ltd.

Breiman, L., 1996. Bagging predictors. Machine learning 24, 123–140.

Breiman, L., 2001. Random forests. Machine learning 45, 5–32.

Cao, L., Weiss, G., Philip, S.Y., 2012. A brief introduction to agent mining. Autonomous Agents and Multi-Agent Systems 25, 419–424.

Chan, P.K., Stolfo, S.J., 1997. On the accuracy of meta-learning for scalable data mining. Journal of Intelligent Information Systems 8, 5–28.

Chawla, N.V., Moore, T.E., Hall, L.O., Bowyer, K.W., Kegelmeyer, W.P., Springer, C., 2003. Distributed learning with bagging-like performance. Pattern recognition letters 24, 455–471.

D'Addabbo, A., Maglietta, R., 2015. Parallel selective sampling method for imbalanced and large data classification. Pattern Recognition Letters 62, 61–67.

Domingos, P., Hulten, G., 2000. Mining high-speed data streams, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 71–80.

Fürnkranz, J., 1998. Integrative windowing. Journal of Artificial Intelligence Research 8, 129–164.

Gehrke, J., Ganti, V., Ramakrishnan, R., Loh, W.Y., 1999. Boat—optimistic decision tree construction, in: ACM SIGMOD Record, ACM. pp. 169–180.

Genuer, R., Poggi, J.M., Tuleau-Malot, C., Villa-Vialaneix, N., 2015. Random forests for big data. arXiv preprint arXiv:1511.08327 .

Kargupta, H., Byung-Hoon, D.H., Johnson, E., 1999. Collective data mining: A new perspective toward distributed data analysis, in: Advances in Distributed and Parallel Knowledge Discovery, Citeseer.

Lichman, M., 2013. UCI machine learning repository. URL: `http://archive.ics.uci.edu/ml/`.

Limón, X., Guerra-Hernández, A., Cruz-Ramırez, N., Acosta-Mesa, H.G., Grimaldo, F., 2015. A windowing based gpu optimized strategy for the induction of decision trees in jaca-ddm, in: Artificial Intelligence Research and Development: Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence, IOS Press. p. 100.

Limón, X., Guerra-Hernández, A., Cruz-Ramírez, N., Grimaldo, F., 2013. An agents & artifacts approach to distributed data mining, in: Castro, F., Gelbukh, A., Mendoza, M.G. (Eds.), 11th MICAI, Springer Verlag, Berlin Heidelberg. pp. 338–349.

Lo, W.T., Chang, Y.S., Sheu, R.K., Chiu, C.C., Yuan, S.M., 2014. Cudt: a cuda based decision tree algorithm. The Scientific World Journal 2014.

Ma, W., Agrawal, G., 2009. A translation system for enabling data mining applications on gpus, in: Proceedings of the 23rd international conference on Supercomputing, ACM. pp. 400–409.

Mehenni, T., Moussaoui, A., 2012. Data mining from multiple heterogeneous relational databases using decision tree classification. Pattern Recognition Letters 33, 1768–1775.

Melgoza-Gutiérrez, J., Guerra-Hernández, A., Cruz-Ramírez, N., 2014. Collaborative data mining on a BDI multi-agent system over vertically partitioned data, in: Gelbukh, A., Castro-Espinoza, F., Galicia-Haro, S.N. (Eds.), 13th Mexican International Conference on Artificial Intelligence: Special Session, Revised Papers, IEEE Computer Society, Los Alamitos, CA, USA. pp. 215–220.

Moemeng, C., Gorodetsky, V., Zuo, Z., Yang, Y., Zhang, C., 2009. Agent-based distributed data mining: A survey, in: Data mining and multi-agent integration. Springer, pp. 47–58.

Moemeng, C., Zhu, X., Cao, L., Jiahang, C., 2010. i-analyst: An agent-based distributed data mining platform, in: Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, IEEE. pp. 1404–1406.

Nguyen, H.L., Woon, Y.K., Ng, W.K., 2014. A survey on data stream clustering and classification. Knowledge and Information Systems , 1–35.

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with cuda. Queue 6, 40–53.

Omicini, A., Ricci, A., Viroli, M., 2008. Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17, 432–456.

Prodromidis, A., Chan, P., Stolfo, S., 2000. Meta-learning in distributed data mining systems: Issues and approaches. Advances in distributed and parallel knowledge discovery 3.

Quinlan, J.R., 1993. C4. 5: programs for machine learning. Morgan kaufmann.

Ricci, A., Piunti, M., Viroli, M., 2011. Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems 23, 158–192. doi:`10.1007/s10458-010-9140-7`.

Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., et al., 2012. Fiji: an open-source platform for biological-image analysis. Nature methods 9, 676–682.

Secretan, J., 2009. An Architecture for High-Performance Privacy-Preserving and Distributed Data Mining. Ph.D. thesis. University of Central Florida Orlando, Florida. Orlando, FL., USA.

Sharp, T., 2008. Implementing decision trees and forests on a gpu, in: Computer Vision–ECCV 2008. Springer, pp. 595–608.

Stefanowski, J., 2013. Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data, in: Emerging paradigms in machine learning. Springer, pp. 277–306.

Stolfo, S.J., Prodromidis, A.L., Tselepis, S., Lee, W., Fan, D.W., Chan, P.K., 1997. Jam: Java agents for meta-learning over distributed databases., in: KDD, pp. 74–81.

Tran, N.L., Dugauthier, Q., Skhiri, S., 2013. A distributed data mining framework accelerated with graphics processing units, in: Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on, IEEE. pp. 366–372.

Tsoumakas, G., Vlahavas, I., 2009. Encyclopedia of Data Warehousing and Mining. second ed.. Information Science Reference, Hershey, PA., USA. chapter Distributed Data Mining. pp. 709–715.

Wang, X.Y., Zhang, X.J., Yang, H.Y., Bu, J., 2012. A pixel-based color image segmentation using support vector machine and fuzzy c-means. Neural Networks 33, 148–159.

Witten, I.H., Frank, E., 2005. Data mining: Practical machine learning tools and techniques. Second ed., Morgan Kaufmann, San Francisco, CA., USA.

Wojnarski, M., Stawicki, S., Wojnarowski, P., 2010. TunedIT.org: System for automated evaluation of algorithms in repeatable experiments, in: Rough Sets and Current Trends in Computing (RSCTC), Springer. pp. 20–29.

Xu, J., Yao, L., Li, L., Chen, Y., 2014. Sampling based multi-agent joint learning for association rule mining, in: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems. pp. 1469–1470.

Yoon, H., Alsabti, K., Ranka, S., 1999. Tree-based incremental classification for large datasets. Technical Report. CISE Department, University of Florida.

Zhong, N., Matsui, Y., Okuno, T., Liu, C., 2002. Framework of a multi-agent kdd system, in: Intelligent Data Engineering and Automated Learning—IDEAL 2002. Springer, pp. 337–346.

Distributed Transparency in Endogenous Environments: the JaCaMo Case

Xavier Limón[1], Alejandro Guerra-Hernández[1], Alessandro Ricci[2]

[1] Universidad Veracruzana, Departamento de Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000
`xavier120@hotmail.com, aguerra@uv.mx`
[2] DISI, Università di Bologna, Via Sacchi, 3, Cesena, Italia 46100
`a.ricci@unibo.it`

Abstract. This paper deals with distribution aspects of endogenous environments, in this case, distribution refers to the deployment in several machines across a network. A recognized challenge is the achievement of distributed transparency, a mechanism that allows the agent working in a distributed environment to maintain the same level of abstraction as in local contexts. In this way, agents do not have to deal with details about network connections, which hinders their abstraction level, and the way they work in comparison with locally focused environments, reducing flexibility. This work proposes a model that creates a distinctive layer for environment distribution, which the agents do not manage directly but can exploit as part of infrastructure services. The proposal is in the context of JaCaMo, the Multi-Agent Programming framework that combines the Jason, CArtAgO, and MOISE technologies, specially focusing on CArtAgO, which provides the means to program the environment. The proposal makes an extensive use of the concept of workspace to organize the environment and transparently manage different distributed sites.

Keywords: Distributed environments, Endogenous environments, Environment Programming, JaCaMo framework.

1 Introduction

Traditionally, agents are conceived as entities situated in an environment, which they can perceive and modify through actions, also reacting to changes in it accordingly [16]. Not only that, but the agents' goal is to achieve an environment desired state. This conception of environment, as the locus of agent perception, action, reaction, interaction, and goals, stays true in current MAS development.

Two general perspectives are adopted when defined the concept of environment in MAS: exogenous, and endogenous [14]. The exogenous perspective is rooted in Artificial Intelligence, conceiving the environment as the external world, separated to the actual MAS, which can be only perceived and acted upon by agents. An example of this conception can be found in EIS [1]. In contrast, the endogenous perspective, grown in the context of Agent-Oriented Software Engineering (AOSE) [10], conceives the environment as a first class abstraction for MAS engineering [17], that not only can be perceived and acted upon, but it can also provide services and tools for the agents to aid them in their tasks, and as such, it is designed and implemented as part of the whole system. An example of this conception is found in CArtAgO [13].

From a Software Engineering point of view, environments can also be of two types: local, and distributed. In the local case, the entirety of the environment is centralized in a single process, being the easiest case for implementation. Distributed environments, on the other hand, entail multiple processes, possibly across a network, to contain the environment, and presents various challenges in the implementation and conceptualization side. In this work, we expose the idea that, from the agents point of view, there should not be any difference between local and distributed environments, the right level of abstraction should be encouraged instead, recognizing this by the term Distributed Transparency.

Distribution is not a new topic in MAS, multiple MAS technologies such as JADE [2], have a mature support for it, but it is mostly centered in agent communication and interaction, not on the endogenous conception of environment which this works entails. In this regard, JaCaMo [4] is a better example, as it supports endogenous distributed environments. This support is practical, but it lacks distributed transparency as there is a clear conceptual distinction between local and distributed environments, so agents, and agent plan programmers, need to handle the difference, reducing in this way the flexibility of the system, and forcing the programmer to change the level of abstraction in each case.

Scalability and fault tolerance are also issues when dealing with distribution, a flexible configuration is required in order to deploy the system in different settings, allowing it to grow or change as network problems arise. A good example of a distributed deployment system for JADE is [6]. Returning to the case of JaCaMo, there is no support for fault tolerance, and it lacks proper configuration facilities for distributed deployment.

This work proposes an extension to the Agents & Artifacts model of JaCaMo for modeling distributed transparent environments, while giving insights of how to address distributed deployment and fault tolerance. The outcome is an implemented JaCaMo-oriented infrastructure and Agent API that gives support to the mentioned requirements, while extending the dynamics and possibilities of MAS programming in general.

The paper is organized as follows. Section 2 briefly introduces the JaCaMo framework, addressing its distributed model. Section 3 introduces the problems present in the JaCaMo distributed model, presenting a proposal to solve them, first in an intuitive and informal manner, and then formally. Section 4 features different aspects of the implementation, such as the general architecture, and configuration and deployment. Section 5 discusses a case study that shows how the new JaCaMo-oriented implementation approach compares to current JaCaMo, giving code examples. Being a work in progress, section 6 discusses various topics regarding future work, including proper evaluation and fault tolerance implementation. Finally, section 7 closes this paper with a conclusion.

2 Background

Although the discussion here is about endogenous environments in general, we adopt the JaCaMo [4] framework to implement our proposed model and guide our discussion, this is due the fact that, to the best of our knowledge, it has the most mature implementation of endogenous environments for MAS. As such, a brief introduction of JaCaMo is presented in this section. JaCaMo is the result of the orthogonal composition of three technologies for MAS: Jason [5] (taken as a proper name inspired by Greek mythology), CArtAgO [13] (Common ARTifact infrastructure for AGents Open environments), and MOISE [9] (Model of Organisation for multI-agent SystEms).

Jason provides the means for programming autonomous agents. It is an agent oriented programming language that entails the Belief-Desire-Intention (BDI) approach, it is based on the abstract language $AgentSpeak(L)$ [12]. Apart from its solid BDI theoretical foundations, the language offers several facilities for programming Java powered, communicative MAS. Communication in Jason is based on Speech Acts, as defined in KQML [7].

CArtAgO provides the means to program the environment, following an endogenous approach where the environment is part of the programmable system. In CArtAgO terms, the aspects that characterize a model for environment programming are the following [14]: 1) Action model: how to perform actions in the environment. 2) Perception model: how to retrieve information from the environment. 3) Environment computational model: how to represent the environment in computational terms. 4) Environment data model: how to share data between the agent and environment level to allow interoperability. 5) Environment distributed model: how to allow computational distributed environments. Aspects 1-3 are directly supported by artifacts [11], which are dynamical sets of computational entities that compose the environment and encapsulate services and tools for the agents. Artifacts are organized and situated in workspaces, which essentially are logical places (local or remote) where agents center their attention and work. Aspect 5 is supported by workspaces, but also partially by artifacts, as artifact actions can be executed remotely. Aspect 4, on the other hand, depends on the underlying agent programming language used and is not directly related to artifacts or workspaces.

MOISE provides the means to create agent organizations, which have the aim to control and direct agent autonomy in a general purpose system. To this end, it is possible to specify tree aspects: i) Structural, consisting on the different agent groups and roles that take part in the organization; ii) Functional, defined by social schemes, missions, and goals which direct the agent behaviour toward organization ends; and finally iii) Normative, defined though norms that bind roles to missions, constraining agent's behaviour when entering a group and playing a certain role.

2.1 JaCaMo and CArtAgO Distribution Model

As mentioned earlier, environment programming in JaCaMo is provided by means of CArtAgO, considering distribution in its model. At the higher level, distribution is achieved through workspaces, which serve as logical places where agents may center their attention, and where artifacts are situated. Agents can create, join, and quit workspaces. If an agent is in a workspace, it can use the artifacts situated there.

At the low level, nodes enable distribution. A node is a CArtAgO process that can be remote, where workspaces can be spawned. When a JaCaMo MAS is deployed, it is contained in a default node, that node is also the default for agents, which consider it as it's local context, so workspaces created in that node are also local workspaces, but workspaces created in different nodes are considered remote workspaces. The distinction between remote and local workspace is not only conceptual, but also syntactical, requiring IP and port information at the agent level to manipulate remote workspaces. Figure 1 depicts the current CArtAgO environment model from the workspaces and nodes perspective. From the figure, it is apparent the fact that there is no connection between nodes, and in consequence between workspaces in different nodes, needing to explicitly know the IP address and port of each node.

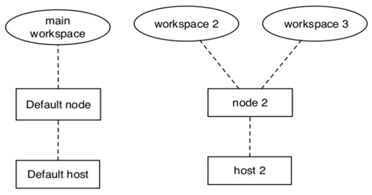


Fig. 1. Current CArtAgO environment model depicting multiple nodes and workspaces deployed.

More concretely, the following code snippet shows the difference in the JaCaMo API for the local and remote versions of join workspace, taking as a basis figure 1 where *default node* represents the local node, and *node*2 a remote one:

```
joinWorkspace("main", WspId1);
joinRemoteWorkspace("workspace2", "192.168.0.2:8091", WspId2);
```

3 Proposal

Environment programming in JaCaMo comes with various shortcomings regarding distributed settings, being the most important the fact that local and remote workspaces are defined and treated differently, which derives in the following problems: i) There is not distributed transparency for agents, being forced to directly manipulate network information, making network distinctions between workspaces. ii) The underlying environment topology is difficult to represent and exploit by the agents as it does not follow any structure or workspace relations beyond the sharing of the same node. All of these problems have the consequence of reducing the abstraction level in which agents work, impacting flexibility and environment exploitation as well.

Another problem is the lack of proper configuration facilities to allow the inclusion of remote workspaces information at deployment time, meaning that host information for remote workspace spawning need to be hard-coded on the agent programs or externally supported. To spawn a remote workspace, a CArtAgO node needs to be running on the destination host, and there is not any integrated facility to manage them automatically when needed. Furthermore, the current distributed implementation does not exhibit any degree of fault tolerance, this is specially important for possible network connection problems that may arise in a distributed system.

In this section, a proposal to solve the identified problems is presented. A separation between environment and infrastructure is suggested. The environment is represented as a hierarchical tree structure, which represents the topology. In this tree, each node is a workspace which actual physical placement on the distributed system is irrelevant. Workspaces may be deployed in different places, but for the agents point of view, it only matters their placement in the topology. A workspace may be the logical parent of another one, multiple workspaces can be in the same physical place, and there is no restriction about how the topology may be organized, e.g.; workspaces on the same physical place may be on different branches. This allows to organize environments as it is usually done in CArtAgO, but in a more structured way, also supporting remote workspaces transparently.

In a practical sense, each workspace in the tree is represented by a path starting at the root workspace, these paths brings the notion of logical placement that agents require to organize and exploit their environment. We adopt a Unix-like path format to represent this placement, but using a ”.” instead of a ”/”, following Jason syntax. These paths are used by the agents to execute environment related actions, such as creating new workspaces or joining one. From the proposed JaCaMo API, there is no difference between local and remote actions related to workspaces. For example, returning to the code snipped presented in section 2.1 for joining local and remote workspaces, which it is related to figure 1; with the proposal, a workspace topology would be created, a possibility is to have $workspace2$ and $workspace3$ as direct descendants of the root workspace *main*, with this setting the associated code snipped is as follows:

```
    joinWorkspace("main", WspId1);
    joinWorkspace("main.workspace2", WspId2);
```

As in current CArtAgO, agents may work in multiple workspaces at the same time, but the concept of current workspace is dropped since in actuality all the joined workspaces should be considered the current context of working. Nevertheless, agent may specify the target workspace for an action. A new introduced concept is the home workspace of an agent, which it is the workspace where the agent is initially deployed, serving as a relative reference to other places in the topology, providing a default place for the agent, and also serving as the default workspace to execute actions when a target workspace is not specified.

On regard of the infrastructure, a layer is added to manage distribution, this layer provides the required services for the agents to exploit their distributed environment. These services include: i) Workspace management, so agents can create, join, and quit workspaces no matter their physical placement; ii) Topology inspection, so agents can reason about the current topology organization and do searches concerning workspaces; iii) Workspace dynamics observation, so agents can know when other agents manage workspaces, or when workspaces disconnect and reconnect after a network problem; iv) Disconnection and fault tolerance to manage and recuperate from network problems, which it is currently left as future work, but initially designed as presented in section 6.2 . We believe that the set of mentioned services do not only bring distributed support, but also enhance the dynamics of MAS in general, extending its possibilities.

3.1 Formal description

JaCaMo assumes an endogeneous approach to MAS, i.e., the environment is an explicit part of the system:

Definition 1. *A MAS is composed by a set of agents (Ags), their environment (Env), and an infrastructure ($Infr$) running both of them:*

$$MAS = \{Ags, Infr, Env\}$$

The set of agents is composed by $n \geq 1$ agents:

$$Ags = \{a_1, \dots, a_n\}$$

Each agent, as usual, is composed by beliefs, actions, and other elements equal to:

$$a_i = \{bels, acts, \dots\}$$

By default, when created, an agent includes minimally:

$$a_i = \{joined(home)\}, \{join, quit, create\}, \dots\}$$

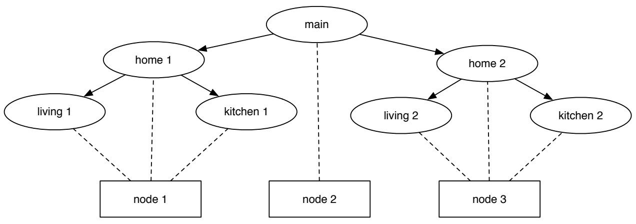


Fig. 2. The intented view of an endogeneous environment.

which means that every agent believes he has joined a home workspace, and has actions to join, quit, and create workspaces; and update the information about the environment.

Figure 2 illustrates the intended view of the environnment in this proposal. First, the environment, properly speaking, is a tree of workspaces, expressing a kind of spatial relation among workspaces, e.g., the kitchen 1 is at the home 1. Second, nodes and hosts are not part of the environment, but are defined as part of the infrastructure of the MAS, nevertheless, workspaces keep information about its corresponding physical node.

The infrastructure is a layer hidden to the agents, that gives the low level support to distribution, formally defined as:

$$Infr = \{Nodes, Hosts\}$$

where:

- $Nodes = \{node_1, \dots, node_k\}$ is a set of CArtAgO nodes, i.e.; processes, possibly remote, where workspaces can be created. Each $node_i$ is a tuple $\langle ni, SWsps, hi, port \rangle$, where ni is an unique identifier for the node; $SWsps \subseteq W$ is the set of spawned workspaces in the node, containing at least a default workspace for the node; hi is an identifier of the host computer where the node exists; and $port$ is the host port used by the node process.
- $Hosts = \{host_1, \dots, host_p\}$ is the set of available computer devices on the distributed system. Each $host_i$ is a tuple $\langle hi, HNodes, ip \rangle$, where hi is a host unique identifier, $HNodes \subseteq Nodes$ is a set of hosted nodes, and ip is the IP address of the computer.

Formally, the environment Env is defined as a graph:

$$Env = \{W, E\}$$

where:

- $W = \{w_1, \dots, w_i\}$ is a finite, non-empty set of $i \geq 1$ workspaces that contain artifacts. Each $w_i = \langle idW, name, ni \rangle$, where idW is an unique identifier for the workspace, $name$ is a logical name, and ni is a reference to the CArtAgO node in $Infr$ that contains w_i. The *node* element establishes a connection between the environment and the infrastructure, in order to forward agent actions to the destined physical place.
- $E \subset W^2$ is a set of edges over the workspaces, such that Env is minimally connected, i.e., it is a rooted tree that represents the environment topology.

For instance, following Figure 2, $Env = \{W, E\}$, and considering for simplicity only the name of each w_i, such that:

- $W = \{main, home1, home2, living1, kitchen1, living2, kitchen2\}$
- $E = \{\{main, home1\}, \{main, home2\}, \{home1, living1\}, \dots\}$

The expression $w_1.w_2 \dots .w_n$ denotes a path on Env, if:

- $w_i \in W$ for $i = 1, \dots, n$;
- $\{w_{i-1}, w_i\} \in E$ for $i = 2, \dots, n$.

Abusing a little bit of the notation, we can write $w_1. \dots .w_n \in Env$. For instance, $main.home1.living1 \in Env$. Some useful operations over paths, include:

- $last(w_1.w_2. \dots .w_n) = w_n$
- $butlast(w_1.w_2. \dots .w_{n-1}.w_n) = w_1.w_2. \dots .w_{n-1}$
- $add(w, w_1.w_2. \dots .w_n, Env) = w_1.w_2. \dots .w_n.w$. This involves adding w to W, and $\{w_n, w\}$ to E in Env.
- $del(w, w_1.w_2. \dots .w_n.w, Env) = w_1.w_1. \dots .w_n$. This involves deleting w from W, and $\{w_n, w\}$ from E in Env.

In what follows, the transition rules related to environment agent actions are described, workspaces denote paths in the environment.

Joining a workspace An agent can ask himself about the workspaces he has currently joined: $ag_{bels} \models joined(w)$, if and only if, w is a workspace currently joined by the agent. Recall that by default $ag_{bels} \models joined(home)$. An agent can join different worspaces concurrently, so that $ag_{bels} \models joined(Ws)$ unifies Ws with a list of the workspaces joined by the agent. Two transtion rules define the behavior of the action *join*. First, an agent can join a worspace w, if and only if w is a path in the environment Env and it is not believed to be already joined:

$$(\mathbf{join_1}) \qquad \frac{join(w) \mid w \in Env \land ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \to \langle ag', Env \rangle}$$
$$s.t.\ ag'_{bels} = ag_{bels} \cup \{joined(w)\}$$

Second, nothing happens if an agent tries to join a previously joined worspace:

$$(\mathbf{join_2}) \ \frac{join(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \to \langle ag, Env \rangle}$$

Any other use of *join* fails.

Quiting workspaces An agent can quit the workspace w if he believes he had joined w. The agent forgets such belief.

$$(\mathbf{quit_1}) \quad \frac{quit(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env\rangle \rightarrow \langle ag', Env\rangle}$$
$$s.t.\ ag'_{bels} = ag_{bels} \setminus \{joined(w)\}$$

If the agent tries to quit a workspace he has not joined yet, nothing happens:

$$(\mathbf{quit_2}) \quad \frac{quit(w) \mid ag_{bels} \not\models joined(w)}{\langle ag, Env\rangle \rightarrow \langle ag, Env\rangle}$$

Creating workspaces An agent can create a workspace w, if it is not a path in the environment, but $butlast(w)$ is one:

$$(\mathbf{create_1}) \quad \frac{create(w) \mid w \notin Env \wedge butlast(w) \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env'\rangle}$$
$$s.t.\ Env' = add(last(w), butlast(w), Env)$$

Observe that the result of creating a workspace must be propagated to the rest of the agents in the MAS. This could be done by the infrastructure, or broadcasting the *add* operation. The actual node where the workspace is going to be created is decided by the infrastructure following a policy, by default the infrastructure spawns the workspace on the same node where its parent workspace is running.

Trying to create an existing workspace does nothing:

$$(\mathbf{create_2}) \quad \frac{create(w) \mid w \in Env}{\langle ag, Env\rangle \rightarrow \langle ag, Env\rangle}$$

4 Implementation

The model introduced on section 3 is open enough to allow different implementations. This section presents a practical possibility, intended to be integrated with JaCaMo. The core implementation and main design choices are related to the general architecture, and configuration and deployment.

4.1 General architecture

The architecture to support agent services is based on the concept of Node, which refers to the *Nodes* element in $Infr$. Nodes represent independent CArtAgO processes, possibly remote, running on a given host (*Hosts* element in $Infr$), and associated to a port. Nodes are the main abstraction to manage workspaces (W element in Env), and as such, they provide all the necessary tools to create, join, and quit workspaces, as well as the means to communicate with other nodes in order to maintain a consistent workspace topology, and properly dispatch topology related events. The workspace topology corresponds to the E element in Env. A NodeArtifact is the gateway used by an agent to interact with the node services and to observe the distributed environment. There is a NodeArtifact in each workspace, and every agent has access to one of them, which one depends on its *home* workspace, which in turn it is intended to be on the same node as the agent process.

Nodes communicate between each other following a centralized approach: one node is designated as the central node, this is usually the node deployed by default by JaCaMo, so every change on the topology is inspected and approved by a single node, and the associated actions and events are dispatched from there. This centralized approach makes possible to maintain a consistent topology, avoiding run conditions. To exemplify node communication, the workflow for creating a new workspace is the following:

- An agent that wants to create a workspace issues the action to its corresponding NodeArtifact, passing a tree path.
- The artifact checks if the tree path is consistent with the topology tree, if it is, it sends a request to the central node.
- The central node issues a request to the end node where the workspace is actually going to exist. By default, it chooses as end node the same as the parent workspace from the path given.
- The end node creates the workspace and returns control to the central node.
- The central node makes the corresponding changes to the workspace topology and communicates the success to the original requesting node. It also dispatches a create and tree change event to the other nodes.

As the node model is centralized, there exists the concern of a single point of failure, that is why all nodes maintain redundant information about the topology, so it is possible to recuperate from a central node dropping, as any node can take the role of central node. The topology structure is also lightweight, which speeds up the tree synchronization among nodes, this synchronization is only required when there is a change in the topology. This redundancy also allows to boost the efficiency of operations such as joining and quitting workspaces, since those operations only need to read from the topology, so the local copy is used in those cases. Communication with the central node is only required in cases where a change in the topology is required. We believe that in traditional environment management, it is more common for the agents to join and quit workspaces than to create new ones.

4.2 MAS configuration and deployment

To ease the deployment of the distributed infrastructure is a goal of our overall proposal, this means to be able to configure and launch the desired hosts, nodes, and workspaces that will take part in the MAS from the start. It is also possible to manually add new nodes after deployment. The idea is to extend the deployment of JaCaMo, where only workspaces are considered. JaCaMo uses a text file known as the JCM file to configure the deployment of the MAS. The intention is to further include fields in this file to also configure host, and nodes for distributed systems; and add the facilities to automatically launch CArtAgO nodes in remote machines through a daemon service.

The changes to the JCM file include:

- Host configuration: assign a logical name and IP address to each host.
- Node configuration: assign a logical name for the node, i.e.; the name of the default workspace; the related host name; and optionally a port number.
- Workspaces configuration: relate each workspace to a specific node.
- Lib file: the path to a jar file containing all the necessary files to launch CArtAgO nodes. This includes custom artifacts binaries, third party libraries, custom classes binaries, and any configuration file. This file is intended to be shared among all nodes.

5 Case study

In order to show how to exploit the proposed distributed model and implementation, and how it compares to the current version of JaCaMo, a small case study is presented in this section. This case study pretends to show the new proposed agent API, and the flexibility of the proposed model, focusing mainly on workload distribution, which is one of the aspects that can be enhanced, but other aspects such as fault tolerance, reactiveness on environment changes, and more complex agent organizations and dynamics are also possible.

The case study consists on constantly fetching current weather information for every city in a country, and with that information, construct weather prediction models for each city, so the models could be consulted by end users through a user interface. The construction of each model is an online learning [3] task that can be heavy on the computational side, so work distribution is desirable. To simplify the distributed setting, assume that the cities of the country are divided in west-cities and east-cities, one computer is in charge of the models from the west-cities, and another one from the models of the east-cities; furthermore, information fetching need to be quick and constant, and also the end user searching service, so one computer takes care of both. The described setting yields a total of three different computers in the distributed system.

Workflow is modeled through 3 agent programs: fetcher, which fetches weather information and forwards it to a destination depending the type of city; learner, which task consist on creating online weather prediction models for each city with the data provided by the fetcher agent; and finally, a searcher agent, which attends end user requests to retrieve information from the learned models.

When implementing this case study in current JaCaMo some problems will arise: the IP addresses of the different computers should be hard-coded in the agent programs; every CArtAgO node representing a remote workspace should be manually started in every computer on the distributed setting; startup plans should be implemented in order to focus the attention of each agent in its designated place; when the searcher agent has to collect information about learned models to attend a petition, it necessarily has to ask learner agents about its location, not only considering the workspace but also the ip address where the workspace is, making also necessary to distinguish between local and remote workspaces when the agent intend to return the result to the end user. A better solution using our new approach that solves all the mentioned problems, and better exploits the environment is presented next.

– A possible JACAMO configuration file for this example, following the idea from section 4.2, is the following. For the sake of clarity, artifact related configuration, and MOISE related organization is not shown.

```
mas weather {
   host c1 {
     ip: 192.168.0.2
   }
   host c2 {
     ip: 192.168.0.3
   }
   node west {
     host: c1
     port: 8080
   }
   node east {
     host: c2
     port: 8080
   }
   agent fetcher : fetcher.asl {
     home: main
   }
   agent west_learner : learner.asl {
     home: main.west
   }
   agent east_learner : learner.asl {
     home: main.east
   }
   agent searcher : searcher.asl {
     join: main.central
   }
   lib_file : /home/system/wheather.jar
}
```

Note that workspaces *main*, *main.west*, and *main.east* are implicitly created as they are the default workspaces for the nodes, e.g.; *main* is the node deployed by default by JaCaMo.

In our proposed model, agents refer to workspaces only through their path in the topology, giving in this way a more structured sense of placement. As in UNIX file system paths, tree paths can be considered absolute or relative. A path is relative from the *home* workspace of the agent, and should start with a single dot, any other path is considered absolute and must begin at the root workspace (by default *main*).

A simplified version of the agent programs is presented next.

– fetcher:

```
//creations and initializations omitted
+!fetch : true <-
  getData(Data);
  category(Data, Cat);
  if(Cat == "west") {
    .send(west_learner, tell, add(Data));
  }
  else {
    .send(east_learner, tell, add(Data));
  };
  !fetch.
```

– learner:

```
//initialization of agent omitted
+add(Data): true <-
   getCity(Data, City);
   .concat(".", City, Path); //from home
   createWorkspace(Path); //does nothing if already present
   joinWorkspace(Path);// does nothing if already joined
   //creations and initializations omitted
   addData(Data)[wsp(Path)]; //route action to wsp
   induceModel[wsp(Path)].
```

– searcher:

```
+search(Query) : true <-
   //Query is just the name of the city
   .concat(".*", Query, RegExp);
   searchPaths("main", RegExp, [H | T]);
   .length(List, Len);
   if(Len > 0) {
     joinWorkspace(H);
     getForecast(Forecast)[wsp(H)];
     quitWorkspace(H);
     sendReply(Forecast);
   }.
```

Some of the actions from the agent programs correspond to the API of the proposed model, such actions are described as follows.

– $joinWorkspace(+WspPath, -WspId)$: the agent adds a specified workspace to its joined workspaces list, obtaining a workspace ID.
– $quitWorkspace(+WspPath)$: removes a specified workspace from its joined workspaces.
– $createWorkspace(+WspPath)$: creates a new workspace on the specified path. By default, the workspace will actually be spawned on the same CArtAgO node as the parent workspace derived from $WspPath$, this allows workload management for workspaces.
– $searchPaths(+PointPath, +RegExp, -WspsPathList)$: returns a list with the workspaces that follow a certain regular expression, the search is restricted to the subtree given by $PointPath$. This action exemplifies how the topology organization may be exploited.

It is worth mentioning that the actual API is more extensive, including perception, events, and more actions related to the environment. For example, it is possible for the agents to react to the creation of a new workspace through the event $created(WspPath)$, or analyze and exploit the current topology organization through the perception $topology\_tree(List)$ where $List$ is of the form: $[main, [subNode_1, [subsubNode_1, [...], subsubNode_m]], ..., [subNode_n]]$.

As the example shows, agents do not concern about computer distribution, they simply work with workspaces: learner agents organize their work in workspaces that can be on different computers; and the searcher agent can reach any workspace directly, not relying on agent communication, but directly acting though the knowledge of the topology. Deployment is also greatly enhanced since the distributed setting is properly configured beforehand, and the launching of nodes is automatic.

6 Discussion and future work

As a work in progress, our proposal still lacks a proper treatment of different aspects such as evaluation, and fault tolerance. This sections briefly discusses and outlines these topics, which are considered as immediate future work.

6.1 Evaluation

Our proposed model, while introduced in some formal way, still needs a proper formal evaluation, for this end, the adoption of a more formal representation such as the ones proposed in [8], and [15] seem to be required.

Concerning performance evaluation, with the adopted centralized model, it is required to asses scalability issues that may arise as nodes are added to the MAS. In case of finding such problems, we can still improve some of the required subprocess, as the synchronization of the topology among all nodes, and event propagation, which could be distributed.

6.2 Fault tolerance

Given the proposed architecture, connection loss is the same as node dropping, and as such it directly impacts the topology tree structure used by agents as all the corresponding workspaces are also lost. An intuitive idea of how fault tolerance could be implemented following our design choices is described next.

Following the overall node organization introduced in section 4.1, all nodes maintain a keepalive connection with the central node, and a ordered list of connected nodes. If a node losses connection, then the central node issues the corresponding dropping event to the rest of the nodes, and modifies the topology tree structure accordingly. The disconnected node tries to establish a connection with the rest of the nodes, following the order of the connected nodes list, this being useful on the case that several nodes lost connection or the central node dropped. With the available nodes (it may be only one), the node in the upper position on the connected nodes list is designated as the new central node, issuing the corresponding disconnection events and creating a new tree node structure where every default workspace from the nodes is on the same level. The new central node keeps trying to reconnect to the original central node for a period of time.

When successfully reconnecting, the original central node will try to return the topology to the way it was before the problem, but sometimes that would not be possible, e.g.; when one of the nodes keep missed. It is strongly recommended that every default workspace corresponding to a node is mounted on the same upper tree level of the tree, that way when reconnecting, the tree structure will keep consistent with the way it was before, otherwise the tree topology may vary in an unexpected manner, which can be problematic on certain applications. After the node tree structure is recreated, the reconnecting nodes return to work with the original central node, and the central node triggers the corresponding reconnection events.

7 Conclusion

The introduced model is a step forward to improve environment programming for MAS, it addresses issues related to distribution, which are important for a wide variety of applications. We see distributed transparency as the most important contribution of this work, as Multi-Agent Systems are intended to raise the level of abstraction in software development, as compared with other industry established programming paradigms such as POO. Proper abstraction levels for aspects such as concurrency management are already accomplished, but distributed computing is still an important topic to improve.

With a solid foundation for distributed environment programming, it is possible to address new challenges like MAS interoperability, which refers to the integration and collaboration of independent Multi-Agent Systems. An extension to the proposed model and implementation is envisaged to support MAS interoperability features, such as MAS composition where two different MAS can fuse together to extend the scope of their work; and also MAS attachment, where a mobile MAS can temporally join a MAS in order to exploit services. These features bring new possibilities to the dynamics of MAS in general, and are also interesting from the software engineering point of view as they allows an upper level of flexibility and scalability.

References

1. Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.
2. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

3. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.
4. Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.
5. Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason.* John Wiley & Sons Ltd, 2007.
6. Lars Braubach, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. Deployment of distributed multi-agent systems. In *International Workshop on Engineering Societies in the Agents World*, pages 261–276. Springer, 2004.
7. T. Finin et al. An overview of KQML: A knowledge query and manipulation language. Technical report, University of Maryland, CS Department, 1992.
8. Matthew Hennessy. *A distributed Pi-calculus.* Cambridge University Press, 2007.
9. Jomi F Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
10. James J Odell, H Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer, 2002.
11. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.
12. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
13. A. Ricci, M. Viroli, and A. Omicini. Construenda est cartago: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574, 2006.
14. Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.
15. Alessandro Ricci, Mirko Viroli, and Maurizio Cimadamore. Prototyping concurrent systems with agents and artifacts: Framework and core calculus. *Electronic Notes in Theoretical Computer Science*, 194(4):111–132, 2008.
16. Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.
17. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.