ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

Eterogeneità dei sistemi di Aggregate Programming: estensione del sistema ScaFi per l'uso di robot Thymio

Tesi di laurea in:
Objective Oriented Programming

Relatore
Chiar.mo Prof. Mirko Viroli

Candidato
Elvis Perlika

Correlatore
Dott. Gianluca Aguzzi

III Sessione di Laurea
Anno Accademico 2023-2024

Abstract

Max 2000 characters, strict.

Qualsiasi tecnologia sufficientemente avanzata è indistinguibile dalla magia.
Terza Legge di Arthur C. Clarke

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Write your intro here.

Elvis Perlika: Add sidenotes in this way. They are named after the author of the thesis

You can use acronyms that your defined previously, such as Internet of Thing (IoT). If you use acronyms twice, they will be written in full only once (indeed, you can mention the IoT now without it being fully explained). In some cases, you may need a plural form of the acronym. For instance, that you are discussing Virtual Machines (VMs), you may need both VM and VMs.

Structure of the Thesis

Elvis Perlika: At the end, describe the structure of the paper

Chapter 2

Background

2.1 Paradigma dell'Aggregate Programming

2.1.1 Evoluzione dei paradigmi di programmazione

L'Objective Oriented Programming è un paradigma nel senso stretto del termine poiché rappresenta un modo di organizzare e rappresentare un mondo. Il paradigma in questione deve la sua potenza nella capacità di simulare entità reali ed è riassumibile con la frase *Everything is an Object*. È rilevante parlare di OOP in quanto il paradigma di programmazione funzionale, che è alla base di ScaFi, è un'estensione di esso. Il potere della programmazione ad Oggetti (OOP), come detto precedentemente, risiede nella capacità di simulare un mondo e permette di farlo grazie agli "oggetti", essi sono istanze di Classi, le quali a loro volta sono strutture dati astratte che permettono ad ogni loro istanza di avere uno stato (definito dai *fields*) e un comportamento (definito dai *methods*). I pilastri della programmazione ad oggetti sono l'incapsulamento, l'ereditarietà e il polimorfismo.

Il difetto della programmazione ad oggetti è che richiede un particolare impegno nel gestire il sistema da realizzare all'aumentare della sua complessità. Altri paradigmi, come la programmazione funzionale, possono essere più adatti a determinati problemi.

Nel caso della OOP abbiamo riassunto il paradigma con la frase *"Everything is an Object"*, per la programmazione funzionale possiamo riassumerla con *"Everything is a Function"*. Quando si parla di funzioni nel ambito della Functional

Programming (FP) si intendo **funzioni pure** cioè senza effetti collaterali. Per effetti collaterali si intende che la funzione fa altro oltre a restituire un risultato. Un paio di esempi, presi da `Functional Programming in Scala`, sono:

- Modifica di una variabile
- Modifica del campo di un oggetto
- Leggere da o scrivere su un file
- "Disegnare" sullo schermo

Si potrebbe pensare che con l'uso della FP si possano costruire solo programmi semplice, nella realtà non c'é alcuna limitazione sulla complessità del software da costruire poiché il paradigma della FP esprime un nuovo modo di pensare e scrivere il codice.

Nel dettaglio, per **funzione pura** si intende una funzione $f : A \to B$, (una funzione che prende un input di tipo A e restituisce un output di tipo B) che mette in relazione ogni elemento di A con esattamente un valore di B. Qualsiasi altra operazione che non sia utile a calcolare $f(a) = b$ con $a \in A$ e $b \in B$ deve essere intesa come effetto collaterale della funzione e quindi evitata se si vuole creare una funzione pura.

Un esempio di funzione pura, senza effetti collaterali, è la funzione di somma che prende in input due valori e ne restituisce la loro somma listing 2.1.

Listing 2.1: Esempio di funzione pura in Scala

```
1 def sum(a: Int, b: Int): Int = a + b
```

Un esempio di funzione non pura, con effetti collaterali, è la funzione che prende in input un valore e lo stampa a schermo listing 2.2.

Listing 2.2: Esempio di funzione non pura in Scala

```
1 def print(a: Int): Unit = println(a)
```

Formalmente si può definire una funzione pura con il concetto di Referential Transparency (RT):

> Una funzione f è Referentially Transparent se per ogni contesto C nel quale la funzione viene inserita, essa può essere sostituta dal risultato della stessa funzione f senza condizionare il risultato di C.

È proprio questa proprietà che permette ad un programma progettato con approccio funzionale di essere maggiormente scalabile e mantenibile.

2.1.2 Sistemi collettivi ed eterogenei

Per comprendere il potenziale del paradigma dell'Aggregate Programming (AP) è fondamentale definire quale tipo di problema si vuole risolvere. La crescita del mondo IoT ha portato alla presenza di numerosi dispositivi connessi tra loro di ogni tipo e caratterizzati da protocolli di comunicazione eterogenei. Il mondo dell'IoT in continua evoluzione promette di migliore la produzione, gli spazi di lavoro e la sanità attraverso la raccolta e l'analisi di grandi quantità di dati. È nato di conseguenza il bisogno di progettare sistemi resistenti e resilienti che permettessero di gestire queste grandi quantità di dispositivi. È necessario che questi sistemi siano capaci di: adattarsi a cambiamenti imprevedibili, permettere l'implementazione di nuovi comportamenti, permettere una capacità di rilevazione e attuazione nel ambiente in modo distribuito e coordinato. Allo stato attuale della tecnologia è difficile progettare sistemi di questo tipo poiché non esistono framework che permettano di gestire in modo semplice e intuitivo la complessità di questi sistemi.

Un'astrazione grafico del mondo IoT è presente in fig. 2.1. È possibile comprendere come ci sia uno strato contente i nodi (dispositivi)[1], uno strato che rappresenta la rete (internet) mediante la quale comunicano[2] ed uno strato contente il mondo Cloud dove vengono raccolte tutte le informazioni prodotte dai nodi della rete e computate attraverso applicazioni complesse per l'analisi, la pianificazione, l'ottimizzazione ed altri scopi.

Il problema di questa architettura è che inviare queste ingenti quantità di dati da un numero elevato di nodi può avere effetti negati sui costi, le disponibilità, la scalabilità e la latenza delle comunicazioni. Una delle soluzione trovate è quella di

[1] HMI: Human-Machine-Interaction.

[2] Prima di accedere alla rete internet le informazioni dei sensori vengono raccolte dai Edge Gateway che si occupano di instradare le informazioni.

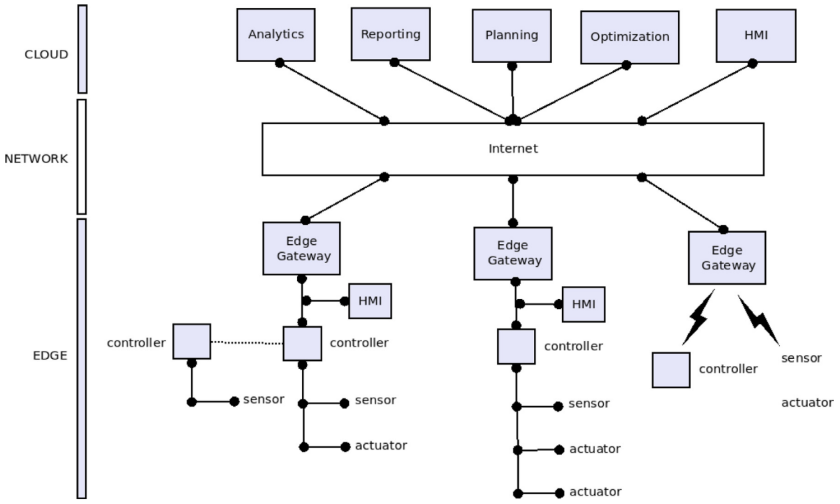

Figure 2.1: Architettura del mondo IoT

spostare la computazione più vicino possibile ai nodi della rete, in modo da ridurre la quantità di dati da inviare e la latenza delle comunicazioni. Questo approccio è chiamato *Edge Computing* non è sempre conveniente applicarlo, dipende dal contesto (non sarebbe adatto nel caso i nodi abbiamo specifiche tecniche, come memoria e capacità di calcolo, deboli).

Nella progettazione di sistemi per il controllo di sistemi aggregati[3] si sottolinea la necessità di meccanismi che rendano possibile ed intuitivo il riutilizzo e la capacità di composizione dei componenti così da poter costruire API[4] a più strati.

Ci sono diversi esempi di sistemi aggregati che necessitano di un controllo distribuito e coordinato, un esempio è il controllo di droni in formazione. In questo caso, ogni drone deve essere in grado di comunicare con i propri vicini per mantenere la formazione, evitare collisioni e raggiungere l'obiettivo comunefig. 2.2. Un altro esempio è il controllo di un gruppo di robot che devono esplorare un ambiente sconosciuto con lo scopo di raccogliere informazioni.

Per andare in contro a queste necessità nasce il paradigma dell'AP che da la nascita al concetto di *intelligenza collettiva*.

[3]Sistemi caratterizzati da un insieme di dispositivi che collaborano tra di loro per raggiungere un obiettivo comune.

[4]Application Programming Interface

Figure 2.2: Droni in formazione in ambiente notturno

2.1.3 Stato dell'arte

Nei capitoli precedenti si è esaminata l'evoluzione dal paradigma Objective Oriented Programming (OOP) a quello FP, la quale, ha permesso di gestire in modo più pratico progetti più complessi e semplificandone la manutenibilità. Una ulteriore, più specifica, evoluzione è quella portata dal paradigma dell'AP. Quest'ultimo mira a rendere la progettazione, manutenzione e testing nell'ambito del controllo di dispositivi hardware di larga-scala (anche conosciuti come **collective adaptive systems (CAS)**).

L'Aggregate Computing (AC) si basa sulla mappatura di un mondo cyber-fisico in un diverso modello logico. Da un punto di vista strutturale possiamo vedere un sistema aggregato come una rete di dispositivi capaci di computare ed equipaggiati con (o sui quali è possibile equipaggiare) sensori ed attuatori, che corrispondono ai nodi. I nodi sono connessi tra loro secondo una logica di vicinato, che può essere definita in base alla distanza fisica o alla comunicazione. Da un punto di vista comportamentale, invece, ogni nodo interpreta il programma aggregato solo in relazione al proprio contesto locale. Da un punto di vista delle interazioni, i nodi estraggono informazione dal proprio vicinato e propagano le proprie nello stesso contesto. Queste interazioni son ciò che permettono al contesto locale e globale di influenzarsi reciprocamente.

La tecnica dell'AP si basa su 3 principi fondamentali per la costruzione di sistemi robusti e resilienti:

- I dettagli implementativi dei sistemi hardware che si vuole andare a manipolare devono essere nascosti così da permettere ai programmatori di concentrarsi solo sulla logica di alto livello del sistema, in alcuni casi è possibile che questa astrazione delle specifiche di basso livello sia tale da poter essere pensato come uno spazio continuo, invece di un insieme di dispositivi separati. Per esempio, invece di immaginarci una stanza piena di sensori, la trattiamo come una area unificata definita da un flusso continuo di dati.

- Il programma manipola le strutture dati della rete di dispositivi sia in funzione della loro estensione spaziale che di quella temporale. Approccio particolare utile in sistemi in cui l'informazione cambia in base al luogo.

- Ogni dispositivo della rete esegue le operazioni necessarie in autonomia, coordinandosi con i dispositivi vicini attraverso meccanismi robusti e resilienti. In questo modo il sistema rimane fluido ed efficiente anche in situazioni anomale, ad esempio causate dal guasto di un qualche dispositivo.

L'approccio AP nella progettazione di sistemi è profondamente diverso classico approccio "dispositivo centrico". Nel secondo caso, ogni dispositivo della rete ha il compito di compiere tutte le operazioni richieste per il raggiungimento della soluzione e simultaneamente comunicare con gli altri dispositivi della rete. Questo approccio, seppur semplice, è poco scalabile e difficile da mantenere al crescere della complessità del sistema. Il nuovo paradigma, invece, prevede la scomposizione in componenti del programma assegnato ad ogni dispositivo, ogni componente esegue un compito specifico e comunica principalmente solo con i componenti dello stesso tipo presenti nel suo intorno per eseguire un servizio ed eventualmente con un altro componente presente nello stesso dispositivo al fine di realizzare una nuova soluzione inerente ad un'altro servizio. Le componenti sono una astrazioni dei *campi di calcolo*.

Il paradigma AP, il quale ha permesso la creazione di framework come ScaFi, Protelis e MacroSwarm, si basa sul concetto dei sopracitati *campi di calcolo*. Per *campo di calcolo* o *computationals field* si intende la mappatura degli elementi del dominio in cui si opera in un insieme di valori. I campi di calcolo vengono utilizzati per standardizzare le interazioni tra i dispositivi, permettendo di analizzare

e progettare sistemi distribuiti in modo più semplice e intuitivo. L'idea di campo prende ispirazione dai campi fisici, ad esempio quello magnetico. Ogni dispositivo della rete è considerato come un punto nello spazio ed il campo rappresenta un dato valore assegnato ad ognuno di questi punti. L'insieme di tutti i valori, che chiameremo *campi* o *fields* rappresentano lo stato di un sistema in un dato istante.

Questo approccio è generalmente utilizzato per un controllo prolungato dei dispositivi della rete sulla quale si vogliono eseguire task che necessitano di eseguire un gran numero di rilevazioni (ad esempio utilizzando sensori), computazioni ed attuazioni.

Ogni dispositivo del proprio sistema aggregato esegue una computazione in modo asincrono attraverso i **sense-compute-(inter)act rounds**, ogni round è composto da tre fasi concettuali:

1. **Aggiornamento del contesto (sense)**: ogni nodo memorizza il suo stato precedente, i dati ambientali restituiti da eventuali sensori e le più recenti informazioni ricevute dai nodi vicini.

2. **Esecuzione del Programma Aggregato (compute)**: il campo di ogni nodo viene computato in base al contesto locale e produce un valore di output detto *export* da condividere con il proprio vicinato ed un output utile per l'attuazione.

3. **Azione (inter-act)**: in questa fase il nodo effettua le seguenti azioni:

   (a) **Condivisione**: il nodo condivide il proprio *export* verso i nodi del suo vicinato in formato broadcast

   (b) **Controllo di attuatori**: l'output del nodo viene utilizzato per l'esecuzione di uno o più attuatori nel proprio ambiente

Posizionandoci in un livello più implementativo troviamo i seguenti costrutti per la manipolazione dei campi:

- **Functions**: funzioni

  $$b(e_1, \ldots, e_n)$$

  applicate agli argomenti $e_1, \ldots, e_n$. Possono essere sia funzioni matematiche, logiche o algoritmiche ma possono anche rappresentare sensori o attuatori.

- **Dynamics**:

$$rep(x \leftarrow v)s_1; \ldots; s_n$$

  rappresenta un una variabile di stato locale x, inizialmente inizializzata con il valore v e che può essere modificata dalle istruzioni $s_1, \ldots, s_n$. In questo modo si definisce un campo dinamico.

- **Interaction**:

$$nbr(s)$$

  rappresenta il vicinato di un dispositivo, ovvero l'insieme dei dispositivi con cui è possibile interagire.

- **Restrictions**:

$$\begin{array}{l} \text{if } e \\ \qquad s_1; \quad \ldots \quad s_n; \\ \text{else} \\ \qquad s'_1; \quad \ldots \quad s'_m; \end{array}$$

  permette di andare a definire sotto spazio del campo principiale in base ad una condizione e sul quale poi andare ad eseguire certe istruzioni invece di altre. È importante che le istruzioni riferite ad un certo sotto spazio non abbiano effetti su altri sotto spazi.

Come viene eseguito un modello di AP?

Questo paradigma, proprio come quelli precedenti, non è privo di difetti. Implementazioni di questo tipo sono difficili da integrare in sistemi programmati in modo più tradizionale, inoltre, dal punto di vista della programmazione sono mancanti i meccanismi per la gestione della concorrenza e della computazione dinamica dei campi. Il framework ScaFi, che vedremo nel prossimo capitolo, cerca di risolvere questi problemi.

2.2 ScaFi

ScaFi è un toolkit open-source basato su Scala per l'AC per la progettazione ad alto livello di sistemi aggregati. Il termine ScaFi deriva dalla unione di Scala e

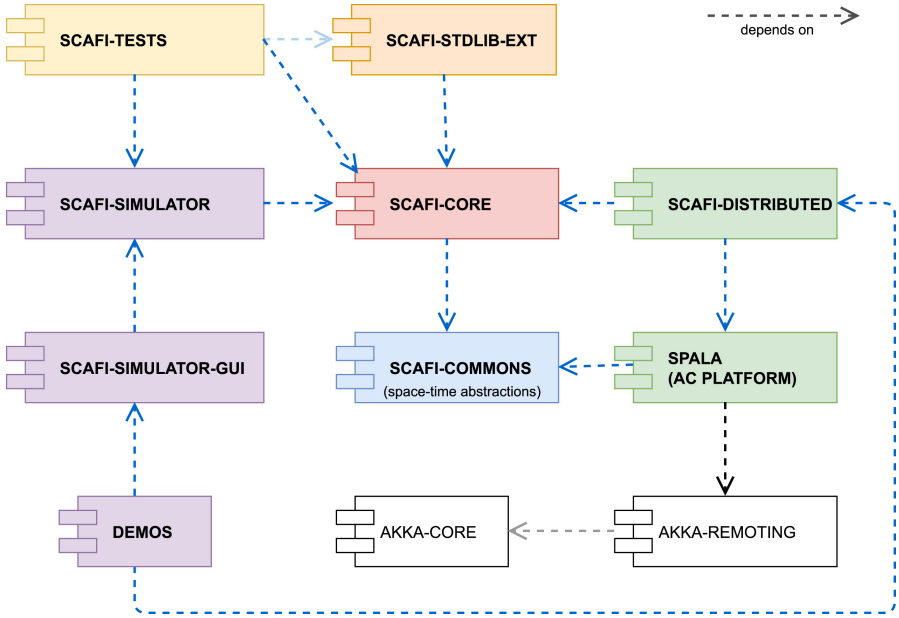

Figure 2.3: Architettura del framework ScaFi

Fields. Scala è un linguaggio java-like moderno e flessibile progettato per unificare il paradigma OOP e quello FP. È stato progettato per essere coinciso e facilmente leggibile, inoltre si basa sulla Java Virtual Machine, la quale permette l'utilizzo delle librerie Java. L'utilizzo della JVM e la flessibilità del linguaggio sono state le caratteristiche chiave per il raggiungimento della popolarità di questo linguaggio. Il termine Field invece deriva dai *field calculus* (campi di calcolo) di cui abbiamo parlato nei capitoli precedenti.

L'Architettura di ScaFi è riassumibile nella immagine fig. 2.3.

Ispezionando la figura, troviamo:

- **scafi-commons**: fornisce le entità di base, ad esempio astrazioni temporali e spaziali
- **scafi-core**: rappresenta il core del framework fornendo il DSL[5] per la progettazione di sistemi aggregati; con il supporto di librerie standard.

[5]Domain Specific Language: linguaggio di programmazione progettato specificamente per risolvere problemi o esprimere concetti di un dominio ristretto. Il DSL di ScaFi (libreria di programmazione) è quindi un linguaggio specifico per creare e valutare programmi di tipo aggregato, utilizzando una sintassi e una semantica pensate appositamente per questo tipo di calcoli e simulazioni.

- **scafi-stdlib-ext**: librerie extra
- **scafi-simulator**: fornisce un supporto per la simulazione dei sistemi di AP sviluppati
- **scafi-simulator-GUI**: fornisce un'interfaccia grafica per la simulazione
- **spala**: ("spatial scala") si tratta di un *middleware* che permette di eseguire programmi AC basati sugli attori ed è indipendente dal DSL di ScaFi
- **scafi-distributed**: layer integrativo per l'utilizzo di **spala** in ScaFi

Perché usare Scala?

Il motivo per cui Scala è un linguaggio particolarmente adatto per la programmazione di sistemi di AC è dovuto proprio alla sua flessibilità, essa permette di lavorare sui campi di calcolo, che necessitano di essere supportati dalle seguenti proprietà, in modo molto naturale:

- una sintassi coincisa per definire funzioni sui campi, le quali in Scala sono le espressioni standard
- meccanismi di controllo sul *quando* e *come* le espressioni vengono valutate
- capacità di calcolare lo stato di un campo basandosi sulle informazioni più recenti dei nodi presenti nel suo intorno

In Scala ogni valore è un oggetto ed i comportamenti vengono definiti attraverso i metodi. Per implementare un campo di calcolo in Scala andremo a definire gli operato dei campi attraverso la definizione di metodi che verranno interpretati dagli oggetti sui quali sono stati chiamati. Gli oggetti, in questo modo, possono essere visti come macchine virtuali in locale per l'esecuzione degli operatori di campo.

Di seguito l'interfaccia fondamentale per la definizione di un campo di calcolo in Scala (listing 2.3):

Gli elementi principali di questa interfaccia sono:

Listing 2.3: Interfaccia per la definizione di un campo di calcolo in Scala

```
trait Constructs {
  def rep[A](init: => A)(fun: A => A): A
  def nbr[A](expr: => A): A
  def foldhood[A](init: => A)(acc: (A, A) => A)(expr: => A): A
  def aggregate[A](f: => A): A

  def branch[A](cond: => Boolean)(th: => A)(el: => A)
  def share[A](init: => A)(fun: (A, () => A) => A): A

  def mid: ID
  def sense[A](sensorName: String): A
  def nbrvar[A](name: CNAME): A
}
```

- `rep(init)(f)` cattura l'evoluzione di stato di un valore inizializzato ad `init` e aggiornato ad ogni round con la funzione `f`
- `nbr(expr)`: rileva le comunicazioni relative al valore computato dalla espressione `expr` dei nodi vicini
- `foldhood(init)(acc)(expr)`: si occupa, partendo da un valore iniziale `init`, di accumulare i valori computati dai nodi vicini attraverso la funzione di accumulazione `acc` e settare i valori computati dalla espressione `expr` verso i propri vicini
- `branch(cond)(th)(el)`: cattura una partizione (spazio-temporale) del dominio in base alla condizione `cond`, se questa è rispettata esegue l'espressione `th` altrimenti `el`
- `mid`: provvede ad identificare un certo nodo
- `sense(sensorName)`: permette di accedere (ad alto livello) al sensore locale `sensorName`
- `nbrvar(sensorName)`: permette di accedere (ad alto livello) ai sensori dei nodi vicini, simile ad `nbr` ma valido per sensori forniti dalla piattaforma, questi sensori forniscono un valore per ciascun vicino

In ScaFi, i campi non sono reificati esplicitamente ma esistono solo a livello semantico, questo significa che un'espressione Scala non viene gestita come un'espressione di campo finché non viene passata all'interprete ScaFi.

ScaFi fornisce alcuni operatori, detti anche *blocchi*, di alto livello già implementati tipici dei sistemi aggregati:

- **Sparse choice (definizione di un leader)**: Blocco

  ```
  S(grain: Double):Boolean
  ```

  produce un campo booleano auto organizzato che definisce a `true` un insieme sparso di dispositivi distanziati tra di loro di almeno `grain`

- **Gradient-cast (propagazione distribuita)**: Blocco

  ```
  G[T](source: Boolean, value: T, acc: T=>T): T
  ```

  utilizzato per propagare un valore `value` da una sorgente `source` a tutti i nodi della rete, in ordine di distanza crescente, seguendo un gradiente che modifica i valori di ogni nodo secondo una funzione di accumulo `acc`

- **Collect-cast (collezione distribuita)**: Blocco

  ```
  C[T](sink: Boolean, value: T, acc(T,T)=>T): T
  ```

  utilizzata per riassumere le informazioni distribuite in un unico dispositivo `sink`, i valori `values` prodotti dai dispositivi vengono "raccolti" dal gradiente che si muove in direzione di `sink` e vengono accumulati secondo la funzione di accumulo `acc`

2.3 Macroswarm

Macroswarm è un framework per la programmazione di sistemi di robotica distribuita basato su ScaFi e che estende lo stesso ScaFi. Macroswarm nasce con l'avanzare delle tecnologie ed il crescente interesse verso il controllo di flotte di dispositivi come droni, robot, veicoli o folle di persone definite da dispositivi portatili/indossabili in modo. Si è voluto, quindi, creare un framework che permettesse di programmare sistemi distribuiti di robot in modo semplice ed intuitivo

secondo degli *swarm behavior*[6], permettendo di concentrarsi solo sulla logica di alto livello del sistema. Questo framework è stato scelto per il progetto in esamina. Il paradigma AP e di conseguenza ScaFi sono state scelte naturali per la progettazione di tale framework (fig. 2.4).

Fondamentalmente, Macroswarm è un programma ScaFi il quale esegue un certo programma aggregato su una certa piattaforma. L'idea è quella di creare una rappresentazione astratta di una flotta di dispositivi attraverso campi di calcolo, tipicamente vettori. All'intero del Framework troviamo le API per progettazione dei sistemi aggregati su sistemi fisici (i rettangoli bianchi in fig. 2.5 rappresentano i moduli principali).

È rilevante notare che in Macroswarm la computazione della attuazione da eseguire su un certo dispositivo e la reale attuazione nel mondo reale sono disaccoppiate. Il motivo di questa scelta è dovuto al fatto che ad ogni roundsection 2.1.3 il programma aggregato può variare un qualche parametro per completare il task. È, comunque, possibile scegliere quale modalità di computazione-attuazione scegliere poiché, in base al contesto, una può essere più adatta dell'altra. Le modalità sono:

- **round-based**fig. 2.6: è possibile eseguire l'attuazione al termine del prossimo round
- **long-standing**fig. 2.7: l'attuazione che si vuole eseguire è valida finché non viene revisionata o ritirata

Questo modello di attuazione è modellato come una funzione purasection 2.1.1 quindi priva di effetti collaterali. Se ci si trova in un contesto in cui l'esecuzione dell'attuazione richiede un certo tempo allora il progettista del sistema aggregato può andare a modellare il sistema in modo tale da tenere in considerazione questo aspetto.

2.4 Thymio e tdmclient

> "Thymio è un robot educativo open-source progettato da ricercatori dell'EPFL (Politecnico federale di Losanna), in collaborazione con

[6]Comportamenti di sciame/flotta

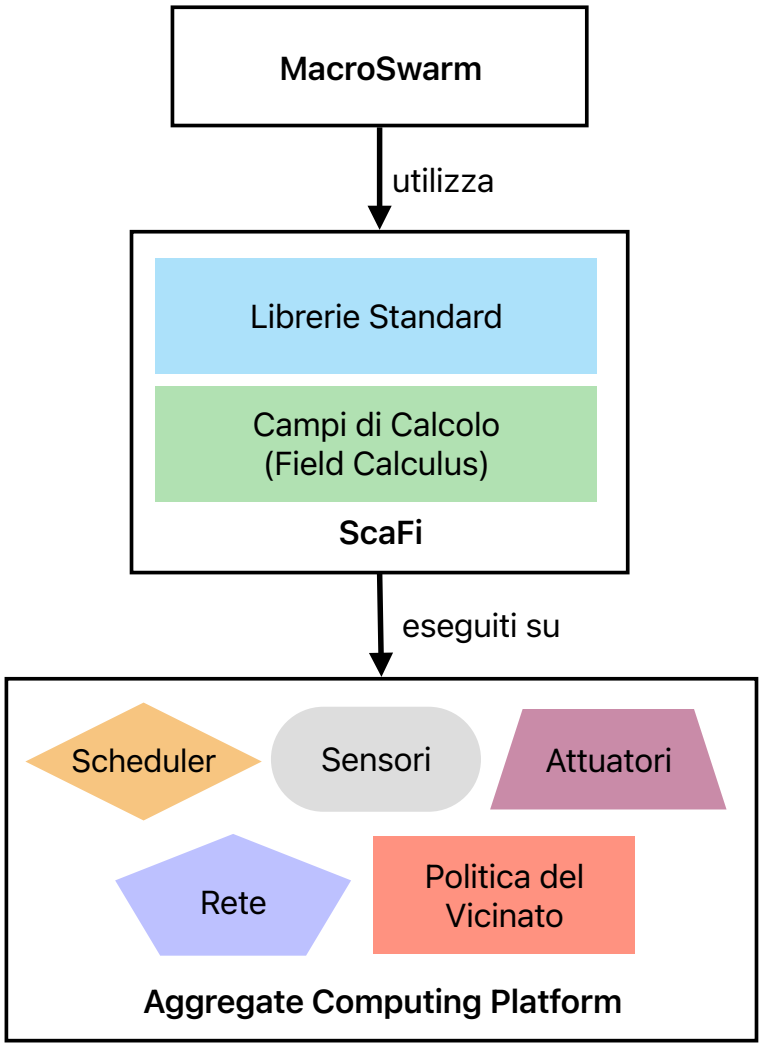

Figure 2.4: Architettura di Macroswarm

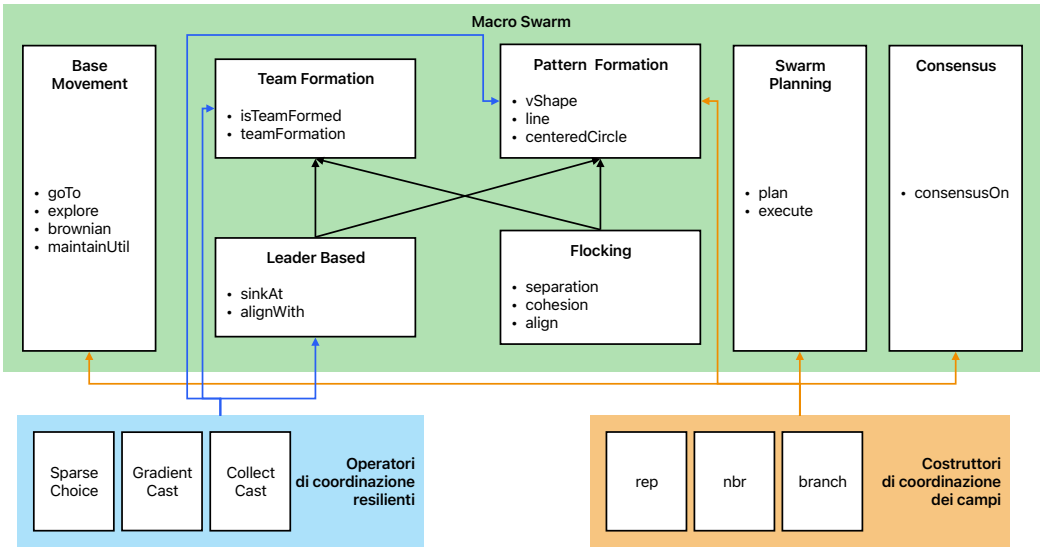

Figure 2.5: Struttura di Macroswarm e moduli interni

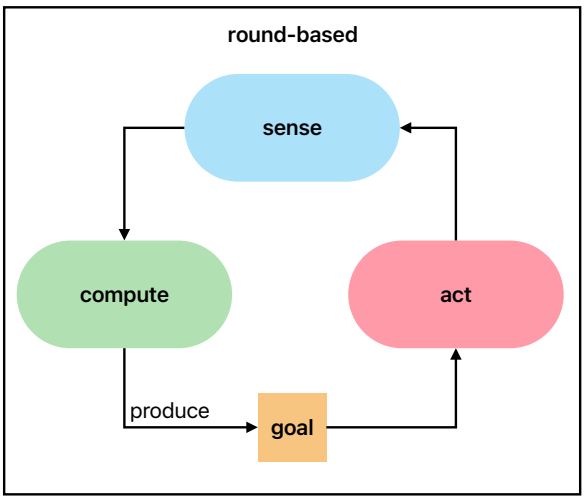

Figure 2.6: Modalità Round Based

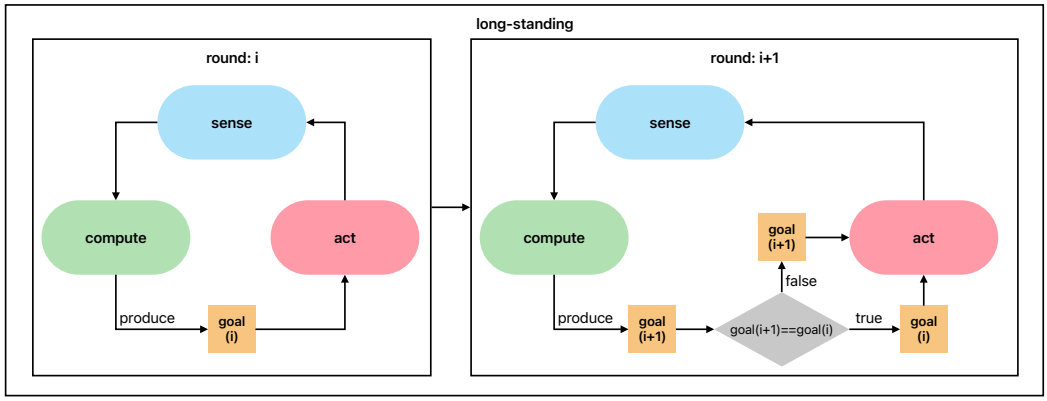

Figure 2.7: Modalità Long Standing

> l'ECAL (Università d'arte e design di Losanna), e prodotto da Mobsya, un'associazione no-profit la cui missione è quella di offrire percorsi STEAM completi e coinvolgenti a studenti di tutte le età."
>
> – *Mobsya*

Thymio è un robot programmabile con un'ampia varietà di sensori e attuatori, tra cui:

- 9 sensori a infrarossi (portata circa 10 cm)
- 5 pulsanti a sfioramento (tecnologia capacitiva)
- 1 accelerometro a tre assi
- 1 termometro
- 1 microfono
- 1 ricevitore a infrarossi per il telecomando
- 39 LED controllabili
- 2 motori DC collegati alle ruote
- 1 altoparlante

Il deploy del codice sul robot Thymio avviene tramite un radio dongle USB.

Figure 2.8: Robot Thymio

Thymio Network : La connessione wireless dei robot è basata sul protocollo 802.15.4 in banda con frequenza 2.4 GHz. Questo protocollo permette di gestire una rete con molti nodi a discapito del rispetto delle seguenti condizioni:

- tutti i nodi devono trovarsi tutti nello stesso canale radio (sono disponibili 3 reti: [0, 1, 2])
- tutti i dispositivi devono avere lo stesso Identificativo di rete (PAN ID)
- tutti devono avere un nodo Identificativo univo (nodeID)

Questo protocollo è stato scelto per la sua bassa potenza e la sua affidabilità. Il protocollo è stato implementato in modo da permettere la comunicazione tra i robot e con il computer tramite il dongle USB.

Il dispositivo è stato scelto per essere implementato nella demo per la notte dei ricercatori in quanto è un robot molto versatile e adatto a molteplici applicazioni. Inoltre, il robot è molto diffuso nelle scuole e nei laboratori di robotica educativa, quindi è un'ottima scelta per mostrare il potenziale di ScaFi in un contesto educativo.

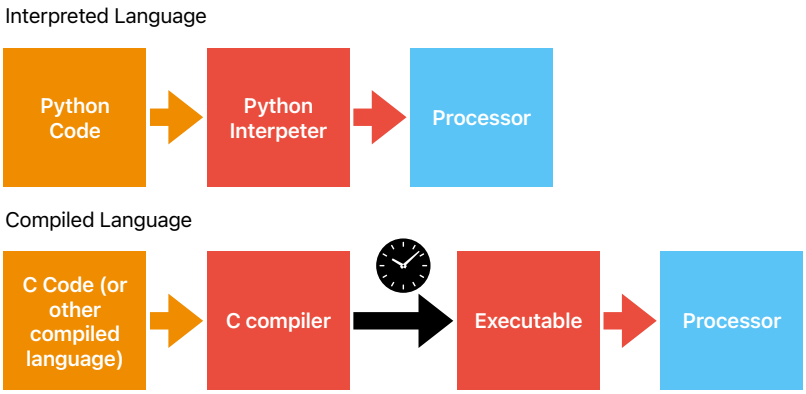

Figure 2.9: Interprete vs Compilatore

Questo piccolo robot nasce per essere programmato con linguaggi a blocchi come VPL, VPL3, Scratch, Blocky e non come Aseba, Python e Ros, tutti accessibili dalla suite dedicata *"Thymio Suite"*. Noi ci concentreremo nel uso di Python.

Python a differenza di altri linguaggi (ad esempio C/C++) è interpretato e non compilato. I linguaggi compilati prevedono la conversione del programma scritto in un linguaggio ad alto livello in linguaggio macchina per poi farlo eseguire dal processore, occorre quindi compilare il sorgente ad ogni nuova modifica per poi re-eseguirlo. Python, invece, è un linguaggio interpretato, il codice sorgente viene eseguito direttamente dal programma interprete, il quale esegue ogni comando riga per riga.

Nel caso del robot Thymio, non è presente un interprete nel suo microcontrollore. È presente, invece, una Aseba Virtual Machine che permette di eseguire programmi scritti in linguaggio Aseba.

Aseba è un linguaggio di programmazione ad alto livello basato su architettura ad eventi, il che significa che gli eventi sono eseguiti in modo asincrono. Gli eventi sono identificati da un Identificativo ed opzionalmente da un *payload* (dati aggiuntivi). Gli eventi possono essere di due tipi:

- **global events**: eventi generati dai nodi e condivisi con la rete

- **local events**: eventi generati da un nodo e non condivisi con la rete (ad esempio un evento generato da un sensore dello stesso nodo)

Un esempio di codice Aseba è il seguente listing 2.4.

Listing 2.4: Esempio di codice Aseba con eventi

```
var state

callsub init  # Inizializza il programma

sub init
    state = 0
    call leds.bottom.left(0,0,32)
    call leds.bottom.right(0,32,0)
    call leds.top(32,0,0)

# Re-inizia quando il pulsante centrale viene premuto
onevent button.center
    callsub init
```

Il **Thymio Device Manager** è una delle feature della Thymio Suite, si occupa di gestire la rete dei Thymio. Inoltre ha il compito di tradurre il sorgente, scritto dall'utente, da linguaggio Aseba ad Aseba bytecode per poi eseguire il *deploy* sul robot. Per l'utilizzo di Python, invece, è necessario l'utilizzo del modulo **tdmclient** che si occupa di far comunicare Python con il Thymio Device Manager (TDM) fig. 2.10. È necessario che la Thymio Suite sia in esecuzione per poter utilizzare il modulo **tdmclient**, il quale, a sua volta, necessita dell'utilizzo di **Python 3** come interprete.

Questo modulo permette di:

- accedere alle variabili, sensori, attuatori del robot
- convertire uno script Python in un script Aseba (transpiler)
- inviare codice Aseba al TDM che a sua volta invia il relativo bytecode al robot Thymio

Le possibilità offerte da questo modulo sono molteplici, ad esempio è possibile inviare comandi direttamente ad un robot Thymio da terminale:

Listing 2.5: Esempio di invio di comando (foo) ad un robot Thymio

```
python3 -m tdmclient sendevent --event foo --data 123
```

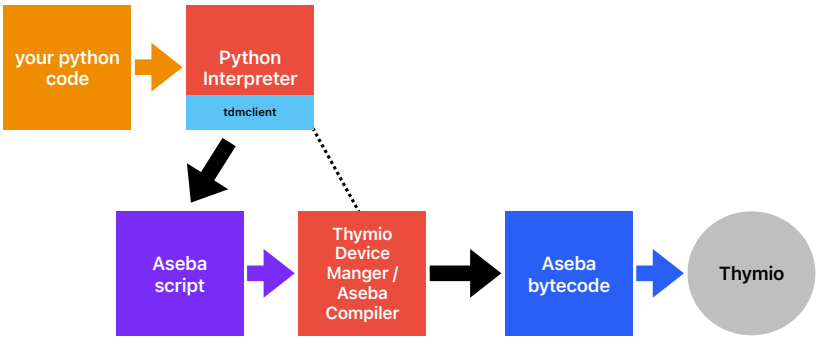

Figure 2.10: Tdmclient workflow

In listing 2.5 si assume che il robot accetti un evento di nome *foo* con un payload di tipo intero e che il robot sia in ascolto di eventi (*unlocked*). Per metter in ascolto un robot di un certo evento è necessario andare ad inserire una vista su quel determinato robot dalla Thymio Suite.

Listing 2.6: Esempio di programma interno a Thymio per la ricezione di eventi (foo)

```
var x

onevent foo
    x = event.args[0]
```

Il modulo permette di eseguire il transpile (conversione) di un programma Python in Aseba. Per farlo è necessario utilizzare il comando *transpile* del modulo **tdmclient**:

Listing 2.7: Esempio di transpile di un programma Python (print.py) in Aseba

```
python3 -m tdmclient transpile examples/print.py
```

Utilizzando il modulo aggiuntivo `ClientAsync` è possibile modificare le variabili dei robot Thymio attraverso chiamate asincrone in uno script **Python**. Si andrà ad esaminare ulteriormente questa feature di **tdmclient** nel capitolo chapter 4 e chapter 5 poiché è ciò che si è utilizzato per la progettazione del nostro sistema.

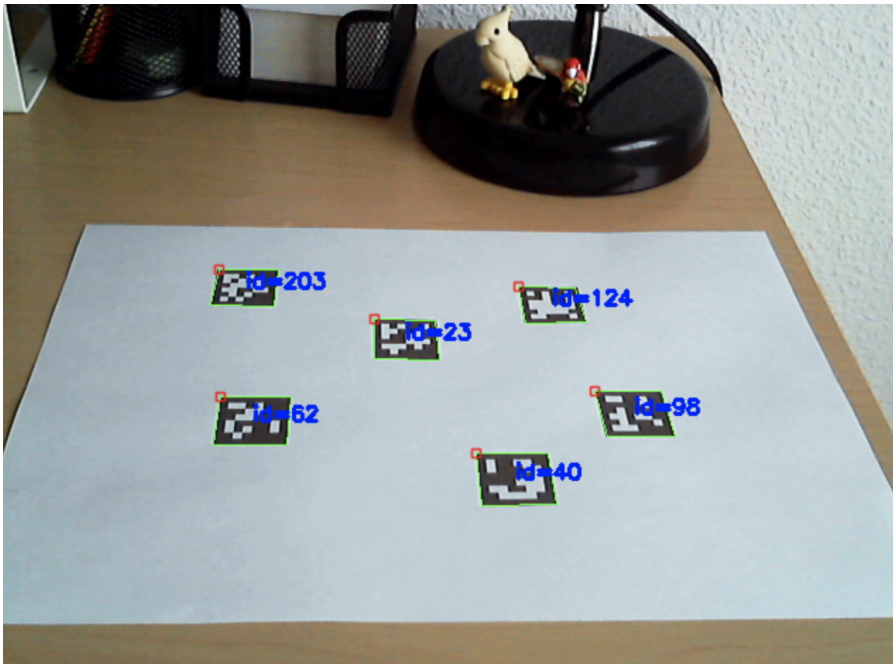

Figure 2.11: Aruco markers identificati da un algoritmo di visione artificiale

2.5 Aruco Tag

Gli Aruco marker sono tag quadrati che possono essere rilevati da una camera. Ogni tag è una matrice 4×4 ed ha un codice (intero) univoco associato che può essere rilevato da un algoritmo di visione artificialefig. 2.11. È rilevante notare che la matrice di ogni Aruco marker corrisponde ad uno stesso codice per ogni sua versione trasposta. Questi tag sono molto utili per la localizzazione e la navigazione di robot autonomi. Sarà questo strumento che verrà utilizzato per la demo della notte dei ricercatori per la localizzazione e calcolo della direzione dei robot Thymio.

Questo strumento è già presente nel sistema in questione.

Chapter 3

Analisi

3.1 Estensibilità del sistema ad un nuovo modello di Robot (Thymio)

3.2 Gestione dei vincoli di compatibilità del sistema Thymio

Chapter 4

Design

4.1 Architettura server Flask

4.2 File di configurazione

Chapter 5

Implementazione

5.1 Implementazione del server Flask

5.2 Esempi di Algoritmi AP applicati ai Robot Wave e Thymio nello stesso ambiente

You may also put some code snippet (which is NOT float by default), eg: section 5.2.

5.3 Fancy formulas here

```
1 public class HelloWorld {
2   public static void main(String[] args) {
3     // Prints "Hello, World" to the terminal window.
4     System.out.println("Hello, World");
5   }
6 }
```

Chapter 6

Conclusione

Bibliography

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and computer modelling*, 17(7):57–73, 1993.

[ACO21] Andrea Agiollo, Giovanni Ciatto, and Andrea Omicini. *Shallow2Deep*: Restraining neural networks opacity through neural architecture search. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *Lecture Notes in Computer Science*, pages 63–82. Springer Nature, Basel, Switzerland, 2021.

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowl. Based Syst.*, 8(6):373–389, 1995.

[AFWZ02] Alessandro Artale, Enrico Franconi, Frank Wolter, and Michael Zakharyaschev. A temporal description logic for reasoning over conceptual schemas and queries. In *European Workshop on Logics in Artificial Intelligence (JELIA 2002)*, pages 98–110. Springer, 2002.

[AK12a] M. Gethsiyal Augasta and T. Kathirvalavakumar. Reverse engineering the neural networks for rule extraction in classification problems. *Neural Processing Letters*, 35(2):131–150, April 2012.

[AK12b] M. Gethsiyal Augasta and T. Kathirvalavakumar. Reverse engineering the neural networks for rule extraction in classification problems. *Neural Process. Lett.*, 35(2):131–150, 2012.

[Apt90] Krzysztof R Apt. Logic programming. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990:493–574, 1990.

[Baa03] Franz Baader. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95, USA, 2003. Cambridge University Press.

[BDKT97] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial intelligence*, 93(1–2):63–101, 1997.

[BFOS84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[BH16] Guido Bologna and Yoichi Hayashi. A rule extraction study on a neural network trained by deep learning. In *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*, pages 668–675. IEEE, 2016.

[BH18] Guido Bologna and Yoichi Hayashi. A comparison study on rule extraction from neural network ensembles, boosted shallow trees, and svms. *Appl. Comput. Intell. Soft Comput.*, 2018:4084850:1–4084850:20, 2018.

[BHS79] A. E. Bryson, Y. Ho, and G. M. Siouris. Applied optimal control: Optimization, estimation, and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(6):366–367, June 1979.

[BKB17] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting blackbox models via model extraction. *CoRR*, abs/1705.08504, 2017.

[BL04] Ronald J. Brachman and Hector J. Levesque. The tradeoff between expressiveness and tractability. In Ronald J. Brachman and Hector J. Levesque, editors, *Knowledge Representation and Reasoning*, The Morgan Kaufmann Series in Artificial Intelligence, pages 327–348. Morgan Kaufmann, San Francisco, 2004.

[BU18] Tarek R. Besold and Sara L. Uckelman. The what, the why, and the how of artificial explanations in automated decision-making. *CoRR*, abs/1808.07074:1–20, 2018.

[CBMO19] Giovanni Ciatto, Michael Bosello, Stefano Mariani, and Andrea Omicini. Comparative analysis of blockchain technologies under a coordination perspective. In Fernando De La Prieta, Alfonso González-Briones, Pawel Pawleski, Davide Calvaresi, Elena Del Val, Fernando Lopes, Vicente Julian, Eneko Osaba, and Ramón Sánchez-Iborra, editors, *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection*, volume 1047 of *Communications in Computer and Information Science*, chapter 7, pages 80–91. Springer, June 2019.

[CCDMSO20] Ashley Caselli, Giovanni Ciatto, Giovanna Di Marzo Serugendo, and Andrea Omicini. Engineering semantic self-composition of services through tuple-based coordination. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*, volume 12477 of *Lecture Notes in Computer Science*, pages 205–223. Springer International Publishing, Cham, 2020.

[CCDO19a] Roberta Calegari, Giovanni Ciatto, Jason Dellaluce, and Andrea Omicini. Interpretable narrative explanation for ML predictors with LP: A case study for XAI. In Federico Bergenti and Stefania Monica, editors, *WOA 2019 – 20th Workshop "From Objects to Agents"*, volume 2404 of *CEUR Workshop Proceedings*, pages 105–112. Sun SITE Central Europe, RWTH Aachen University, 26–28 June 2019.

[CCDO19b] Roberta Calegari, Giovanni Ciatto, Enrico Denti, and Andrea Omicini. Engineering micro-intelligence at the edge of CPCS: Design guidelines. In *Internet and Distributed Computing Systems (IDCS 2019)*, volume 11874 of *Lecture Notes in Computer Science*, pages 260–270. Springer, 10–12 October 2019.

[CCDO20] Roberta Calegari, Giovanni Ciatto, Enrico Denti, and Andrea Omicini. Logic-based technologies for intelligent systems: State of the art and perspectives. *Information*, 11(3):1–29, March 2020. Special Issue "10th Anniversary of Information—Emerging Research Challenges".

[CCM+18a] Roberta Calegari, Giovanni Ciatto, Stefano Mariani, Enrico Denti, and Andrea Omicini. Logic programming in space-time: The case of situatedness in LPaaS. In Massimo Cossentino, Luca Sabatucci, and Valeria Seidita, editors, *WOA 2018 – 19th Workshop "From Objects to Agents"*, volume 2215 of *CEUR Workshop Proceedings*, pages 63–68. Sun SITE Central Europe, RWTH Aachen University, 29–30 June 2018.

[CCM+18b] Roberta Calegari, Giovanni Ciatto, Stefano Mariani, Enrico Denti, and Andrea Omicini. LPaaS as micro-intelligence: Enhancing IoT with symbolic reasoning. *Big Data and Cognitive Computing*, 2(3), 2018.

[CCM+18c] Roberta Calegari, Giovanni Ciatto, Stefano Mariani, Enrico Denti, and Andrea Omicini. Micro-intelligence for the IoT: SE challenges and practice in LPaaS. In *2018 IEEE International Conference on Cloud Engineering (IC2E 2018)*, pages 292–297. IEEE Computer Society, 17–20 April 2018.

[CCM+18d] Giovanni Ciatto, Roberta Calegari, Stefano Mariani, Enrico Denti, and Andrea Omicini. From the blockchain to logic programming and back: Research perspectives. In Massimo Cossentino, Luca Sabatucci, and Valeria Seidita, editors, *WOA 2018 – 19th Work-*

shop "From Objects to Agents", volume 2215 of *CEUR Workshop Proceedings*, pages 69–74. Sun SITE Central Europe, RWTH Aachen University, June 2018.

[CCMO21a] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. Logic-based technologies for multi-agent systems: A systematic literature review. *Autonomous Agents and Multi-Agent Systems*, 35(1):1:1–1:67, 2021. Collection "Current Trends in Research on Software Agents and Agent-Based Software Development".

[CCMO21b] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. Logic-based technologies for multi-agent systems: Summary of a systematic literature review. In *20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2021)*, pages 1721–1723, May 2021.

[CCN+21] Davide Calvaresi, Giovanni Ciatto, Amro Najjar, Reyhan Aydoğan, Leon Van der Torre, Andrea Omicini, and Michael Schumacher. EXPECTATION: Personalized explainable artificial intelligence for decentralized agents with heterogeneous knowledge. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *Lecture Notes in Computer Science*, pages 331–343. Springer Nature, Basel, Switzerland, 2021.

[CCO20] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32, 2020.

[CCO21a] Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. 2p-kt: A logic-based ecosystem for symbolic ai. *SoftwareX*, 2021.

[CCO21b] Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Lazy stream manipulation in Prolog via backtracking: The case of 2P-KT.

In Wolfgang Faber, Gerhard Friedrich, Martin Gebser, and Michael Morak, editors, *Logics in Artificial Intelligence*, volume 12678 of *Lecture Notes in Computer Science*, pages 407–420. Springer, 2021. 17th European Conference, JELIA 2021, Virtual Event, May 17–20, 2021, Proceedings.

[CCS+20] Giovanni Ciatto, Roberta Calegari, Enrico Siboni, Enrico Denti, and Andrea Omicini. 2P-Kt: logic programming with objects & functions in kotlin. In Roberta Calegari, Giovanni Ciatto, Enrico Denti, Andrea Omicini, and Giovanni Sartor, editors, *WOA 2020 – 21th Workshop "From Objects to Agents"*, volume 2706 of *CEUR Workshop Proceedings*, pages 219–236, Aachen, Germany, October 2020. Sun SITE Central Europe, RWTH Aachen University. 21st Workshop "From Objects to Agents" (WOA 2020), Bologna, Italy, 14–16 September 2020. Proceedings.

[CCSO20] Giovanni Ciatto, Davide Calvaresi, Michael I. Schumacher, and Andrea Omicini. An abstract framework for agent-based explanations in AI. In *19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1816–1818. International Foundation for Autonomous Agents and Multiagent Systems, May 2020. Extended Abstract.

[CDD17] Giovanni Ciatto, Elisabetta De Maria, and Cinzia Di Giusto. Spiking neural networks as timed automata. In *Proc. of the Thematic Research School on Advances in Systems and Synthetic Biology (ASSB)*, pages 55–69. EDP Sciences, 2017.

[CDMSL+20] Giovanni Ciatto, Giovanna Di Marzo Serugendo, Maxime Louvel, Stefano Mariani, Andrea Omicini, and Franco Zambonelli. Twenty years of coordination technologies: COORDINATION contribution to the state of art. *Journal of Logical and Algebraic Methods in Programming*, 113:1–25, June 2020.

[CH94] William W. Cohen and Haym Hirsh. Learning the classic description logic: Theoretical and experimental results. In *Principles of Knowledge Representation and Reasoning*, pages 121–133. Elsevier, 1994.

[Cim06] Philipp Cimiano. *Ontology Learning and Population from Text*. Springer US, 2006.

[Cla77] Keith L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, pages 293–322, New York, 1977. Plemum Press.

[CMMO19] Giovanni Ciatto, Alfredo Maffi, Stefano Mariani, and Andrea Omicini. Towards agent-oriented blockchains: Autonomous smart contracts. In Yves Demazeau, Eric Matson, Juan Manuel Corchado, and Fernando De la Prieta, editors, *Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection*, volume 11523 of *Lecture Notes in Computer Science*, pages 29–41. Springer International Publishing, June 2019.

[CMMO20a] Giovanni Ciatto, Alfredo Maffi, Stefano Mariani, and Andrea Omicini. Smart contracts are more than objects: Pro-activeness on the blockchain. In Javier Prieto, Ashok Das Kumar, Stefano Ferretti, António Pinto, and Juan Manuel Corchado, editors, *Blockchain and Applications*, volume 1010 of *Advances in Intelligent Systems and Computing*, pages 45–53. Springer, 2020.

[CMMO20b] Giovanni Ciatto, Stefano Mariani, Alfredo Maffi, and Andrea Omicini. Blockchain-based coordination: Assessing the expressive power of smart contracts. *Information*, 11(1):1–20, January 2020. Special Issue "Blockchain Technologies for Multi-Agent Systems".

[CMO17] Giovanni Ciatto, Stefano Mariani, and Andrea Omicini. Programming the interaction space effectively with ReSpecTX. In Mirjana

Ivanović, Costin Bădică, Jürgen Dix, Zoran Jovanović, Michele Malgeri, and Miloš Savić, editors, *Intelligent Distributed Computing XI*, volume 737 of *Studies in Computational Intelligence*, pages 89–101. Springer, 2017.

[CMO18a] Giovanni Ciatto, Stefano Mariani, and Andrea Omicini. Blockchain for trustworthy coordination: A first study with Linda and Ethereum. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 696–703, December 2018.

[CMO18b] Giovanni Ciatto, Stefano Mariani, and Andrea Omicini. ReSpecTX: Programming interaction made easy. *Computer Science and Information Systems*, 15(3):655–682, October 2018. Special Section: Contemporary Topics in Intelligent Distributed Computing.

[CMO+18c] Giovanni Ciatto, Stefano Mariani, Andrea Omicini, Franco Zambonelli, and Maxime Louvel. Twenty years of coordination technologies: State-of-the-art and perspectives. In Giovanna Di Marzo Serugendo and Michele Loreti, editors, *Coordination Models and Languages*, volume 10852 of *Lecture Notes in Computer Science*, pages 51–80. Springer, 2018. 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings.

[CMOZ20] Giovanni Ciatto, Stefano Mariani, Andrea Omicini, and Franco Zambonelli. From agents to blockchain: Stairway to integration. *Applied Sciences*, 10(21):7460:1–7460:22, 2020. Special Issue "Advances in Blockchain Technology and Applications 2020".

[CNCC21] Giovanni Ciatto, Amro Najjar, Jean-Paul Calbimonte, and Davide Calvaresi. Towards explainable visionary agents: License to dare and imagine. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAA-*

MAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers, volume 12688 of *Lecture Notes in Computer Science*, pages 139–157. Springer Nature, Basel, Switzerland, 2021.

[Col86] Alain Colmerauer. Theoretical model of prolog ii. In M. van Canegham and D.-H.D. Warren, editors, *Logic Programming and its applications*, pages 3–31. Ablex Publishing Corporation, 1986.

[CR93] Alain Colmerauer and Philippe Roussel. The birth of prolog. In John A. N. Lee and Jean E. Sammet, editors, *History of Programming Languages Conference (HOPL-II)*, pages 37–52. ACM, April 1993.

[Cra16] Kate Crawford. Artificial intelligence's white guy problem. *The New York Times*, 25, 2016.

[CROM19] Giovanni Ciatto, Lorenzo Rizzato, Andrea Omicini, and Stefano Mariani. TuSoW: Tuple spaces for edge computing. In *The 28th International Conference on Computer Communications and Networks (ICCCN 2019)*, Valencia, Spain, 29 July–1 August 2019.

[CS95] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 24–30. MIT Press, 1995.

[CSOC20] Giovanni Ciatto, Michael I. Schumacher, Andrea Omicini, and Davide Calvaresi. Agent-based explanations in ai: Towards an abstract framework. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, volume 12175 of *Lecture Notes in Computer Science*, pages 3–20. Springer, Cham, 2020. Second International Workshop, EXTRAAMAS 2020, Auckland, New Zealand, May 9–13, 2020, Revised Selected Papers.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[DCB19] Federica Di Castro and Enrico Bertini. Surrogate decision tree visualization. In *Joint Proceedings of the ACM IUI 2019 Workshops (ACMIUI-WS 2019)*, volume 2327 of *CEUR Workshop Proceedings*, March 2019.

[dGBG01] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artif. Intell.*, 125(1-2):155–207, 2001.

[dGBR+15] Artur S. d'Avila Garcez, Tarek R. Besold, Luc De Raedt, Peter Földiák, Pascal Hitzler, Thomas Icard, Kai-Uwe Kühnberger, Luís C. Lamb, Risto Miikkulainen, and Daniel L. Silver. Neural-symbolic learning and reasoning: Contributions and challenges. In *2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015*. AAAI Press, 2015.

[dK15] Luc de Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

[dKL98] Mark d'Inverno, D. Kinney, and Michael Luck. Interaction protocols in agentis. In *Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160)*, pages 112–119. IEEE, 1998.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[DMDGC17] Elisabetta De Maria, Cinzia Di Giusto, and Giovanni Ciatto. Formal validation of neural networks as timed automata. In *Proceedings of the 8th International Conference on Computational Systems-Biology and Bioinformatics*, CSBio '17, pages 15–22, New York, NY, USA, 2017. ACM.

[DRW96] Steven Dawson, C. R. Ramakrishnan, and David S. Warren. Practical program analysis using general purpose logic programming systems—a case study. In *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI '96, pages 117—126, New York, NY, USA, 1996. Association for Computing Machinery.

[DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *CoRR*, abs/1702.08608, 2017.

[Ell19] Anthony Elliott. *The Culture of AI: Everyday Life and the Digital Revolution*. Routledge, 2019.

[FH17a] Marion Fourcade and Kieran Healy. Categories all the way down. *Historical Social Research/Historische Sozialforschung*, pages 286–296, 2017.

[FH17b] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In Tarek R. Besold and Oliver Kutz, editors, *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 16th and 17th, 2017*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[FK97] Tze Ho Fung and Robert Kowalski. The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, 33(2):151–165, 1997.

[FV17] Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. pages 3449–3457, 2017.

[FW99] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GF17] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

[GG12] Sanjeev Goyal and Sandeep Grover. Applying fuzzy grey relational analysis for ranking the advanced manufacturing systems. *Grey Systems: Theory and Application*, 2(2):284–298, 2012.

[GMR+19] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019.

[GR68] C. Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In *1968 23rd ACM National Conference*, pages 169–181, 1968.

[Gun16] David Gunning. Explainable artificial intelligence (XAI). Funding Program DARPA-BAA-16-53, DARPA, 2016.

[Han06] David J Hand. Data mining. *Encyclopedia of Environmetrics*, 2, 2006.

[HE06] Eduardo R. Hruschka and Nelson F.F. Ebecken. Extracting rules from multilayer perceptrons in classification problems: A clustering-based approach. *Neurocomputing*, 70(1-3):384–397, 2006.

[Hel19] Dirk Helbing. Societal, economic, ethical and legal challenges of the digital revolution: From big data to deep learning, artificial intelligence, and manipulative technologies. In *Towards Digital Enlightenment*, pages 47–72. Springer, 2019.

[Hen08] J. Hendler. Avoiding another ai winter. *IEEE Intelligent Systems*, 23(2):2–4, March 2008.

[Hor05] Ian Horrocks. OWL: A description logic based ontology language. In Peter Van Beek, editor, *Principles and Practice of Constraint Programming (CP 2005)*, pages 5–8. Springer, 2005. Extended Abstract.

[HQR17] Robert Hoehndorf and Núria Queralt-Rosinach. Data science and symbolic ai: Synergies, challenges and opportunities. *Data Science*, 2017.

[HRHL01] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Intelligent agents-summary of an agent infrastructure. In *Proceedings of the 5th International conference on autonomous agents*, 2001.

[Hub99] Marcus J. Huber. Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, 1999.

[JL87] Joxan Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, October 1987.

[JN09] Ulf Johansson and Lars Niklasson. Evolving decision trees using oracle guides. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, part of the IEEE Symposium Series on Computational Intelligence 2009, Nashville, TN, USA, March 30, 2009 - April 2, 2009*, pages 238–244. IEEE, 2009.

[KA09] Humar Kahramanli and Novruz Allahverdi. Rule extraction from trained adaptive neural networks using artificial immune systems. *Expert Syst. Appl.*, 36(2):1513–1522, 2009.

[KBB+21] Philipp Körner, Michael Beuschel, João Barbosa, Vítor Santos Costa, Verónica Dahl, Manuel V. Hermenegildo, Jose F. Morales, Jan Wielemaker, Daniel Diaz, Salvador Abreu, and Giovanni

Ciatto. A multi-walk through the past, present and future of prolog. *Theory and Practice of Logic Programming*, 2021.

[Kot07] Sotiris Kotsiantis. Supervised machine learning: A review of classification techniques. In *Emerging Artificial Intelligence Applications in Computer Engineering*, volume 160 of *Frontiers in Artificial Intelligence and Applications*, pages 3–24. IOS Press, October 2007.

[Kow74] Robert A. Kowalski. Predicate logic as programming language. In Jack L. Rosenfeld, editor, *Information Processing, Proceedings of the 6th IFIP Congress*, pages 569–574. North-Holland, August 1974.

[KSB99] R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999–2009, 1999.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Comput. Intell.*, 3:78–93, 1987.

[LD94] Jaeho Lee and Edmund H. Durfee. Structured circuit semantics for reactive plan execution systems. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 2, pages 1232–1237, Seattle, WA, USA, 31 July—4 August 1994. AAAI Press / The MIT Press.

[Lip18] Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018.

[Llo90] John W Lloyd. *Computational logic*. Springer, 1990.

[Md94] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994. Special Issue: Ten Years of Logic Programming.

[MH03] Ralf Moller and Volker Haarslev. *Description logic systems*, pages 282–305. Cambridge University Press, 2003.

[Mil56] George Abram Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, March 1956.

[Min75] Marvin Minsky. A framework for representing knowledge representation. In *The Psychology of Computer Vision*. Mc Graw-Hill, New-York (NY, US), 1975.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, April 1982.

[MOC17] Stefano Mariani, Andrea Omicini, and Giovanni Ciatto. Novel opportunities for tuple-based coordination: XPath, the Blockchain, and stream processing. In Pasquale De Meo, Maria Nadia Postorino, Domenico Rosaci, and Giuseppe M.L. Sarné, editors, *WOA 2017 – 18th Workshop "From Objects to Agents"*, volume 1867 of *CEUR Workshop Proceedings*, pages 61–64. Sun SITE Central Europe, RWTH Aachen University, June 2017.

[MP88] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA, 1988.

[MS58] John Mccarthy and Claude Shannon. Automata studies. *Journal of Symbolic Logic*, 23(1):59–60, 1958.

[MTC+10] Marco Montali, Paolo Torroni, Federico Chesani, Paola Mello, Marco Alberti, and Evelina Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundamenta Informaticae*, 102(3–4):325–361, 2010.

[NM96] Anil Nerode and G. Metakides. *Principles of Logic and Logic Programming*. Elsevier Science Inc., USA, 1996.

[Pau18] Lawrence C. Paulson. Computational logic: its origins and applications. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2210):20170872, 2018.

[PCC+18] Danilo Pianini, Giovanni Ciatto, Roberto Casadei, Stefano Mariani, Mirko Viroli, and Andrea Omicini. Transparent protection of aggregate computations from Byzantine behaviours via blockchain. In *GOODTECHS'18 – Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good*, pages 271–276, New Work, NY, USA, November 2018. ACM.

[PCCO20] Giuseppe Pisano, Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Neuro-symbolic computation for XAI: Towards a unified model. In Roberta Calegari, Giovanni Ciatto, Enrico Denti, Andrea Omicini, and Giovanni Sartor, editors, *WOA 2020 – 21th Workshop "From Objects to Agents"*, volume 2706 of *CEUR Workshop Proceedings*, pages 101–117, Aachen, Germany, October 2020. Sun SITE Central Europe, RWTH Aachen University. 21st Workshop "From Objects to Agents" (WOA 2020), Bologna, Italy, 14–16 September 2020. Proceedings.

[Pol87] John L. Pollock. Defeasible reasoning. *Cognitive science*, 11(4):481–518, 1987.

[PW78] David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, December 1978.

[Rao96] Anand S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In Walter Van de Velde and John W. Perram, editors, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer, Berlin, Heidelberg, 1996.

[Rei80] Raymond Reiter. A logic for default reasoning. *Artificial intelligence*, 13(1–2):81–132, 1980.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[Ros00] Francesca Rossi. Constraint (logic) programming: A survey on research and applications. In Krzysztof R. Apt, Eric Monfroy, Antonis C. Kakas, and Francesca Rossi, editors, *New Trends in Constraints*, pages 40–74. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[RR19] Avi Rosenfeld and Ariella Richardson. Explainability in human-agent systems. *Autonomous Agents and Multi-Agent Systems*, 33(6):673–705, November 2019.

[RSG16] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM, 2016.

[Rud19] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[RVBW08] Francesca Rossi, Peter Van Beek, and Toby Walsh. Constraint programming. *Foundations of Artificial Intelligence*, 3:181–211, 2008.

[SCO21] Federico Sabbatini, Giovanni Ciatto, and Andrea Omicini. GridEx: An algorithm for knowledge extraction from black-box regressors. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers*, volume 12688 of *Lecture Notes in Computer Science*, pages 18–38. Springer Nature, Basel, Switzerland, 2021.

[Sea80] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–424, 1980.

[Smo87] P. Smolensky. Connectionist ai, symbolic ai, and the brain. *Artificial Intelligence Review*, 1(2):95–109, Jun 1987.

[Sow91] John F Sowa, editor. *Principles of semantic networks: Explorations in the representation of knowledge*. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann Pub, May 1991.

[SS04] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[ST01] M. Sato and H. Tsukimoto. Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 1870–1875 vol.3, 2001.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[Sun01] R. Sun. Artificial intelligence: Connectionist and symbolic approaches. In Neil J. Smelser and Paul B. Baltes, editors, *Inter-*

national Encyclopedia of the Social & Behavioral Sciences, page 783–789. Pergamon, Oxford, 2001.

[TSHL17] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 465–474. ACM, 2017.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(October):433–60, 1950.

[Twa10] Bhekisipho Twala. Multiple classifier application to credit risk assessment. *Expert Systems with Applications*, 37(4):3326–3336, 2010.

[Vit06] Andrew J. Viterbi. A personal history of the viterbi algorithm. *IEEE Signal Process. Mag.*, 23(4):120–142, 2006.

[VvdB17] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR). A Practical Guide*. Springer, 2017.

[YL99] John Yen and Reza Langari. *Fuzzy logic: intelligence, control, and information*, volume 1. Prentice Hall Press, Upper Saddle River, NJ, 1999.

[YWC+18] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking human out of learning applications: A survey on automated machine learning. pages 1–26, 2018.

[ZJC83] Zhi-Hua Zhou, Yuan Jiang, and Shi-Fu Chen. Extracting symbolic rules from trained neural network ensembles. *Mineral Processing and Extractive Metallurgy Review*, 1(1-2):207–248, 1983.

Acknowledgements

Optional. Max 1 page.