



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Formalization of a Cognitive Layered Model of Human Decision- Making for Robotic Applications

TESI DI LAUREA MAGISTRALE IN
AUTOMATION AND CONTROL ENGINEERING -
INGEGNERIA DELL'AUTOMAZIONE

Author: **Gabriele Ribolla**

Student ID: 964022

Advisor: Prof. Matteo Giovanni Rossi

Co-advisors: Livia Lestingi

Academic Year: 2021-22

Abstract

Robotic applications in service sectors such as hospitals, home care and education are rapidly becoming a basic technology as they can support people in need of care in a wide variety of tasks. Their use poses several ethical and technological questions. The scientific community wonders how service robotics can be a step forward in human care and help, and how robotic care systems can be built where the human is the main subject. Therefore, there is a growing demand for frameworks that support robotic application designers in a "human-aware" development process.

Politecnico di Milano has developed a framework for the development of robotic applications in service environments that can manage the unpredictability of human behaviour. The framework presents a formal model of human beings with a probabilistic formulation of free will, to make sure that they can ignore robot instructions or make autonomous decisions.

This thesis project illustrates the formalization of a more sophisticated human decision-making process based on the theory "Layered Reference Model of the Brain ", to be added to pre-existing human models so that the pre-existing framework can manage the unpredictability of human beings cognitively. This model has been made accessible to potential users through integration with the framework's automatic model generator.

In addition, the model has been compared with the current one through the formal verification of a significant human-robot interaction. The effects of the new parameters introduced have also been analyzed, which can describe new human behaviours, for example, if a human being is more determined or more distracted, extending the results obtained through the formal verification of a scenario.

Keywords: human decision-making; robotics application; formal model

Abstract in lingua italiana

Le applicazioni robotiche nei settori dei servizi come gli ospedali, l'assistenza domiciliare e l'istruzione stanno rapidamente diventando una tecnologia di base in quanto sono in grado di supportare le persone bisognose di assistenza in un'ampia varietà di compiti. Il loro utilizzo pone una serie di questioni etiche e tecnologiche. La comunità scientifica si chiede come la robotica di servizio possa essere un passo avanti nella cura e nell'aiuto umano, e come si possano realizzare sistemi robotici di assistenza per cui l'umano è il soggetto principale. Pertanto, vi è una crescente domanda di framework che supportino i progettisti di applicazioni robotiche in un processo di sviluppo "human-aware".

Il Politecnico di Milano ha sviluppato un framework per lo sviluppo di applicazioni robotiche in ambienti di servizio in grado di gestire l'imprevedibilità del comportamento umano. Il framework presenta un modello formale di esseri umani con una formulazione probabilistica del libero arbitrio, per fare in modo che essi possono ignorare le istruzioni del robot o prendere decisioni autonome.

Questo progetto di tesi illustra la formalizzazione di un processo decisionale umano più sofisticato basato sulla teoria "Layered Reference Model of the Brain ", da aggiungere ai modelli umani pre-esistenti, per fare in modo che il framework pre-esistente possa gestire l'imprevedibilità degli esseri umani in modo cognitivo. Tale modello è stato reso accessibile ai potenziali utenti tramite l'integrazione con il generatore automatico dei modelli del framework.

Inoltre, il modello è stato confrontato con quello attuale attraverso la verifica formale di una significativa interazione uomo-robot. Sono stati in seguito analizzati anche gli effetti dei nuovi parametri introdotti, i quali possono descrivere nuovi comportamenti umani, ad esempio se un essere umano è più deciso o più distratto, estendendo i risultati ottenuti tramite la verifica formale di uno scenario.

Parole chiave: processo decisionale umano; applicazioni robotiche; modello formale

Contents

| | |
|---|-----------|
| Abstract | i |
| Abstract in lingua italiana | iii |
| Contents | v |
| List of Figures | vii |
| List of Tables | ix |
| | |
| 1 Introduction | 1 |
| 1.1 Thesis Outline | 2 |
| | |
| 2 Related work | 5 |
| 2.1 Human Model | 5 |
| 2.1.1 Cognitive Models | 5 |
| 2.1.2 Task-Analytic Models | 7 |
| 2.1.3 Probabilistic Human Modelling | 8 |
| 2.2 Human Erroneous Behaviour | 10 |
| 2.3 Discussion | 11 |
| | |
| 3 Background | 13 |
| 3.1 Stochastic Hybrid Automata and Statistical Model Checking | 13 |
| 3.2 Pre-existing Models | 16 |
| 3.2.1 Human Follower | 17 |
| 3.2.2 Human Leader | 18 |
| 3.2.3 Human Recipient | 19 |
| 3.3 A layered reference model of the brain (LRMB) | 19 |
| | |
| 4 Formal model of human decision-making | 23 |

| | | |
|----------|--|-----------|
| 4.1 | Formalisation of LRMB | 23 |
| 4.1.1 | Parameter description | 24 |
| 4.1.2 | Functions developed | 28 |
| 4.2 | Extended pattern model | 35 |
| 5 | Automated Uppaal model generation code | 41 |
| 5.1 | Integration with the pre-existing framework code | 41 |
| 5.2 | User Steps | 43 |
| 5.3 | Code generation implementation | 46 |
| 6 | Experimental Validation | 55 |
| 6.1 | Mission analysed | 55 |
| 6.2 | Parameters analysed | 56 |
| 6.3 | Analysis varying delay and distance parameters | 56 |
| 6.4 | Comparison with pre-existing models | 62 |
| 6.5 | Discussion | 64 |
| 7 | Conclusions | 67 |
| | Bibliography | 69 |
| A | Appendix A | 73 |
| A.1 | All Model parameters | 73 |
| A.2 | System declaration example | 76 |
| A.3 | Analysis Parameters | 80 |
| A.4 | Function Algorithms | 84 |
| | Acknowledgements | 91 |
| | Ringraziamenti | 93 |

List of Figures

| | | |
|------|---|----|
| 3.1 | Human Follower Locations | 17 |
| 3.2 | Human Leader Locations | 18 |
| 3.3 | Human Recipient Locations | 19 |
| 3.4 | Functional Brain Model from [22] | 20 |
| 4.1 | Example graph SEEA0 | 25 |
| 4.2 | Example graph of a possible extraction | 29 |
| 4.3 | Extended Human Leader | 38 |
| 4.4 | Extended Human Recipient | 38 |
| 4.5 | Extended Human_Follower pattern changes | 39 |
| 5.1 | Automated code generator framework integration | 42 |
| 5.2 | User flowchart | 44 |
| 5.3 | CSV SEEA0 example | 45 |
| 5.4 | CSV importance_entity example | 45 |
| 5.5 | CSV importance_sense example | 46 |
| 5.6 | CSV obj_xye example | 46 |
| 5.7 | CSV arg example | 47 |
| 5.8 | Template folder structure example | 48 |
| 5.9 | CSV pattern config example | 48 |
| 5.10 | Mission folder structure | 49 |
| 5.11 | CSV mission config example | 51 |
| 6.1 | Point of interest of the layout from [16] | 55 |
| 6.2 | Layout abstraction as rectangular areas from [16] | 56 |

List of Tables

| | | |
|------|--|----|
| 3.1 | LRMB’s components in the Cerebrum | 21 |
| 6.1 | Queries verified | 57 |
| 6.2 | Recap Decision-Making setting | 57 |
| 6.3 | Recap analyses | 58 |
| 6.4 | First Analysis (A1) data | 59 |
| 6.5 | Second Analysis (A2) data | 60 |
| 6.6 | Comparison of mission probabilities between first (A1) and second analyses (A2) | 61 |
| 6.7 | Third Analysis (A3) data | 62 |
| 6.8 | Fourth Analysis (A4) data | 63 |
| 6.9 | Comparison of mission probabilities between third (A3) and fourth (A4) analyses | 63 |
| 6.10 | Queries verified for the comparison with the pre-existing models | 64 |
| 6.11 | Results comparison | 64 |
| A.1 | Cases Parameters | 73 |
| A.2 | Pattern parameters | 75 |

1 | Introduction

Revolutionary technological advances are shaping the future of the service industry. The innovations brought by Industry 4.0, such as IoT, pervasive sensorization, cloud computing and collaborative robotics, are spreading in non-industrial contexts with significant impacts on our daily lives. More importantly, highly sophisticated robotic systems being developed today are destined to transform the labour market once they become commercially available. The adoption of such solutions poses a number of problems ranging from technological challenges to ethical and social implications.

The scientific community wonders how service robotics can be a step forward in human care and help, and how robotic care systems can be built where the human being is the main subject. Therefore, there is a growing demand for frameworks that support robotic application designers in a "human-aware" development process.

Politecnico di Milano has developed a framework for the development of robotic applications in service environments that can manage the unpredictability of human behaviour. The framework presents a formal model of human beings with a probabilistic formulation of free will, to make sure that they can ignore robot instructions or make autonomous decisions.

The goal of this work is to enhance the framework for developing robotic applications in service settings such as hospitals, home assistance, and education. This framework plays a crucial role in simulating human-robot interactions (HRI), which are becoming more common due to the increasing use of mobile service robots. These robots operate in environments where humans are also present and are expected to become an important technology in various scenarios. A recent study on the future of employment estimates that specific jobs may be partially entrusted to robots with probabilities ranging from 60% to 90% [10].

By simulating real-life situations, designers can focus on the most likely scenarios and identify potential robot improvements for successful HRI. The proposed framework involves a group of humans requesting a service from a robot, with success being achieved when the robot completes all requests. The simulation provides information on the prob-

ability of success within a specific time frame or other specified properties. This data can help designers assess whether their quality criteria are met and make necessary modifications to the scenario. For example, if the probability of success is sufficiently high or the expected value of fatigue is not excessive for any human, the application can move forward to deployment or simulation.

The framework features a formal model of humans with a probabilistic formulation of free will, as they may ignore the robot's instructions or make autonomous decisions altogether.

The aim of the present work is the development of a more sophisticated formulation of the human decision-making process based on cognitive models existing in the literature, to be integrated into the existing formal model and validated to assess the improvement compared to the current model via formal verification of significant use cases.

The new decision-making model is based on the Layered Reference Model of the Brain, which indicates different concepts and elements that together define how a decision is taken by humans. The main goal was to encode this theory on the pre-existing human patterns and propose an analysis of what happens to change some parameters introduced, such as the distance at which the human can perceive the elements and the interval time between 2 decisions. That was not possible with the pre-existing decision-making models. Now it is possible to understand and characterise the patterns of different human behaviours, like distracted and not distracted or decided and undecided. It permits the improvement of the simulation data, representing additional human behaviours, there were not analysable with the pre-existing framework.

However, as with any model, there are some limitations in its ability to accurately and fully represent human decision-making. It is not able, for example, to take into account all the factors that influence human decisions, such as emotions or social influences. Even if the model has the probabilistic characteristic, it can reach a high accuracy but it is not possible to obtain every time the actual human decision. To improve this model, it might be useful to conduct further research to better understand human decision-making and to identify any gaps or limitations.

1.1. Thesis Outline

In Chapter 2, different existing structures used in human-robot collaboration (HRC) are analysed to understand and extract the main characteristics that the human model should have to be useful in human-robot interaction (HRI), which is the largest discussed nowadays because service mobile robots are becoming accessible to the mass market. In ad-

dition, discussions about human erroneous behaviour, focus on the errors that the model described has accounted for, and which are the main characteristics to take into account in the modelling development, specifically the probabilistic and the cognitive ones, are proposed.

In Chapter 3, it is illustrated the main tools and theories needed to develop the model following the requirements individuated. Specifically, the definitions of Stochastic Hybrid Automata (SHA) and Statistical Model Checking (SMC); the UPPAAL and UPPAAL SMC software; the pre-existing model, where the features are inserted; the theory for the layered reference brain model (LBRM), which is used to develop the model for its cognitive feature.

In Chapter 4, it is explained how the cognitive and probabilistic features of the model are developed, through the functions created and their addition to the pre-existing human models.

In Chapter 5, it is explained the creation of the automated code generation of the model and the integration with the pre-existing framework to permit a potential user to generate and simulate automatically the model in a desired scenario. Also, it is explained the needed steps for the correct simulation.

In Chapter 6, it was discussed about the collected data of the model developed with the same mission in which the pre-existing model was inserted, to analyse and understand their differences and discuss the behaviour that the model has, varying the new parameters not present in the pre-existing models.

In Chapter 7, the main important future works are presented and explained, considering their necessity for the next model's version, to improve the model for simulations close to reality.

2 | Related work

Nowadays, there are a lot of human models defined for different scopes and different ways of function. The main important human models are defined for collaborative robotics, in which humans and robots interact with each other to complete a task. Human modelling, in the Human-Robot Collaboration (HRC), is important to predict human behaviour and to analyse which are the critical points where it is needed to pay attention when the HRC operation is designed, specifically focusing on the main problems related to human behaviour that may decrease the safety of the collaboration. Hence a lot of human models were considered with this objective.

In this chapter, first is reported a detailed analysis of the different types of human models used for different applications in HRC, then errors that are defined by the human behaviour are defined and at the end, a discussion about the model used in HRC and their relations with the human behaviour errors are discussed.

2.1. Human Model

In this section, the HRC human models are analysed. There are 3 main families: the cognitive, the task analytic and the probabilistic models. They are different in structure and base characteristics. A description of each of them is here reported, with their existing structures.

2.1.1. Cognitive Models

Cognitive models, according to Askapour et al. [5], specify the rationale and knowledge behind human behaviour while working on a set of pre-defined tasks. They are often incorporated in the system models and contain a set of variables that describe the human cognitive state, whose values depend on the state of task execution and the operation environment. The main existing structures are here reported.

SOAR. In development for thirty years, Soar is a general cognitive architecture that integrates knowledge-intensive reasoning, reactive execution, hierarchical reasoning, planning, and learning from experience, with the goal of creating a general computational system that has the same cognitive abilities as humans. In contrast, most AI systems are designed to solve only one type of problem, such as playing chess, searching the Internet, or scheduling aircraft departures. Soar is both a software system for agent development and a theory of what computational structures are necessary to support human-level agents. Over the years, both software systems and theory have evolved [14]. The current version of Soar features major extensions, adding reinforcement learning, semantic memory, episodic memory, mental imagery, and an appraisal-based model of emotion. However, it only permits one operation at a time [5].

Adaptive Control of Thought—Rational (ACT-R). It is a detailed and modular architecture that depicts the learning and perception processes of humans. It is composed of modules that simulate declarative (i.e., known facts like $2+2 = 4$) and procedural (i.e., knowledge of how to sum two numbers) aspects of human memory. An internal pattern matcher searches for the procedural statement relevant to the task that the human must perform (i.e., an entry in declarative memory) at any given time [5]. According to the ACT-R theory, the power of human cognition depends on the amount of knowledge encoded and the effective deployment of the encoded knowledge [4]. So, it is possible to declare that a simulation of a scenario using the ACT-R structure is dependent on the human knowledge that is actively usable for that model, for that, to define a particular scenario it is needed to be sure that the knowledge of the human for the operation that we would like to simulate should be development, otherwise, it is possible to have a loss, representing the mission because the human model does not have some knowledge elements that may be useful in the simulation.

SAL. This model is an extended version of ACT-R, enriched with a neural architecture called LEABRA (local, error-driven and associative, biologically realistic algorithm)[5], that integrates into one coherent framework a set of basic neural learning and processing mechanisms that have been otherwise separately investigated in the neural modelling community. In Leabra, individual neuron behaviour is governed by a point-neuron activation function that uses simulated ion channels to update a membrane potential, with a nonlinear, thresholded output to other neurons. While this function is substantially simplified over detailed neurochemistry simulations, it is behaviorally richer than more abstract neural networks and produces results that cannot be replicated in its absence [13]. Leabra is a model of learning which is a balance between Hebbian, which is performed

using the CPCA (conditional principal components analysis) algorithm and error-driven learning with other network-derived characteristics. This model is used to mathematically predict outcomes based on inputs and previous learning influences. This model is heavily influenced by and contributes to neural network designs and models.

Programmable User Models (PUM). It defines a set of goals and actions for humans (physical and mental), 2 models one for the human and one for the machine, to distinguish between them. They have general and accurate semantics and could be well-suited inputs for automated formal verification tools upon simplification. On the other hand, they are very detailed and large and risk the state-space exploration phenomena. There exist simplified PUM models that are not able to define a generic HRC [5].

2.1.2. Task-Analytic Models

They analyze human behaviour throughout the execution of the task, therefore, they study the task as a hierarchy of atomic actions, starting from the most important and the most general task (e.g. parent task) to the actions that compose it (e.g. child task), to define which are the things to accomplish, to obtain the more general task. By definition, these models reflect the expected behaviour of the human, which leads to the correct execution of the task, and do not focus on reflecting erroneous behaviour. This model depicts human-robot co-existence and highlights the active role of humans in the execution of the task. However, does not always reduce the overall size of the state spaces of the model, especially when multiple human operators are involved in the execution. The consideration of the scenario for the correct use of these types of models makes them not usable in all situations and so not generalizable [5] but strictly dependent on the scenario. The main existing structures are here reported.

ConcurTaskTrees (CTT). As the name suggests, the structure of this model is strictly correlated with the concept of action hierarchy. It graphically permits the definition (with trees), of the relations between the tasks, defining, which are the more general and specific ones or which are at the same importance level. Moreover, this model can allow the definition of different types of users and applications or software that are involved in a scenario to accomplish the highest task considered [17].

User Action Notation (UAN). It is a task and user-oriented notation to represent behavioural asynchronous, direct manipulation interface designs[5]. Interfaces are specified in UAN as a quasi-hierarchy of asynchronous tasks. At the lower levels, user actions

are associated with feedback and system state changes. The notation makes use of visually onomatopoeic symbols and is simple enough to read with little instruction [12].

Enhanced Operator Function Model(EOFM). It is a generic XML-based notation, to gradually decompose tasks into activities, sub-activities and actions [5]. It uses state and derived variables to specify model behaviour. It models goal-level behaviours as activities. Each activity may include conditions that describe under what conditions it can be undertaken. Activities are decomposed into lower-level sub-activities and, finally, actions. Operators on each decomposition specify how many sub-activities or actions can execute and what the temporal relationship is between them. The EOFM standardizes the type of conditions that modify activities and supports all of the cardinalities and temporal orderings discussed earlier. The EOFM language is Extensible Markup Language (XML) based, thus making it platform-independent and easy to parse. It also supports a visual notation where tasks are represented as treelike graphs. The EOFM language allows for the modelling of a human operator as an input/output system. Inputs may come from the human–device interface, environment, mission goals, and other human operators. Output variables are human actions. The EOFM language allows for the modelling of a human operator as an input/output system. Inputs may come from the human–device interface, environment, mission goals, and other human operators. Output variables are human actions [6].

2.1.3. Probabilistic Human Modelling

This type of model helps to reproduce human non-determinism and uncertainty more vividly [5]. Unlike deterministic models, where there is the certainty of the result, stochastic/probabilistic models produce a probability distribution. In theory, a probabilistic human model could be very beneficial for model-based safety assessment in terms of a trade-off between cost and safety. They help to focus on what may happen with a great probability of success rather than a situation with a lower one and high solution-cost impact [5]. An issue about the probabilistic models is collecting a big enough data set for extracting the correct parameters for a probabilistic distribution, this is not simple, because the data can be not enough informative or are difficult to be collected [5], and also sometimes the data are not so informative for the scenario, hence for these type of models, the search and the analysis of the data to improve the model and obtain a close real human behaviour is very important. The main existing structures are here reported.

The Cognitive Reliability and Error Analysis Method (CREAM) It is a probabilistic cognitive model that is used for human reliability analysis (HRA), therefore, focuses on correct and incorrect behaviour; it defines a set of modes to replicate different types of human behaviour which have different likelihoods to carry out certain errors [5]. The identified cognitive model for CREAM methodology is called “CoCoM” (Contextual Control Model). The “CoCoM” model is based on the four cognitive function definition: observation, interpretation, planning and execution. CREAM divides the error events into observational errors (phenotypes, external manifestations error modes, consequence of non-observational errors) and non-observational ones (they occur during the human thinking process and they are the ultimate cause which leads to human errors). This method defines different correction Factors (CFPs) for the scenario that it is wanted to be analysed, in addition, there are several levels of each factor to reflect its effect on human performance. To reflect the scenario’s effects on human cognitive behaviours, the CREAM method defines four cognitive control modes, which are scrambled (unpredictable situation operator does not have control), opportunistic (limiting actions, lack of knowledge and staff competence), tactical (planned actions, the operator knows the rules and procedures of the system) and strategic (Operator has a long time to plan its work). The procedure to assess the error probability is to add to the CPC levels that contribute positively (\sum improved) and those who contribute negatively (\sum reduced)[9]. From the \sum pair, it is possible to define which is the probability error range, depending on the type of control modes that the pair locates in a scheme of the correlation between CPCs scores and control modes.

Systematic Human Error Reduction and Prediction Analysis (SHERPA). It is a qualitative probabilistic analysis of human errors with a task-analytic direction that contains several failure modes: action errors, control errors, recovery errors, communication errors, and choice errors. The SHERPA technique is considered the best technology available today in terms of accuracy for estimating human errors. The SHERPA model indicates, starting from the lowest hierarchical level of tasks, which errors are most credible in the analysis scenario. The analyst uses field experience and error taxonomy to determine the most credible failure mode for specific tasks. For each identified failure mode, the analyst describes the problem and establishes the consequences. After this, the analyst estimates the occurrence error probability, typically using a qualitative scale: Low - Medium - High. The same rating scale is used to assess the error severity. The final step is to propose and plan strategies for error reduction, which usually consists of changing the process and system.

The Human Error Assessment and Reduction Technique (HEART). It is an approach to define the modelling of human error, based on the reliability analysis of humans to quantify and identify the error and their probability to happen in a nuclear plant model, specifically on the error with the electricity generation. According to Askarpour et al. [5], this method as the following ones are strictly correlated with the scenario they are involved, and for that reason, they need high formalization and customization if the scenario changes concerning their ones. This technique, as explained by Williams [23], was designed to assist engineers not only to assess the likelihood and impact of human unreliability but to apply human-factors technology to optimize overall systems design. A similar method used in nuclear plants is THERP (Technique for Human Error Rate Prediction) [19].

Technique for Human Error Assessment (THEA). According to Pocock et al. [18] this technique can be defined in different steps to be followed to obtain a design of a plant that takes care of the human errors that may happen in a specific scenario, specifically: the scenario definition, to define all the details about the operation under design; the goal decomposition, to define all the tasks needed to obtain the final results, similar to the tasks-analytical model, with a hierarchy between tasks; the error analysis, answering a questionnaire ([18] appendix A) or define all the questions and ask to them about all the possible errors that can be had to reach the scenario's goal, in the plant, in the tasks defined, on the perception and interpretation of the scenario activities; the recording of the results, to compare the results obtained in the simulation of the scenario using software with the project requirements.

2.2. Human Erroneous Behaviour

According to Askarpour et al. [5] an important point about developing a human model is to introduce behavioural human errors, to underline the stochastic nature of the human model, whose is not deterministic such as a robot model, driven by differential equations for battery discharge and motion. Humans do not behave always in the same behaviour, due to the presence of the brain that decides every time which is the action to do in the situation in which the human is, but this action can be the not expected one in that situation for the human model developed. So to improve the modelling of a human model is necessary to understand which are the possible errors to focus on, and to try to insert them into the model. Askarpour et al. [5] suggest using a probabilistic model because it can be easily changed with a correct and informative data set. The main human behavioural errors can be defined and summarised:

- Slips and lapses regarding the quality of performance;
- Cognitive, diagnostic and decision-making errors due to misunderstanding the instructions;
- Errors of commission when the human does an incorrect or irrelevant activity;
- Idiosyncratic errors regarding social variables and human emotional state;
- Software programming errors leading to malfunctioning controllers that put the human in danger.

2.3. Discussion

In this section, a discussion of the main problems of the above models is considered, to understand which are the main characteristics of the model to be reliable. First of all, by analysing them some problems can be underlined. Cognitive models, according to Askarpour et al. [5], are very detailed and extensive. They often originate from different research areas (e.g., psychology), therefore have been developed with a different mentality from that of formal methods practitioners. So, they must be formalised, but their formalization requires a lot of effort for abstraction and HRC-oriented customization. They also require specialist training for modellers to understand the models and be able to modify them. Task-analytic models offer little re-usability; they are so intertwined with the task definition that a slight change in the task might cause significant changes to the model. They also must be first defined with a hierarchical notation for the easy and clear decomposition of the task and then be translated to an understandable format for a verification tool. On the other hand, probabilistic models seem to be an optimal solution; they combine either task-analytic or cognitive models with probability distributions. However, the biggest issue here is to have reliable and large enough data sets to extract the parameters of probability distributions. It requires the robotics community to produce huge training data sets from their system history logs (e.g., how frequently human moves in the workspace, the average value of human velocity while performing a specific task, the number of human interruptions during the execution, the number of emergency stops, the number of reported errors in an hour of execution). In addition, there is an important problem for these models, the complexity to check the model in a formal way using, for example, the stochastic model checking as it is possible with a model formalised as an automaton. In conclusion, to create a new model that can overcome all of these issues, the following characteristics must be considered as the model's implementation requirements:

- Take into count the error of human behaviour;

- Follow the reason mind cycle;
- Be based on the task, which the human should achieve in the simulation scenario;
- Be formalised as an automaton, to be checked with statistical model checking tools.

3 | Background

In this chapter are described and analysed the main tools and theory needed to reach the characteristics of the decision-making model indicated in Section 2.3.

Specifically, at the beginning the stochastic hybrid automata and statistical model checking tools are analysed, then the pre-existing models are illustrated and at the end the theory over which the decision-making model is based is described, considering its main important concepts and elements.

3.1. Stochastic Hybrid Automata and Statistical Model Checking

This section illustrates the pre-existing theoretical concept about the formal definition of Stochastic Hybrid Automata (SHA) used to capture complex temporal dynamics of physical phenomena. To introduce the definition of the Stochastic Hybrid Automaton it is necessary to introduce, according to Lestingi et al. [16], the following elements:

- Z , set of symbols;
- \sim , is a relation in $\{<, =\}$;
- χ^i ($i \in \{1, 2\}$), is an arithmetical term defined by the sum of the elements in Z and \mathbb{N} (e.g., $z_1 + z_2 + 3$, with $z_1, z_2 \in Z$);
- $\Gamma(Z)$, the set of conjunctions of constraints of the form $\chi^1 \sim \chi^2$, by definition, it included the logical constants true (e.g., $0 = 0$) and false (e.g., $0 = 1$);
- an update, it is a constraint where free variables are elements of Z and of its primed version Z' (for example $z' = z + 2$ and $z' \in Z'$);
- $\Xi(Z)$, the set of updates;
- \mathbb{R}_+ , the set of non-negative real numbers;
- \mathbb{R}^Z , the set of assignments to variables of Z (i.e., valuations).

Definition 1. A *Stochastic Hybrid Automaton* is a tuple

$$\langle L, W, F, D, I, C, E, \mu, P, l_{ini} \rangle$$

Where:

1. L is the set of locations and $l_{ini} \in L$ is the initial location;
2. W is the set of real-valued variables of which clocks $X \subseteq W$, dense-counter variables $V_{dc} \subseteq W$, and constants $K \subseteq W$ are subsets;
3. $\mathcal{F} : L \rightarrow \{(\mathbb{R}_+ \cup (\mathbb{R}_+ \times \mathbb{R})) \rightarrow \mathbb{R}^W\}$ is the function that assigns a set of flow conditions to each location, where flow conditions are (continuous) functions with one or two parameters, which assign a valuation to every time instant in \mathbb{R}_+ or to every pair constituted by a time instant in \mathbb{R}_+ and a constant value in \mathbb{R} , depending on the location;
4. $\mathcal{D} : L \rightarrow \{\mathbb{R} \rightarrow [0, 1]\}$ is the partial function assigning a probability distribution from $\mathbb{R} \rightarrow [0, 1]$ to locations which feature flow conditions with two parameters;
5. $\mathcal{I} : L \rightarrow \Gamma(W)$ is the function assigning a (possibly empty) set of invariants to each location;
6. C is the set of channels, including the internal action ϵ ;
7. $\mathcal{E} \subseteq L \times C_{!} \times \Gamma(W) \times \mathcal{P}(\Xi(W)) \times L$ is the set of edges, where $C_{!} = \{c! | c \in C\} \cup \{c? | c \in C\}$ is the set of events involving channels in C . Given an edge $(l, c, \gamma, \xi, l') \in \mathcal{E}$, l (resp. l') is the outgoing (resp. ingoing) location, c is the edge event, γ is the edge condition and ξ is the edge update. For each $l \in L$, $E(l) \subseteq C_{!} \times \Gamma(W) \times \mathcal{P}(\Xi(W)) \times L$ is the set of edges outgoing from l (for each $(c, \gamma, \xi, l') \in E(l)$ then $(l, c, \gamma, \xi, l') \in \mathcal{E}$ and vice versa);
8. $\mu : (L \times \mathbb{R}^W) \rightarrow \{\mathbb{R}_+ \rightarrow [0, 1]\}$ is the function assigning a probability distribution from $\{\mathbb{R}_+ \rightarrow [0, 1]\}$ to each configuration of the SHA, where configurations are (l, v_{var}) pairs constituted by a location $l \in L$ and a valuation $v_{var} \in \mathbb{R}^W$;
9. $\mathcal{P} : L \rightarrow \{(C_{!} \times \Gamma(W) \times \mathcal{P}(\Xi(W)) \times L) \rightarrow [0, 1]\}$ is the partial function assigning a discrete probability distribution from $\{(C_{!} \times \Gamma(W) \times \mathcal{P}(\Xi(W)) \times L) \rightarrow [0, 1]\}$ to locations such that, for each $l \in L$, $P(l)$ is defined if, and only if, $\varepsilon(l)$ is non-empty; also, the domain of the distribution is $\varepsilon(l)$ (hence, $\sum_{\alpha=(c!, \gamma, \xi, l') \in \varepsilon(l)} P(l)(\alpha) = 1$ holds).

In SHA, real-valued variables (i.e., a generalization of clocks) can evolve in time according

to generic expressions referred to as flow conditions [2]. The flow conditions constraining the evolution over time of variables in W are defined through sets of Ordinary Differential Equations (ODEs). This feature makes SHA a suitable formalism to model systems with complex dynamics, as it is possible to model through flow conditions, for example, laws of physics or biochemical processes. ODEs constraining clocks (for which $\dot{x} = 1$ holds for all $x \in X$, dense-counter variables, and constants (where $\dot{v} = 0$) holds for all $v \in v_{dc} \cup K$) are special cases of flow conditions.

If a variable $\theta \in V_{dc}$ is an independent term for a flow condition $f \in F(l)$ on location $l \in L$, i.e., $f = f(t, \theta)$, and θ is interpreted as a randomly distributed parameter, then f is a stochastic process [11]. We limit the analysis to flow conditions depending on at most one random parameter, as per Definition 1, which is enough to model human-robot interaction within the scope of our work.

SHA are eligible for Statistical Model Checking (SMC) [1]. SMC requires a model M with stochastic features (the SHA network), and a property ψ expressed, in our case, in Metric Interval Temporal Logic (MITL) over atomic propositions, belonging to set AP [3], which represent, for instance, constraints over set W (e.g., $w < 10$), or automata locations. SMC experiments are carried out with the UPPAAL tool. Unlike exhaustive model-checking, SMC does not explore the state space but it applies statistical techniques to a set of traces entailed by the formal model to estimate the probability of the desired property holding. More specifically, we compute the value of expression $\mathbb{P}_M(\psi)$ to estimate the probability of ψ holding for M [8]. Property ψ , in our framework, is of the form $\diamond \leq \tau ap$, where \diamond is the metric “eventually” operator and $ap \in AP$. The formula $\diamond \leq \tau ap$ is true if ap holds within τ time units from the time instant 0, i.e., the onset of the system. If the value of $\mathbb{P}_M(\psi)$ is compared against a threshold $\theta \in [0, 1]$, i.e., formula $\mathbb{P}_M(\psi) \sim \theta$ is evaluated, where $\sim \in \{\leq, \geq\}$, the result of the SMC experiment is a Boolean value indicating whether the probability of ψ holding for M is \sim than θ (thus, true) or not (yielding false), which is calculated through hypothesis testing. Otherwise, the SMC experiment returns a confidence interval $[p_{min}, p_{max}]$ of property ψ holding for M , with $p_{min}, p_{max} \in [0, 1]$, which is calculated according to the Clopper-Pearson method [7]. To determine when the generated set of traces is sufficient to conclude the experiment, UPPAAL checks the length of the interval $\epsilon = (p_{max} - p_{min})/2$. UPPAAL stops generating new traces when $\epsilon \leq \epsilon_{th}$ holds, where ϵ_{th} is an experimental parameter indicating the maximum desired estimation error. The smaller ϵ_{th} , the more accurate the estimation must be and, thus, more traces are required. Similarly, by applying the same Monte Carlo-based simulation, it is possible to calculate the expected value of distributions defined using functions \max (maximum) and \min (minimum) when they are applied to

stochastic processes, such as expressions of variables in W . Given an upper bound τ , formula $E_{M,\tau}[\max(v)]$ and $E_{M,\tau}[\min(v)]$, with $v \in W$, indicate respectively the expected value of the maximum and minimum value of variable v along executions that last at most τ time units.

UPPAAL is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

The simulation program used the extension of UPPAAL, the UPPAAL SMC (Statistical Model Checking), which refers to a series of techniques that monitor several runs of the system with respect to some property, and then use results from the statistics to get an overall estimate of the correctness of the design.[20]

3.2. Pre-existing Models

In this section, the pre-existing models made by Lestingi et al. [15] are considered. They focused on human and robot interaction (HRI) in the healthcare domain. In their approach, the designer specifies the main parameters of the mission to generate the model of the application, which includes mobile robots, the humans to be served, some of their physiological features, and the decision-maker that orchestrates the execution. All components are modelled through hybrid automata to capture variables with complex dynamics. The model is verified through Statistical Model Checking (SMC), using the UPPAAL tool, to determine the probability of success of the mission. The results are examined by the developer, who iteratively refines the design until the probability of success is satisfactory. The work was considered as a starting point for the development of the thesis's model for the probability, task-analytic and statistical model checking characteristics as discussed in Section 2.3.

They develop different human models using UPPAAL SMC to simulate different behaviours of humans and to verify the success of different scenarios. Specifically, for the aim of this project, the focus is on the following models:

- Human Leader;
- Human Follower;

- Human Recipient.

The mission proposed by Lestingi et al. [15] involves a group of humans requesting a service that implies interaction with a robot, that has to serve everybody.

These models are developed with different parameters that characterise the model and make them possible to be used in UPPAAL, specifically, the `id`, which represents the identification number of the human, that is needed in a scenario where more than one human is involved, to be able to differentiate the humans that can have different other parameters but the same pattern. The `_v` represents the speed at which the human is constantly moving during walking. The `p_f`, which represents the interaction pattern with the robot. The `p_fw`, which represents the freeWill behaviour of the pattern, which is possible to be set as high, normal or low.

3.2.1. Human Follower

This model represents the basic human behaviour, related to ordinary people that have to be served in the healthcare domain. Specifically, it has the goal to follow the robot with the feature to walk freely and choosing whether to follow or not when the robot issues its command. This is done by changing the introduced probability weights (**obey** and **disobey**). For example, if the value of **obey** is greater than **disobey** one, there is a higher probability that the human follows the robot and the command received. In the opposite situations, there is more probability that the human does not behave as its goal definition, so it does not follow the robot and the command received. The locations of

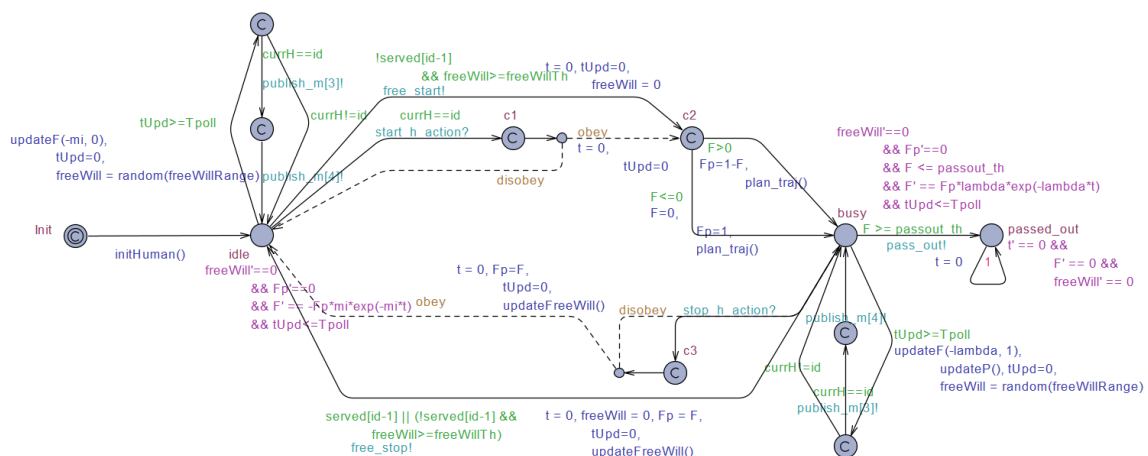


Figure 3.1: Human Follower Locations

the Human Follower are illustrated in fig.3.1. The main elements to pay attention to are:

- **Free will model**, which starts from the locations $c1$ and $c3$, and it involves the **obey** and **disobey** variables, whose are changed by the function `updateFreeWill()`;
- **idle**, location, which represents the human state in which it does not move, even if the robot is moving. In this location fatigue reduces;
- **busy**, location, which represents the human state in which the human is moving, and for that, the fatigue increases.
- F , time-dependent variable, which corresponds to the value of the fatigue at a generic time instant.

3.2.2. Human Leader

This model was developed to represent a different behaviour with respect to the human follower, in which it is the human that drives the robot and not vice-versa. It represents what a nurse can do, in situations in which the robot should be moved from one point to another (for example to carry a tool needed in a different nurse's position). The model covers unpredictable human behaviour, specifically the decision to start or stop. Its locations are illustrated in Figure 3.2.

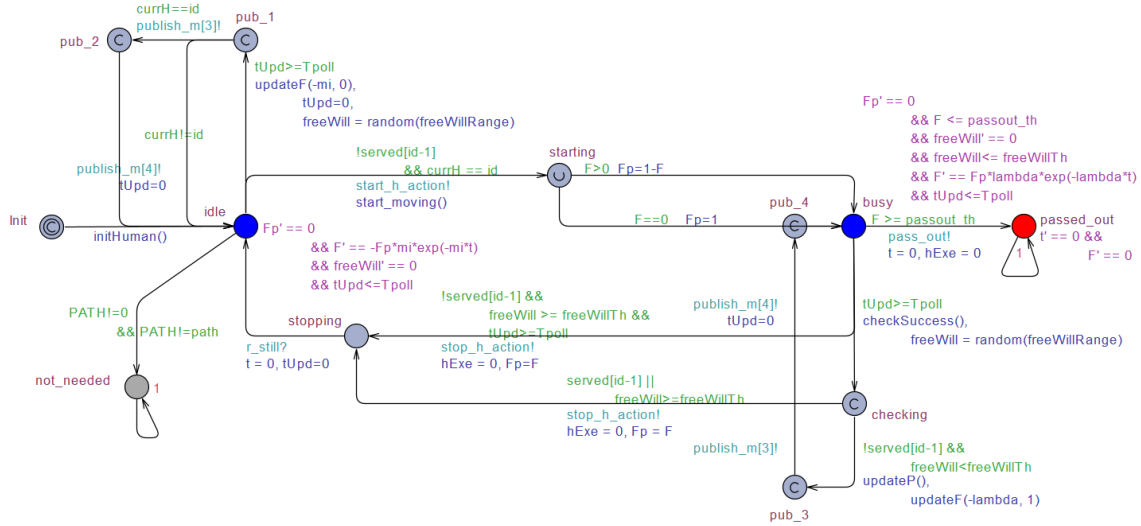


Figure 3.2: Human Leader Locations

The main elements different from the Human Follower are the new locations and edges considered to perform the pattern behaviour (e.g **checking**, **stopping**, **starting** and **passed out**).

3.2.3. Human Recipient

This model represents the behaviour of a human, who can move freely in the environment while waiting for a tool, which the robot is delivering. When the robot is close to the human, the interaction action started, and the human stops and picks up the item. For example, an operator (a doctor, a nurse, or both) in the imagined scenario, needs a tool to continue or finish the started work, so a request is sent, which is received by the robot, which, when it is loaded, leaves to deliver it at the operator to finish the work.

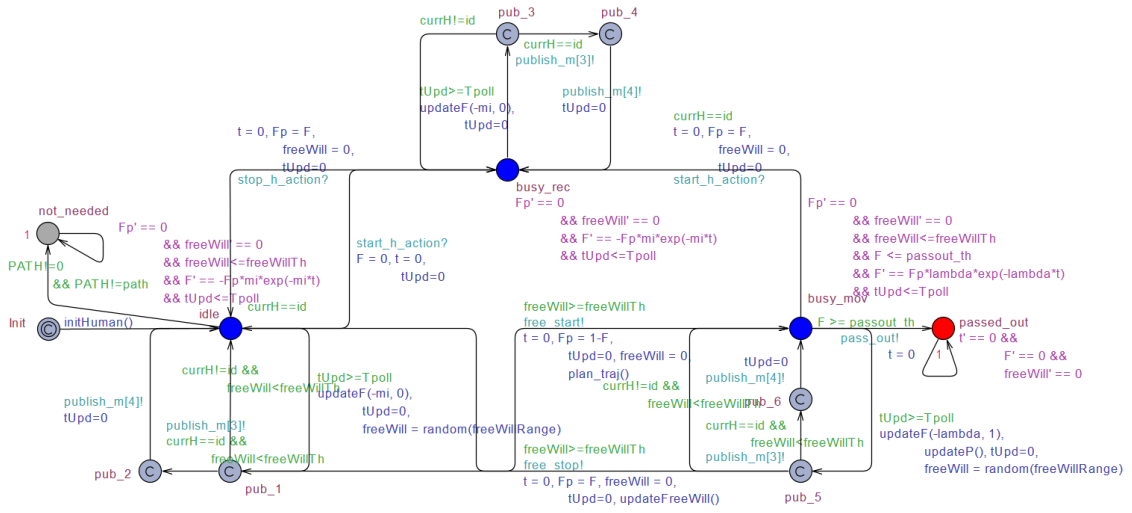


Figure 3.3: Human Recipient Locations

The locations of the Human Recipient are illustrated in Figure 3.3. The main elements different from the Human Follower are the locations involved to define its behaviour (e.g **busy_rec**, **busy_move**, and **passed out**).

3.3. A layered reference model of the brain (LRMB)

This section explicates the theory beyond the layered reference model of the brain (LRMB) introduced by Wang et al.[21, 22], which plays an important role in the cognitive feature of the decision model of this thesis's project, as discussed in Section 2.3. This theory attempts to formally and rigorously explain the functional mechanisms and cognitive processes (CPs) of natural intelligence. A comprehensive and coherent set of mental processes and their relationships, it encompasses 37 CPs at six layers known as the sensation, memory, perception, action, meta-cognitive, and higher cognitive layers from the bottom up. In Figure 3.4 is illustrated the brain model structure and the relations that the

components have between each other.

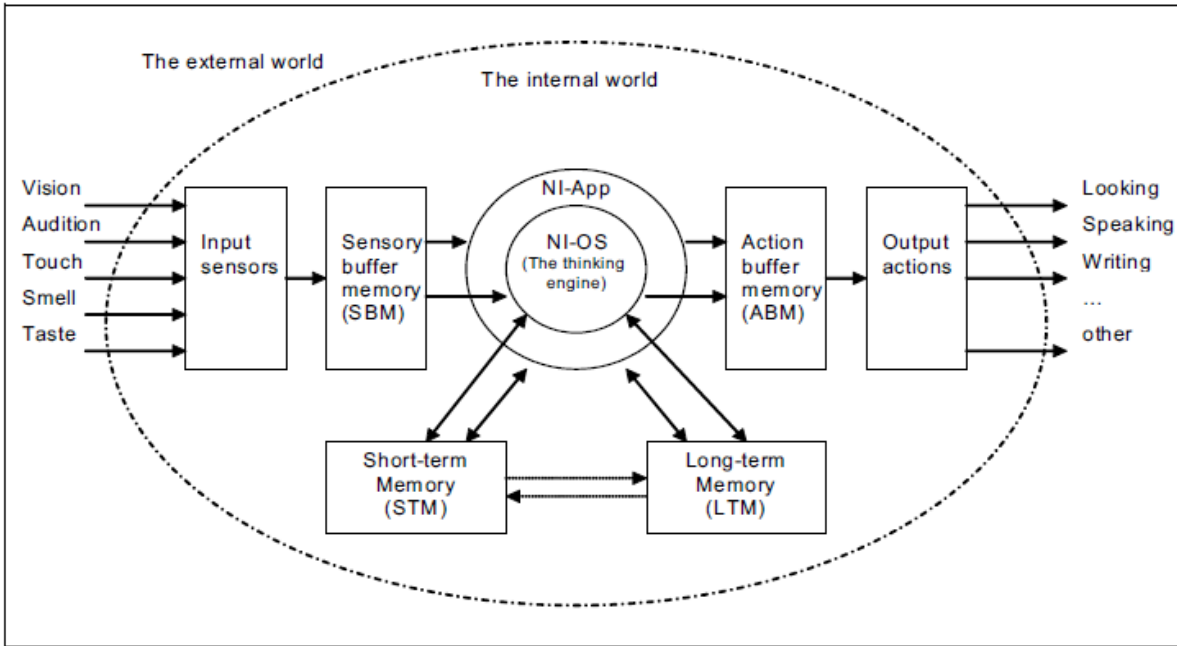


Figure 3.4: Functional Brain Model from [22]

The outputs of the model, as it happens in reality, represent the human actions generated as a response to the stimuli (the inputs) perceived. Specifically, it indicates how they are elaborated and interpreted by the internal world, which represents the conscious and subconscious parts of the human. These can be external and internal and be classified as follows:

1. Willingness-driven (internal events such as goals, motivation, and emotions);
2. Event-driven (external events);
3. Time-driven (mainly external events triggered by an external clock).

To express the relation between inputs and outputs, LMRB considers as elements the collaboration of the natural intelligent applications (NI-app) and the natural intelligent operating system (NI-OS), which together define the natural intelligent thinking engine (NI-sys). This element has to cooperate with: the sensory buffer memory (SBM), to perceive the environment; with the action buffer memory (ABM), to transform the decision taken into actual action in the external world; with the short-term memory (STM) and the long-term memory (LTM), to take a decision referring to the action that is done in a recent time and by experience. It is noteworthy that although the thinking engine NI-Sys is considered the centre of the natural intelligent NI, the memories are essential to enable the NI-Sys to function properly and to keep temporary and stable results retained and

retrievable. Thus, in a broad sense, the NI-Sys includes not only NI-OS and NI-App but also LTM, STM, SBM, ABM, as well as the perceptual sensors and action motors.

To understand better these concepts a bottom-up analysis is proposed, in Table 3.1 the part of the cerebrum for the LRMB is summarised.

| Element | Corresponding part in the Cerebrum |
|---------------|--|
| NI-OS | Thalamus |
| NI-app | Cerebral cortex |
| LTM | Association and premotor cortex in frontal lobe |
| STM | Temporal lobe |
| SBM | Sensory cortex (frontal lobe) and visual cortex (occipital lobe) |
| ABM | Primary motor cortex (frontal lobe), the supplementary motor in the area (frontal lobe) and procedural memory (cerebellum) |

Table 3.1: LRMB's components in the Cerebrum

NI-OS It represents the biological organ called the thalamus, it expresses the subconscious life functions (e.g. the 5 senses, memory, perception, self-consciousness, motivation, goal setting, etc...), which are correlated with the person's self, specifically, when a person is born, these are considered mature and fixed. Therefore, this layer is usually neither directly controllable nor intentionally accessible by the conscious life function layer (NI-app). The NI-OS encompasses the layers of sensation, memory, perception, and action. It is noteworthy that the subconscious life functions determine the majority of human behaviours and CPs and this might be overlooked in psychology and cognitive science.

NI-App It expresses the conscious life function layers, which include meta (e.g. attention, concept establishment, abstraction, search, categorization, memorization, knowledge representation) and higher cognitive (e.g. recognition, imagery, comprehension, learning, decision-making, problem-solving, etc...) functions. With respect to the NI-OS, NI-app is acquired, highly plastic, programmable, and can be controlled intentionally based on willingnesses, goals, and motivations. Specifically, it represents the acquired life applications.

SBM It represents the sensory cortex in the frontal lobe and the visual cortex in the occipital lobe, in the LRMB is the memory layer of the subconscious life functions and

it is a set of input-oriented temporary memory. The functional model of SBM is a set of queues corresponding to each of the sensors of the brain. The basic mechanism of SBM is that the contents stored in it can only last for a short moment until new information arrives at the same sensory. When the new information arrives, the old one in the queue should either be removed from STM or replaced by the new one. Specifically, it transforms the inputs by the sensors (the 5 human senses) to be useful for the Natural intelligent system (NI-sys, composed of NI-OS and NI-app).

ABM It is another memory, related to the action, with the objective to transform the information received by the NI-sys in actual response to the external stimuli in the external world. With respect to the SBM, ABM is an output-oriented temporary memory, therefore, it plays an important role in the brain to plan and implement human behaviours. The functional model of ABM is a set of parallel queues, each of them representing a sequence of actions or a process. Both the action and the sensation layers form a closed loop for implementing various life functions, particularly the cognitive life functions at the conscious layers. It is noteworthy that the wired and usually subconscious procedures known as skills are stored in ABM.

STM It is another memory involved in the brain, it represents the working memory of the brain because some neurological studies established that the Information lasting period in STM is about 24 hours (some literature considered it to be only a few minutes to a few hours). After this time span, the information will be either moved into the long-term memory or removed (erased) from STM. The functional model of STM is a set of stacks. in which the information buffered last will be processed first.

LTM It is the key memory of the brain, and it is mainly located at the association cortex in the frontal lobe of the cerebrum. It represents mainly the memorization meta-cognitive process, from which the brain retrieves the information of the memorised experience about sensation and perception of a lived situation. In addition, it also expresses the learning cognitive process, in which the information is added to the previous ones, changing the person's knowledge.

4 | Formal model of human decision-making

This Chapter aims to explain how the tools and theory analysed in the background (Chapter 3) are used to develop the new decision-making model for the pre-existing human pattern, specifically which are the changes adopted and the functions developed to formalise it.

First, the encoding of LRMB in UPPAAL with the parameters used and the functions developed are described. Following, the integration in the pre-existing models is explained.

4.1. Formalisation of LRMB

In chapter 3, the layered reference brain model was described [21, 22]. This theory is the starting point for the cognitive feature of the decision-making model.

This section illustrates the formalisation of the LRMB's structure in UPPAAL, which is possible to be encoded in an extraction sequence, that follows these steps:

1. Updates the weights, considering the events that can happen by the entities in the human neighbour;
2. Extract sequentially the sense, the entity, the event and the action;
3. Transform the action extracted with a value that the UPPAAL model can transform in a change of location.

These steps follow the LRMB sequence, where the stimuli are elaborated by the SBM and passed to the NY-sys that collaborates with the STM and the LTM (step 1 - step 2) to make the decision, which is transformed by the ABM in action (step 3).

4.1.1. Parameter description

Before introducing the functions developed for the extraction sequence, it is necessary to analyse the parameters and the variables used.

The **SEEA0** matrix permits the declaration of the pattern decision-making model behaviour, defining which are the possible actions that the human may perform in the environment if something happens. The definition of the combinations between the sense, entity, event, action, and output can be represented by, as an initial point, a graph, which makes the relationship representation simple to be transformed into a matrix form. For example, in a scenario where there are 2 entities, a robot and a person, the possible relations may be from the *vision* sense, the *robot* or the *person*. Then from the partial branch composed by the vision and the robot, the event may be *robotseen*, a possible connected action may be *move* and the transformation of it into a basic action (output of the decision-making model: run, walk, sit, stand) can be *walk*. Then the complete example branch is *vision* \rightarrow *robot* \rightarrow *robot seen* \rightarrow *move* \rightarrow *walk*. To make this branch be considered in the decision-making model developed, it is necessary to associate univocally an integer id for each category element. For example, consider the following ids for the elements: *vision* = 1, *robot* = 1, *robot seen* = 1, *move* = 1, *walk* = 1, the same branch can be seen as $1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1$. Hence, considering the graph in Figure 4.1, the associated **SEEA0** matrix is:

$$\text{SEEA0}_{5 \times 5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 4 & 1 \\ 2 & 1 & 2 & 2 & 2 \\ 2 & 1 & 2 & 5 & 4 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

Which can be defined equally with meaningful words or phrases to the ids as:

$$\text{SEEA0}_{5 \times 5} = \begin{bmatrix} \textit{vision} & \textit{robot} & \textit{robot seen} & \textit{move} & \textit{walk} \\ \textit{vision} & \textit{person} & \textit{person seen} & \textit{move} & \textit{walk} \\ \textit{touch} & \textit{robot} & \textit{robot touched} & \textit{stop and sit} & \textit{sit} \\ \textit{touch} & \textit{robot} & \textit{robot touched} & \textit{stop} & \textit{stand} \\ \textit{hearing} & \textit{alarm} & \textit{alarmheard} & \textit{go away} & \textit{run} \end{bmatrix}$$

Generally, if we would like to define m possible combination of sense, entity, event, action,

and output the structure is the following:

$$\text{SEEA0}_{m \times 5} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{15} \\ \vdots & \vdots & \ddots & \vdots & \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & a_{m5} \end{bmatrix}$$

Where a_{ij} are integer values ($i \in [1, m]$ and $j \in [1, 5]$).

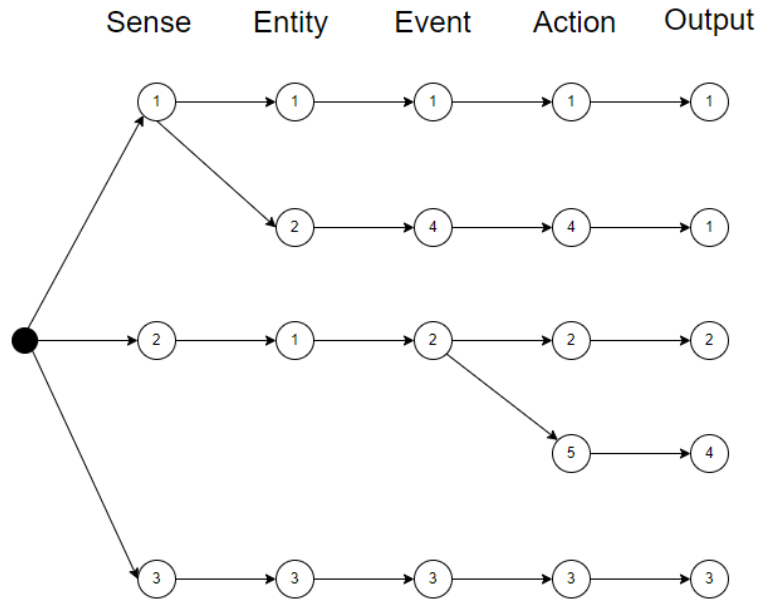


Figure 4.1: Example graph SEEA0

The new decision-making model permits also setting a desired importance weight to each sense and the entity, considering as parameters the `importance_entity` and `importance_sense` vectors. They have on the row (vector index i) the weight related to the sense and entities with $id = i + 1$, respectively.

$$\text{importance_entity}_{m \times 1} = \begin{bmatrix} \text{weight_entity}_1 \\ \text{weight_entity}_2 \\ \vdots \\ \text{weight_entity}_m \end{bmatrix}$$

$$\mathbf{importance_sense}_{m \times 1} = \begin{bmatrix} weight_sense_1 \\ weight_sense_2 \\ \vdots \\ weight_sense_m \end{bmatrix}$$

For example, if we would like that the pattern should consider more the robot with respect to the person, it is possible, to consider $robot_weight = 10$, while the $person_weight = 5$. Assuming that $robot_id = 1$ and $person_id = 2$, the related matrix has the following declaration:

$$\mathbf{importance_entity}_{2 \times 1} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$$

Where the first row represents the $robot_weight$ while the second is the $person_weight$. Due to the ids associated with these entities. The **importance_sense** matrix follows the same declaration logic, hence, consider $vision_id = 1$, $touch_id = 2$, $smell_id = 3$ and $vision_weight = 20$, $touch_weight = 5$, $smell_weight = 10$, the **importance_sense** matrix has the following structure:

$$\mathbf{importance_sense}_{3 \times 1} = \begin{bmatrix} 20 \\ 5 \\ 10 \end{bmatrix}$$

The addition of the fixed objects present in the environment is performed by a matrix, called **obj_xye**, which is used to declare them, on its rows. The pieces of information needed are the position (x, y) and the id of the fixed entity.

$$\mathbf{obj_xye}_{m \times 3} = \begin{bmatrix} x_1 & y_1 & id_1 \\ x_2 & y_2 & id_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & id_m \end{bmatrix}$$

For example, if in the environment there are a cupboard and a chair that have an absolute position concerning the mission starting point equal to $(4000.0, 700.0)$ [cm] and $(400.0, 270.0)$ respectively, and the related ids are 5 and 3 ($cupboard_id = 5$ and $chair_id = 3$). The **obj_xye** is:

$$\mathbf{obj_xye}_{2 \times 3} = \begin{bmatrix} 4000.0 & 700.0 & 5 \\ 400.0 & 270.0 & 3 \end{bmatrix}$$

To set other parameters of the model the **arg** vector is used. It collects 23 different parameters to characterise different pattern characteristics.

$$\mathbf{arg}_{25 \times 1} = \begin{bmatrix} \textit{upd_weight} \\ \textit{case_upd_weight} \\ \textit{stm_weight} \\ \textit{ltm_weight} \\ \textit{case_weight_entity} \\ \textit{case_weight_sense} \\ \textit{case_event} \\ \textit{case_fixed_obj} \\ \mathbf{delay} \\ \textit{last_id_extracted_output} \\ \textit{case_last_id} \\ \mathbf{case_obj} \\ \textit{distance_min} \\ \mathbf{distance_threshold} \\ \textit{case_stm_ltm} \\ \textit{n_stm} \\ \textit{n_ltm} \\ \textit{id_robot} \\ \textit{id_walk} \\ \textit{id_sit} \\ \textit{id_run} \\ \textit{id_stand} \\ \textit{id_person} \end{bmatrix}$$

The parameters to focus on, between all the **arg** vector ones, are the **case_obj**, which permits to select of the entities to base the decision-making (e.g. only the fixed object, only the fixed object and robot, only the human or all). This parameter is important because can change the number of entities on which the decision-making model is dependent. For example, assuming to have 7 fixed objects, 1 robot and 4 humans. Selecting **case_obj** = 3 (case in which the pattern considers robots, fixed objects and humans for the decision-making process), the total entities in which the decision-making is defined is 12, while if **case_obj** = 2 (fixed objects and robots) the number of entities is 8. In the Chapter 6 is analysed the mission probability for 2 different missions where the number the entities involved changes, it is possible to say from the data obtained that if the decision-making

process is based on 8 entities with respect to 12 the mission probability is greater, hence if the pattern is less focused (more entities that can be perceived) the has a lower probability to succeed.

The `distance_threshold` [cm], which defines the distance at which the entities in the environment are perceived by the human, for the computation of the events that may happen and consecutively the output sequence extraction's weights. In Chapter 6, different analyses show how varying the value of this parameter changes the mission probability, specifically with values greater than 600 [cm] the mission probability decreases, this is associated with the fact that the pattern has a very large neighbour to perceive the entities and for that, it is possible to be considered more distracted (more entities on which the decisions are made), the best results in terms of mission probabilities was obtained considering `distance_threshold = 200` [cm].

Another important parameter is the `delay` [s] parameter, which sets off the time at which the model has to wait before taking another decision when the robot is not directly interacting with it. In Chapter 6, some analyses show how setting its value between 12 and 60 [s], the mission probability at 400 [s], is close to 50 %. while considering lower and greater values the probability decreases. This parameter can represent a human that is decided (`delay > 60[s]`) or undecided (`delay > 12[s]`), and ordinary (`12[s] <= delay <= 60[s]`)

The `case_event` parameter is used to select how the events may happen are defined, randomly (`case_event = 1`) or considering the distance of the entities with respect to the pattern position (`case_event = 2`).

The `case_upd_weight` parameter instead is used to set if the action weights change considering the actions stored in STM and LTM at every extraction (`case_upd_weight = 1`) or randomly (`case_upd_weight = 2`).

In the Appendix A.1, there are tables where all arg elements are analysed (cases in Table A.1, parameters in Table A.2).

4.1.2. Functions developed

The model works thanks to the cooperation of different functions, that reach as output the action that the human will do, depending on the entities that the human has close to it. To obtain the human output action a sequence of functions, that represent different extractions, one for the sense, one for the entity, one for the event, one for the action and one for the transformation of the extracted action to a primary action as walk, sit,

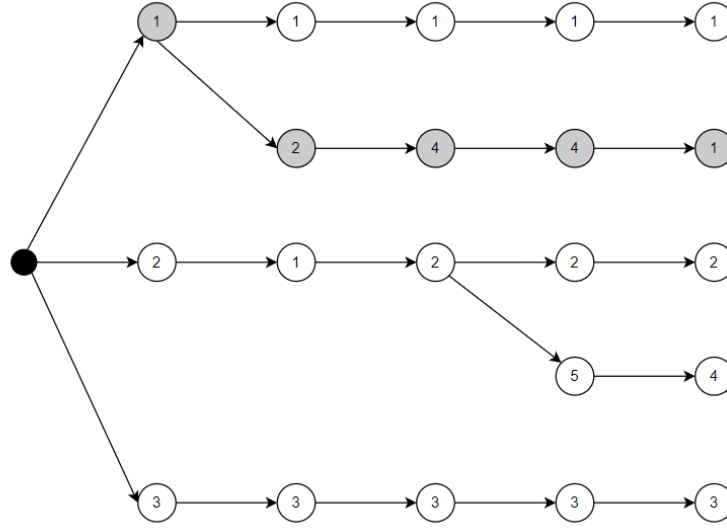


Figure 4.2: Example graph of a possible extraction

stand, run, are involved. The relations between sense, entity, event, action and output are defined by **SEEA0** matrix as described in Section 4.1.1. The function used to incorporate the entire decision-making model functions is the *out_id()* (Algorithm 4.4) function. It has the objective to call the other functions in order to obtain the id of the output action. In it, there are different sub-functions with different tasks. The main important one is the *update_weight()* (Algorithm 4.1), which represents the function that computes the weights of sense, entity, event, and action, for the weight-based extraction of the id with the relative extraction functions, which are: *sense_id_extraction()* (Algorithm 4.2), *entity_id_extraction()* (Algorithm A.1); *event_id_extraction()* (Algorithm A.2); *action_id_extraction()* (Algorithm A.3). Each of these functions has the task to extract the id of the element that is possible to be extracted depending on the relations defined by the **SEEA0** matrix. For example, referring to the **SEEA0** matrix defined in Section 4.1.1, if the sense id extracted by the *sense_id_extraction()* function is equal to 1, the entities ids that can be extracted are 1 or 2. If the *entity_id_extraction()* extracts 2 the only possible id event to be extracted by *event_id_extraction()* is 4, and for the simplicity of the **SEEA0** matrix taken as an example, the id of the action extracted will be 4 and its transformation will be 1, which is the output of the *out_id()* if the extraction follows the same ids extracted. The extraction graph representation is reported in Figure 4.2, in grey are highlighted the node selected by the example explained.

The *output_id_extraction()* (Algorithm 4.3), is a function different with respect to the previous ones because starting with the ids extracted from each function before, it identifies the output action, searching into the **SEEA0** matrix the row, which has the sense,

entity, event, action combination, and it returns the output action related to that row.

An additional function with the role to define short-term and long-term memory is the *stm_ltm()* (Algorithm 4.5), this function cooperates with the *update_weights()* one to compute the weight of the action, considering the previous action extracted.

To conclude this section the algorithms of the most important function are here analysed, the others are illustrated in Appendix A.4.

Updating the weights

This operation is accomplished by the *update_weights()* (Algorithm 4.1), which computes the weights of sense, entity, event, and action, for the weight-based extractions of the ids with the relative extraction functions, that are called after this one. It can be divided into 3 phases. The first one, the reset to zero of some variables is needed to avoid the dependency of the weights of the previous decision-making process, to compute and define the weights depending on the situation in which the pattern is in the scenario. The next phase is the actual computation of the weights of senses, entities and actions for the events that may possibly happen and refer to the active case, in the Algorithm 4.2, is reported as an example, the computation of the entities weights if the **case_weight_entity** selected is the zero one (in Table A.1 are reported all the other cases). The last phase is needed to normalise the weights and to have a maximum weight and probability of extraction equal to 1 for the sense, event, action and entities. For example, assuming that the *id_entity* = 1 related to the event that may happen, line 10 of the Algorithm 4.1, add 1 to the previous weight for that entity, hence if the *id_entity* = 1 is present in the **SEEA0** matrix 10 times, the final weight associated with that entity is 10. If there is also another event that may happen with an *id_entity* = 2 and for 5 times, the weight associated with that entity is 5. Then, **entity_weight_vector** = [10, 5] before entering the normalisation phase. After, **entity_weight_vector** is equal to [0.66, 0.33] (0.66 = 10/15 while 0.33 = 5/15). Because the **entity_weight_sum** is equal to 15 (10+5, computed in the for cycle from 18 to 20 line) and the division of the vector elements for that value (cycle for from line 21 to 23). In other cases, the same logic is used but the weights can be different. For example, with **case_upd_weight_entity** = 1 the addition is not given by 1 but, with the value set for that entity (in **importance_entity** vector). Then assuming the same example above but with the following **importance_entity** vector:

$$\mathbf{importance_entity}_{2 \times 1} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}$$

Algorithm 4.1 *update_weights()*

```

1: Input: event_weight_sum, entity_weight_sum, sense_weight_sum,
   action_weight_sum, sense_weight_vector, entity_weight_vector,
   event_weight_vector, action_weight_vector, event_may_happen,
   SEEA0, case_weight_entity, case_weight_sense, id_vector_action,
   stm_action, ltm_action, case_upd_weight, stm_weight, ltm_weight,
   importance_entity, importance_sense
2: //Phase 1 - Reset to zero some variables
3: event_weight_sum, entity_weight_sum, sense_weight_sum ,
   action_weight_sum sense_weight_vector, entity_weight_vector,
   event_weight_vector, action_weight_vector = 0
4: //Phase 2 - Compute the weights considering the active cases
5: for all events that may happen do
6:   // Search in the SEEA0 matrix for each row with the related
     id_event to add weight to the element_weight_vector for the
     entity, sense and action considering the cases active
7:   if case_weight_entity == 0 then
8:     for k = 0 to n_entity do
9:       if id_vector_entity[k] == id_entity then
10:        entity_weight_vector[k] = entity_weight_vector[k] + 1.00
11:      end if
12:    end for
13:  end if
14:  //Definition of all the other cases for the entity sense and action
15:  :
16: end for
17: //Phase 3 - Normalise weights
18: for i = 0 to n_sense do
19:   sense_weight_sum = sense_weight_sum + sense_weight_vector[i]
20: end for
21: for i = 0 to n_sense do
22:   sense_weight_vector[i] = sense_weight_vector[i]/sense_weight_sum
23: end for
24: //Some transformation for event, action and entity weight vector
25: :

```

The results is $\text{entity_weight_vector} = [0.8, 0.2]$, because $\text{entity_weight_sum} = 125$ ($10 * 10 + 5 * 5 = 100 + 25$) and the vector before the normalisation is defined as $\text{entity_weight_vector} = [100, 25]$.

Extraction of the sense

The extraction of the sense is an operation implemented in *sense_id_extraction()* (Algorithm 4.2), which was developed to obtain the id of the extracted sense with weights

Algorithm 4.2 *sense_id_extraction()*

```

1: Input: sense_weight_vector, N_interval, interval_vector_sense,
           n_sense id_found
2: //Phase 1 - Reset to zero some variables
3: id_selected, selected_vector, id_extracted, id_trovato = 0
4: //Phase 2 - Define some variables for the extraction
5: for q1 = 0 to n_sense do
6:   interval_vector_sense[q1] = sense_weight_vector[q1] * N_interval
7: end for
8: for q = 1 to n_sense do
9:   interval_vector_sense[q]
     = interval_vector_sense[q - 1] + interval_vector_sense[q]
10: end for
11: //Phase 3 - Sense extraction
12: l = 0
13: while id_trovato == true do
14:   if rand ≤ interval_vector_sense[l] then
15:     id_extracted = id_vector_sense[l];
16:     id_found = false
17:   else
18:     l = l + 1
19:   end if
20: end while
21: return id_extracted

```

computed by the *update_weights()* function. It is possible to divide it into 3 phases. The first, as what happens in phase 1 of Algorithm 4.2, is needed to reset some variables. In the second phase, there is the creation of different variables used for the extraction. Specifically, the *interval_vector_sense* has the objective to define the interval in which a random number can finish. Considering, for example, *id_vector_sense* = [1, 2, 3, 4, 5] and *sense_weight_vector* = [0.2, 0.3, 0.5, 0, 0], the line 4 define the following operation *interval_vector_sense* = [0.2, 0.3, 0.5, 0, 0]101 = [20.2, 30.3, 50.5, 0, 0], and at line 7, *interval_vector_sense* = [20.2, 50.5, 101, 101, 101]. Now, phase 3 is considered. If *rand* = 50 (*rand* ∈ [1, 100]), *id_extracted* = 2, because it is in the interval of the probability of the sense in position 1 of the *id_vector_sense*. *N_interval* = 101 is considered to avoid the problem that can happen if *N_interval* = 100, when the sum of the weights reaches 99 and *rand* = 100, in which the while loop never ends. For example if *sense_weight_vector* = [0.33, 0.33, 0.33, 0, 0], then *interval_vector_sense* = [33, 66, 99, 99, 99] and *rand* = 100 is never found.

Obtaining the output id

To obtain the output that is defined for the row uni-vocally defined by the combination of sense, entity, event and action extracted by the relative functions, the *output_id_extraction()* function is used. Its main phase is the second one (Algorithm 4.3), in which the search for the id of the output is done over the **SEEA0** matrix, till the matrix's row which verifies the if condition ($SEEA0[m][0] == \text{sense_id_extracted} \ \&\& \ SEEA0[m][1] == \text{entity_id_extracted} \ \&\& \ SEEA0[m][2] == \text{event_id_extracted} \ \&\& \ SEEA0[m][3] == \text{action_id_extracted}$). The id found for that row is returned as the output of the function. For example, assuming that the ids extracted are $\text{sense_id_extracted} = 2$, $\text{entity_id_extracted} = 1$, $\text{event_id_extracted} = 2$, $\text{action_id_extracted} = 5$ and the **SEEA0** matrix is equal to defined in Section 4.1.1

$$SEEA0_{5 \times 5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 4 & 1 \\ 2 & 1 & 2 & 2 & 2 \\ 2 & 1 & 2 & 5 & 4 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

The $\text{output_id_extracted}$ is equal to 4 because the combinations of the sense, entity, event and action extracted uniquely selected the fourth row, which has in the fifth column (related to the basic action transformation of the action extracted) the output id is equal to 4.

Algorithm 4.3 *output_id_extraction()*

```

1: Input: sense_id_extracted, entity_id_extracted, event_id_extracted,
           action_id_extracted, SEEA0
2: //Phase 1 - Reset to zero some variables
3: id_extracted = 0
4: //Phase 2 - Search in SEEA0 matrix the row that has the combination of
   the ids of sense, entity, event and action extracted
5: for m = 0 to n_rows do
6:   if  $SEEA0[m][0] == \text{sense\_id\_extracted} \ \&\& \ SEEA0[m][1] ==$ 
       $\text{entity\_id\_extracted} \ \&\& \ SEEA0[m][2] == \text{event\_id\_extracted} \ \&\&$ 
       $SEEA0[m][3] == \text{action\_id\_extracted}$  then
7:     id_extracted =  $SEEA0[m][4]$ 
8:   end if
9: end for
10: return id_extracted

```

Extraction sequence

To obtain the id of the output that the human will do, the *out_id()* (Algorithm 4.4) function is developed. It is the one used in the template location, to set the value of the *id_out* variable, which is compared with the values set in for the parameters *id_walk*, *id_run*, *id_sit* and *id_stand* to move in the location after the decision-making process (called by the function *out_id()*, and ended with the set of *id_out* variable value). The main important phase is the third one where there is the extraction sequence to obtain the *id_out* value. Phase 2 provides a control on the possible events, to avoid starting the extraction sequence, because it works only if there is at least one event that may happen, so if there is at least one entity that is perceived by the pattern, which is in the neighbour of it (e.g. If the *case_event* is set to 2 an entity is considered to be in the pattern's neighbour if the distance between the pattern and the entity is lower than the *distance_threshold* parameter set.) On the other hand, the lack of possible events provides that the *id_out* extracted is equal to the *last_id_extracted_output* defined by the *case_last_id* set (Table A.1).

Short and Long term memory

The *stm_ltm()* (Algorithm 4.5) function represents the short-term and long-term memory of the LRMB. It tries to describe human behaviour, for which, if an action was already done in a specific situation it can have a greater probability with respect to an action that was never done before. This function provides all the cases defined for the *case_stm_ltm* parameter set (Table A.1). The action stored in the *stm_action* (short-term memory) and *ltm_action* (long-term memory) vectors are used in the *update_weights()* function to modify (if the case related is active or not) the action's weight adding to the *action_weight_vector* for the action in the *stm_action* or in the *ltm_action* vectors the weights selected for the *stm_weight* and *ltm_weight*, respectively (like what happens in *update_weights()* function). For example, assuming the *case_stm_ltm* = 3, in which the *stm_action* and *ltm_action* have FIFO (First Input First Output) behaviour, and *n_stm* = 6 (maximum number of actions that can be saved in the *stm_action*), *n_ltm* = 10 (maximum number of actions that can be saved in the *ltm_action*) and if the condition defined in line 13 is true, so when the *stm_tail* == *n_stm* (when the number of actions in *stm_action* is equal to *n_stm*, hence the *stm_action* is full) the *stm_ltm()* removes one action (*stm_dequeue()*) and then add one action (*stm_enqueue()*) to the *stm_action* vector, otherwise if the *stm_action* is not full (else line 16), the action is only added. The same is done for the *ltm_action* (from line 19 to line 24). Hence considering *stm_action* = [1, 2, 3, 4, 3, 0] and *stm_tail* = 5, and the *action_id_extracted* = 6, be-

Algorithm 4.4 *out_id()*

```

1: Input: sense_id_extracted, entity_id_extracted, event_id_extracted,
          action_id_extracted, last_id_extracted_output
2: //Phase 1 - Set and increase the values of some variables
3: event_all_zero, index_cycle = 0
4: //Phase 2 - Call of initial function and control if there is at least
   one event for the extraction
5: id_initialization() //to set in a crescent way the value of the id related
   to sense, event, action and entity considered
6: assignment_last_id() //to assign the last id for the initial extraction
7: update_event_may_happen() //to set which are the events that may happen
8: //control if there is at least one event that may happen in the
   event_may_happen vector
9: //Phase 3 - Call the function for the id_out extraction
10: if There is at least one event possible then
11:   update_weight()
12:   id_sense_extracted = sense_id_extraction()
13:   id_entity_extracted = entity_id_extraction()
14:   id_event_extracted = event_id_extraction()
15:   id_action_extracted = action_id_extraction()
16:   id_extracted_output = output_id_extraction()
17:   stm_ltm()
18:   last_id_extracted_output = id_extracted_output
19: else id_extracted_output = last_id_extracted_output
20: end if
21: return id_extracted_output

```

cause *stm_tail* < *n_stm*, *stm_action* becomes [1,2,3,4,3,6], and so in the next decision-making process due to *stm_tail* = 6, the *stm_action*, if *action_id_extracted* = 3, becomes [2,3,4,3,6,3], and it continues to be updated as in this last example for the next decision-making processes because *stm_tail* has reached *n_stm* value.

4.2. Extended pattern model

The pre-existing human models (section 3.2) were modified to introduce the new decision-making model. The main idea was to insert the *out_id()* function with some changes for the correct integration, without changing totally the patterns. To make the new decision-making model easy to be added also in more complex patterns. Specifically, its output is compared with values set for the access of an edge, which moves the pattern in the related location.

To develop this idea some changes were done to have the correct functioning of the model.

Algorithm 4.5 *stm_ltm()*

```

1: Input: index_random, weight_ltm, weight_stm, stm_tail, n_stm,
          ltm_tail, n_ltm, stm_action
2: //Phase 1 - Definition of different cases for the memory loss or no
   loss, random or FIFO
3: if case_stm_ltm == 0 then
4:    $\vdots$ 
5: end if
6: if case_stm_ltm == 1 then
7:    $\vdots$ 
8: end if
9: if case_stm_ltm == 2 then
10:   $\vdots$ 
11: end if
12: if case_stm_ltm == 3 then
13:   if stm_tail == n_stm then
14:     stm_dequeue() //function to remove an action in the stm
15:     stm_enqueue() //function to insert the action extracted in the stm
16:   else
17:     stm_enqueue()
18:   end if
19:   if ltm_tail == n_ltm then
20:     ltm_dequeue() //function to remove action in the ltm
21:     ltm_enqueue() //function to insert the action extracted in the ltm
22:   else
23:     ltm_enqueue()
24:   end if
25:
26: end if

```

Firstly, eliminate the elements that are not needed, related to the previous decision model, specifically all the things depending on the *updateFreeWill()* function: the **obey** and **disobey** weights and the **freeWill** variable (e.g., **freeWill = 0** and **freeWill'== 0**, **freeWill >= freeWillTh** and **freeWill < freeWillTh**). Secondly, add all the elements for the new decision-making model, specifically the insertion of the *out_id()* function in the model, which takes the place of the *updateFreeWill()* function. With the new model, to move from idle to busy locations or vice versa, a different way, as introduced above, is needed. It is based on the introduction of 4 edges, each one with a guard that verifies the condition to access that edge if the value of the *out_id()* function stored in the **id_out** variable corresponds with the constant set for the id of the output action (Specifically **id_out == id_walk**, **id_out == id_sit**, **id_out == id_stand**, **id_out == id_run**). As discussed in Section 4.1.1, the new decision-making model has the possibility to set the

interval time between 2 decisions. To introduce this characteristic an additional edge and a guard on the edge at which the `out_id()` function is called is considered, in particular, the extraction can be done when the `tOut >= delay`, where `tOut` represents the time passed between 2 decisions, otherwise the decision remains the previous one.

In Figure 4.5, Figure 4.3 and Figure 4.4 are highlighted the parts that are changed for the extension of the model considering the changes explained above, for the **Human_Follower**, **Human_Leader** and **Human_Recipient** respectively. Specifically for the Figure 4.5, it is proposed a direct comparison with the pre-existing **Human_Follower** model.

In these figures, the area defined with the letter **A** represents the autonomous decision-making process that the pre-existing models do when they do not receive any command from the orchestrator, for this part, in the extended pattern, there is the introduction of the 4 edges and the maintenance of the updates and guards that are not related to the `freeWill` variable as discussed above.

Part **B**, instead represents the `freeWill` variable update, in the extended models this part has different changes, specifically the ones for the `tOut` variable. It has 2 edges one for the no-decision case (`tOut < delay`) and the other when the decision is made (`tOut >= delay`). In this edge there are new updates: `tOut = 0`, `extraction_new = 1` and `id_out = out_id()`.

While part **C** represents the decision-making process that happens when the pattern receives the orchestrator commands (`start_h_action` and `stop_h_action?`), specifically this is used to define the erroneous behaviour that a human can have when a command is given, ideally if orchestrator sends `start_h_action?` command, the pattern has to start (go in **busy** location), while if it is sent the command `stop_h_action?` in **idle**. The pattern as it is represented can move in **idle** even if the command sent `start_h_action?`, depending on the probability weights `obey` and `disobey` (updated by the `updateFreeWill()`). This, in the extended pattern model, for the new decision-making models is done considering the `out_id()` function.

For the **Human_Leader** and **Human_Recipient**, it is possible to notice that the C area is not present, that is because the pre-existing human patterns for the leader and the recipient do not involve the pre-existing decision model with the `obey` and `disobey` weights to avoid the orchestrator command (`start_h_action?` and `stop_h_action?`), as happened in the human follower one, because, for the **Human_Leader**, there is no need of that part. After all, it does not receive any command by the orchestrator. While the **Human_Recipient**, due to the immediate action that should happen between **busy_move** and **idle**, it is not possible to consider the pattern given by the `obey` and `disobey` weights (even if it has

other "erroneous behaviours" that are not reported in this thesis).

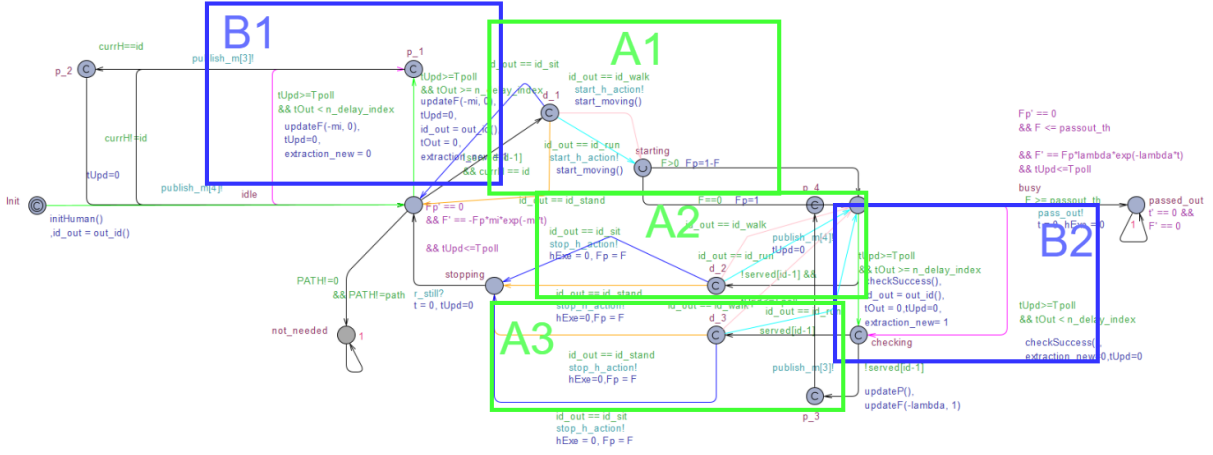


Figure 4.3: It represents the extended **Human_Leader**, where A represents the autonomous decision-making process and B the decision delay developed (see Section 4.2).

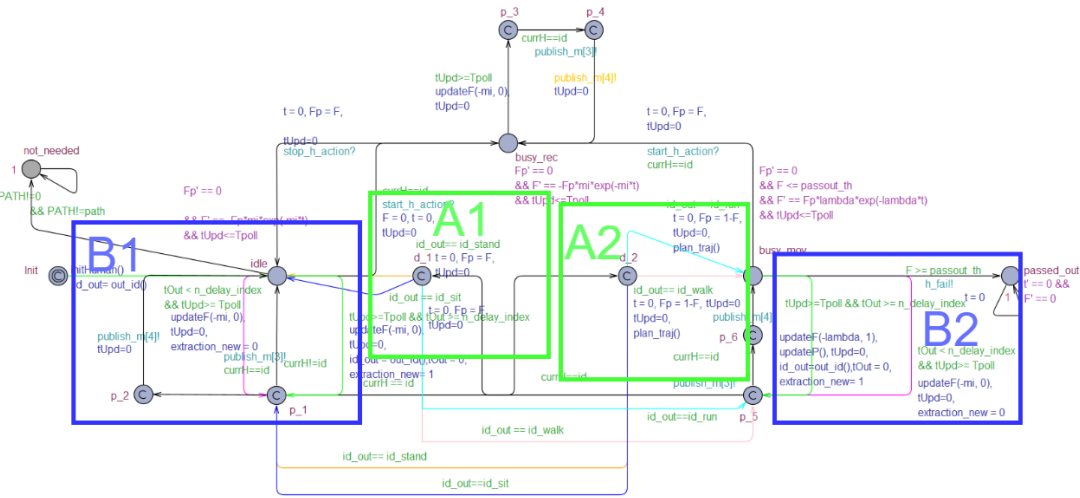


Figure 4.4: It represents the extended **Human_Recipient**, where A represents the autonomous decision-making process and B the decision delay developed (see Section 4.2).

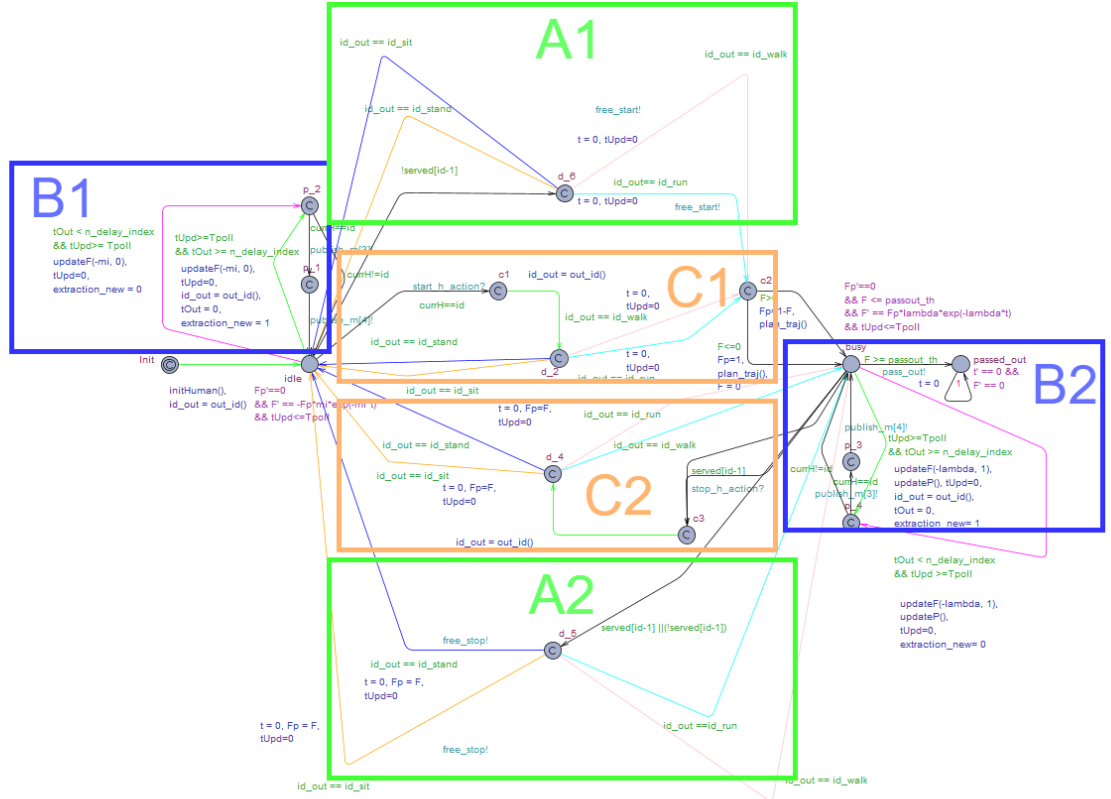
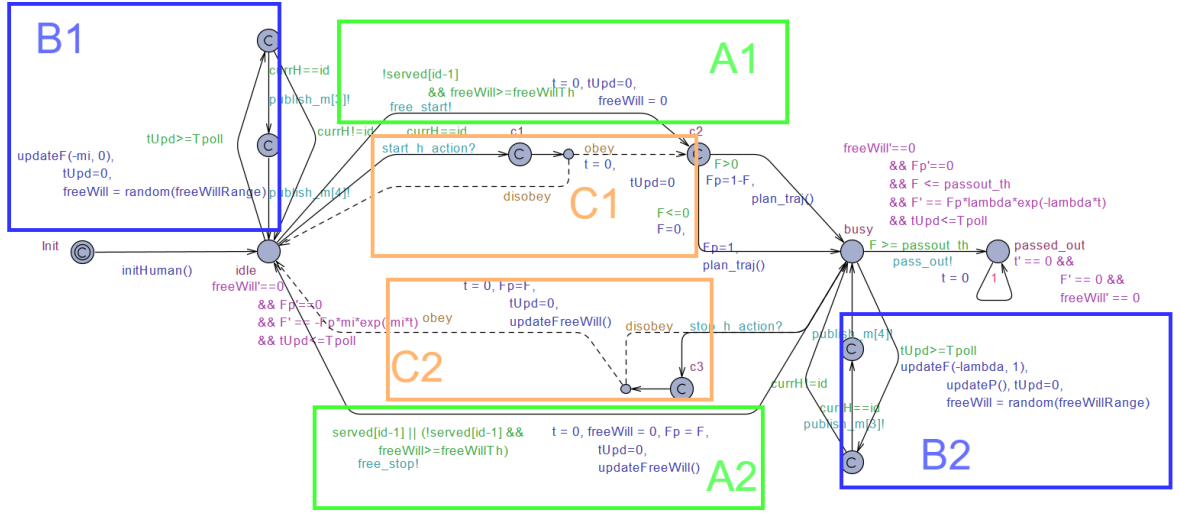


Figure 4.5: Changes from the pre-existing (4.5a) to the extended human follower pattern (4.5b) where the correlations between the pre-existing parts and the extended parts have the same letter, where A represents the autonomous decision-making process, B the decision delay developed and C the decision-making process for the orchestrator commands (see Section 4.2)

5 | Automated Uppaal model generation code

Aiming to insert the extended models in the pre-existing framework, to be used by a potential user to set up its desired scenario with them, the pre-existing automated UPPAAL model generation code has to be modified to make it possible. The main additions are the insertion of the extended pattern models and the creation of a tool that helps the user to declare the extended pattern parameters.

In this chapter, firstly, the automated model generator integration is briefly discussed. Then, the user steps to declare its own parameters file are discussed. Lastly, the tool code is analysed, showing the output structure.

5.1. Integration with the pre-existing framework code

The Automated UPPAAL Model Generator is an important component of the framework because it permits to generate automatically the UPPAAL model of the desired mission to obtain the verification results of the UPPAAL queries declared. The pre-existing procedure to obtain the UPPAAL model was to create a DSL file for the desired mission that the user would like to analyse, adding all the information needed, and then a JSON file is used to pass this information to the pre-existing UPPAAL model generator that using the pre-existing patterns for the humans as `Human_Leader`, `Human_Follower`, `Human_Recipient` and for the other actors like the `robot` and the `orchestrator` and the `main.xml` file, which has the objective to collect all the patterns in one file that can be open in UPPAAL and compiled to be used to verify the queries declared and obtain the verification results. The pre-existing UPPAAL model generator has the goal to fill the placeholders that are present in the patterns and in the other XML files with the information present in the JSON file, making a working UPPAAL model, with all the data previously declared by the user. The integration of the extended human pattern developed in this thesis with the pre-existing UPPAAL model generator has the objective to maintain the pre-existing procedure without modifying the pre-existing code. To accomplish this, 2 additions are made. First, add

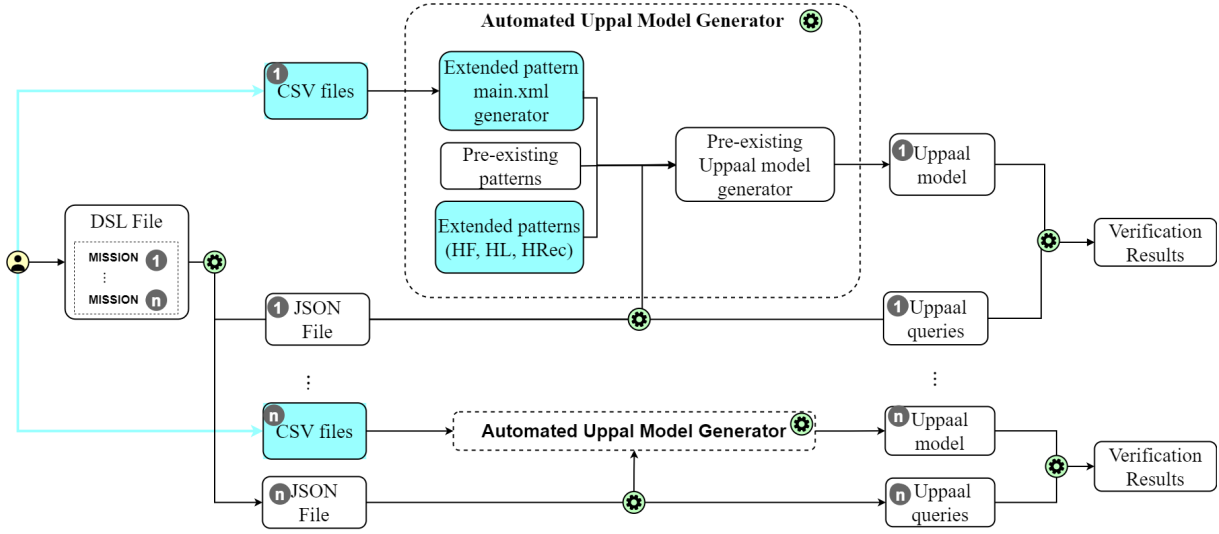


Figure 5.1: Diagram presenting the process that translates DSL and CSV files into Uppaal models with the integration of the extended patterns developed and the extended pattern main.xml. Boxes are numbered to identify the relations between defined missions and the output files of each phase, while yellow circles mark actions performed by the user (i.e., the application designer) and green circles correspond to the automated tasks.

the extended pattern of the **Human_Follower**, **Human_Leader** and **Human_Recipient** with the correct pre-existing code placeholders to make the insertion of the data declared for the mission in the correct place in the pattern. This step does not make any changes in the pre-existing code because it works already with the pre-existing pattern, so with some modifications of the extended pattern developed (insertion of the placeholders) the addition is accomplished. The second, concerning the previous one, was more complex, to the fact that the creation of an external tool is needed, specifically, it gives the possibility to set the parameters of the extended patterns in the correct way to make the mission UPPAAL model to work correctly and be used to obtain the verification results.

This tool, to the fact that the patterns are developed considering the parameters as arguments of the template, has the objective to declare them in the system declaration part of the UPPAAL model file. This part is where the instances of the mission actors are involved.

To accomplish that a cpp code is written to create this tool. This is also called the extended patterns main.xml generator because it creates the main.xml file with the system declaration and the global declaration needed to the pre-existing code to generate a final mission UPPAAL model that works correctly.

The tool for defining the parameters of the patterns in the system declaration following

the UPPAAL language rules to avoid compile errors needs the information related to the extended pattern in CSV files (in the Section 5.2), that are associated with the extended pattern that the user would like to add to the mission. To make this tool to be used by the pre-existing UPPAAL model generator, because it needs the main.xml file to work correctly, a small change in the pre-existing code is made, specifically to add the launch of this tool before the pre-existing code UPPAAL model generation procedure starts, to have the main.xml file present for the UPPAAL model generation. Hence the pre-existing code calls the tool, that generates the main.xml file from the CSV files, and when the main.xml file is created the pre-existing code starts its procedure having all its elements working correctly. In Figure 5.1, the additions are done to make the integration of the extended patterns to the pre-existing code in the automated UPPAAL model generator framework's components are shown in light blue. They are the extended patterns developed in this thesis and the extended pattern main.xml generator that uses the CSV files generated by the user to work properly.

5.2. User Steps

In this section, a description of the steps (Figure 5.2) that the users have to follow is provided to correctly generate the UPPAAL model of the desired mission using the extended pattern.

First, the users have to think about the scenario and the pattern of humans that are involved, and then they have to transform the data for the mission filling the JSON and CSV (comma-separated values) files. The last ones are related to the human pattern characteristics that the users would like to have in the mission, specifically to add information about the extended pattern model, which is provided with a different decision model, based on different parameters, as described in Section 4.1.1. The main parameters are the **SEEA0** matrix, the **importance_sense** vector, the **importance_entity** vector, the **obj_xye** matrix and the **arg** vector. A way to accomplish that is to create a folder for each human pattern with the new decision feature, where the parameters are in a CSV file. For example, the following **SEEA0** matrix:

$$\mathbf{SEEA0}_{5 \times 5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 4 \\ 1 & 1 & 1 & 3 & 2 \\ 1 & 1 & 1 & 5 & 4 \end{bmatrix}$$

It is transformed in the CSV file defined in the Figure 5.3, where each row of the CSV file

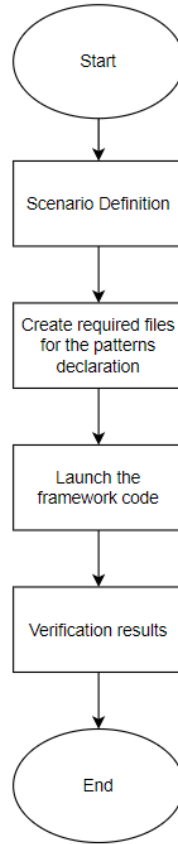


Figure 5.2: User flowchart

represents each row of the **SEEA0** matrix. The first row of the CSV files is used to have a recap of the data needed for that parameter to be set. In the Figure 5.4 and in Figure 5.5, there are respectively examples of how the **importance_entity** and **importance_sense** vectors should be declared, specifically the data for the addition of a sense and an entity required for a correct row of the CSV file are the entity name, the id and the weight associated with it. While in Figure 5.6, the declaration of a possible **obj_xye** is provided, the data to be declared are the position (x,y) and the id of the entity. The **arg** has to be declared as in Figure 5.7, in which each row represents one of the 23 additional extended pattern parameters (Table A.2 and Table A.1).

All these parameters can be easily declared by the user, creating the related CSV file. The extended pattern main.xml generator to create the correct parameters declaration in the main.xml file needs also the config.csv file, in which there are the relations between the parameter and the CSV file created for that. In Figure 5.9 is shown an example of this file, for a pattern called **HF_1** of the mission **DPA**. When all of these files are created, it is possible to insert them in a folder that can have the structure shown in Figure 5.8. All



```
DPA > HF_1 >  HF_seeao.csv
1  sense,entity,event,action,output
2  1,1,1,1,1
3  1,1,1,2,4
4  1,1,1,3,2
5  1,1,1,5,4
```

Figure 5.3: CSV **SEEAO** example, each row represents the combination of the ids of sense, entity, event, action and output that the pattern performs in the mission


```
DPA > HF_1 >  HF_impEntity.csv
1  entity,id,weight
2  robot,1,10.0
3  person,2,10.0
4  doctor,3,10.0
5  chair,4,5.0
6  cupboards,5,5.0
7  door examination open,6,5.0
8  door examination closed,7,5.0
9  enter door open,8,5.0
10 enter door closed,9,5.0
11 alarm,10,5.0
12 crowd,11,5.0
```

Figure 5.4: CSV **importance_entity** example, each row represents one entity that the pattern considers, with the id and the weight associated.

the parameters CSV files and the related config.csv have to be in.

After the creation of the folders for each human involved in the mission, it is possible to insert all of these in the mission folder, with a **mission_config.csv** (Figure 5.11) file, in which at each row in the first column is indicated the type of the human (**HF**, is for the human follower; **HL**, for the human leader; **HRec**, for the human recipient), in the second the human folder name (e.g. human_name_1, human_name_2, human_name_n), and in the third column the name of the config.csv file in the human folder. Hence the final mission folder has the structure shown in Figure 5.10. In which all the pattern folders are inserted.

```
DPA > HF_1 > HF_impSense.csv
1  sense,id,weight
2  sight,1,10.0
3  touch,2,10.0
4  hear,3,10.0
5  taste,4,10.0
6  smell,5,10.0
```

Figure 5.5: CSV `importance_sense` example, each row represents one sense that the pattern considers, with the id and the weight associated.

```
DPA > HF_1 > HF_obj_env1.csv
1  x,y,entity
2  400.0,270.0,2.0
3  4000.0,270.0,2.0
4  1400.0,450.00,5.0
5  1200.0,680.0,2.0
6  3000.0,450.0,5.0
7  200.0,200.0,3.0
8  4000.0,700.0,3.0
```


Figure 5.6: CSV `obj_xye` example, each represents one fixed object perceived by the pattern, with the position (x,y) and the id of the entity associated. In this case, the id should be inserted as a double, the pattern internally transforms it into an integer.

5.3. Code generation implementation

In this section, the automated model generation code is illustrated (Algorithm 5.1). It can be divided into three phases, that sequentially create the `main.xml` file with the correct structure to be used by the pre-existing code to generate the UPPAAL mission model, adding the required file in the correct place for the file definition. Specifically, these files are sequentially added to the variable `vector_string_main`, which is used to generate (in line 17) the `main.xml` file, in the folder receiving (`xml_main_folder`) set.

The first phase is related to the generation of the global declaration. To create it, different files are added at the `vector_string_main` variable with the function `add_string(vector_string,vector_string_to_add)`, which returns the `vector_string` with the `vector_string_to_add` added and the function `f_print_txt_file(file)`, which creates a `vector_string` of the file, that can be added to the `vector_string` using the `add_string(...)` function. The global declaration in UPPAAL XML file is defined by

```

DPA > HF_1 >  HF_arg.csv
1  name,value
2  upd_weight_ref,50.0
3  case_upd_weight_ref,0.0
4  stm_weight_ref,1.0
5  ltm_weight_ref,2.0
6  case_weight_entity_ref,1.0
7  case_weight_sense_ref,1.0
8  case_event_ref,2.0
9  case_fixed_obj,1.0
10 n_delay_index_ref,1.0
11 last_id_extracted_output_ref,1.0
12 case_last_id_ref,2.0
13 case_obj_ref,3.0
14 distance_min,0.0
15 distance_threshold_ref,400.0
16 case_stm_ltm_ref,3.0
17 n_stm_ref,10.0
18 n_ltm_ref,6.0
19 id_robot,1.0
20 id_walk,1.0
21 id_sit,2.0
22 id_run,3.0
23 id_stand,4.0
24 id_person,2.0

```

Figure 5.7: CSV `arg` example, each row has the associated parameter and the value defined for it (see Table A.2 and Table A.1). In this case, also the parameters that are integers such as the cases should be inserted as a double, the pattern internally transforms them into their correct format

pieces of XML code that open and close them, these files are added in the global declaration at the initial point where they start and where they end. Respectively at line 4 by *open_global_dec* XML file and in line 8 by *close_gloDec* XML file. Between the open and closed files, the global declaration can be created. In the code is done, initially adding the pre-existing global declaration (line 5), by the *global_dec_pre* XML file, and then adding the global declaration related to the extended pattern in line 6 by *global_dec_ext* file. Line 7, represents the insertion in the global declaration of a part that is dynamically generated by the code from the CSV files created. Specifically, the *ad_globDec_dyn_format()* function, generates the piece of UPPAAL code in Listing 5.3, which is defined by the information declared in the CSV files for the correct compilation of the UPPAAL model and makes it possible to have one template for the pattern but the possibility to have different pattern parameters, specifically it adds in UPPAAL code the maximum number of rows between the **SEEA0**, the **importance_sense**, **importance_entity** and **obj_xye** matrices between the patterns declared. For example, assuming that one pattern has 57 rows of the **SEEA0** matrix and another has 37, the variable in the global declaration related to **nr_seeao_max** is equal to 57. In Figure 5.3, it is shown the piece

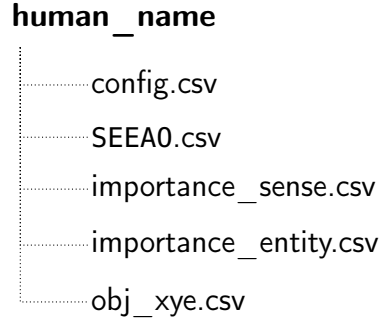


Figure 5.8: Template folder example, represents the structure of how a pattern folder has to declare. It has to contain all the parameters files defined in the config.csv file (see Figure 5.9)

```

DPA > HF_1 > HF_config.csv
1  csv file imp ENTITY name,HF_impEntity.csv
2  csv file imp SENSE name,HF_impSense.csv
3  csv file SEEA0 name,HF_seea0.csv
4  csv file OBJ name,HF_obj_env1.csv
5  csv file ARGV name,HF_arg.csv
_

```

Figure 5.9: CSV pattern config example, each row represents a parameter and the file associated for its generation

of code added in the global declaration for a mission where the maximum number of rows of the `importance_entity` is 11, 5 for the `importance_sense`, 7 for the `obj_xye` and 57 for the `SEEA0` matrix. This helps in phase 2 (from line 9 to line 14) to have in the system declaration all the parameters with the same dimension, but different in the contents. The code fills automatically the parameter rows that are needed to reach the maximum number of rows for that parameter, setting their values to zero. For example, if a pattern `SEEA0` matrix has 3 rows and the maximum is 5, the declared `SEEA0` matrix for that pattern is set equal to:

$$\text{SEEA0}_{5 \times 5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 4 \\ 1 & 1 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Where the first three rows are the rows declared in the `SEEA0.csv` file. The related UPPAAL code declaration is reported in Listing 5.1.

To pass the information on the number of rows at which the pattern code has to stop

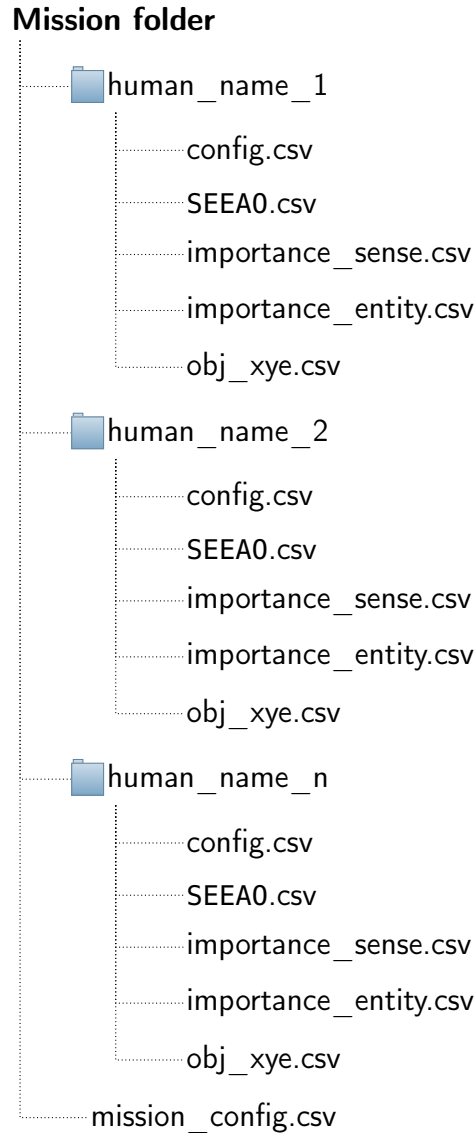


Figure 5.10: Mission folder structure example. It represents how a mission folder defined by n pattern has to be defined

to avoid operating also on the rows that are not declared by the user but set to zero by the code, the `arg_ad` vector is used, it has the objective to pass different information to the template, to make possible the use of only one pattern but with different parameters. It has the structure defined in Listing 5.2, in which the information related to different parameters is declared. Specifically for the rows of the **SEEA0** matrix (`nr_seea0`), and `obj_xye` (`nr_obj`), the others are defined by the code looking at the **SEEA0** matrix, considering the maximum id value for the related category.

The `vector_string_system_dec` is used to store the system declaration to be added in between the open (line 12) and close (line 14) UPPAAL templates for the system declara-

```

1  /* matrix seeao num HF_1 */
2  const int SEEA0_HF_1[nr_seeao_max][nc_seeao]={
3  {1,1,1,1,1},
4  {1,1,1,2,4},
5  {1,1,1,3,2},
6  {0,0,0,0,0},
7  {0,0,0,0,0}
8  };

```

Listing 5.1: Example of SEEA0 matrix with number of rows lower respect to the maximum set to 5

```

1  /* args_ad HF_1 */
2  const int arg_ad_HF_1[nr_argAd]={
3  35, /* nr_seeao*/
4  1, /* n_sense*/
5  11, /* n_entity */
6  11, /* n_event */
7  20, /* n_action */
8  4, /* n_output */
9  7 /* nr_obj */
10 };

```

Listing 5.2: Example of `arg_ad` vector definition, where some pattern's parameters are declared automatically by the code, taking the information from the CSV file defined

```

1  const int nr_seeao_max = 57;
2  const int nr_impEntity_max = 11;
3  const int nr_impSense_max = 5;
4  const int nr_obj_max = 7;

```

Listing 5.3: Example of Global declaration part defined by the tool for the main.xml generation depending on the pattern parameters involved in the mission, to declare the maximum number of rows for the parameters

```

DPA > mission_config.csv
1  MODEL TYPE [HF/HL/HRec], NAME FOLDER IN USER, NAME CONFIGURATION FILE
2  HF,HF_1,HF_config.csv
3  HL,HL_2,HL_config.csv
4  HRec,HRec_3,HRec_config.csv

```

Figure 5.11: CSV mission config example. It represents how the patterns in the mission has to be defined, considering its model type, its name and the configuration file associated.

Algorithm 5.1 Automated main.xml code generator

```

1: Input: mission_folder, xml_main_folder, xml_main_name,
           mission_config_path, open_global_dec, global_dec_pre,
           global_dec_ext, close_global_dec, open_sys_dec,
           mission_sys_dec_instances, close_sys_dec, template_place_holders,
           vector_string_main, BB, close_gloDec
2: //Phase 1 - Add the global declaration part at the vector_string_main
   variable
3: //Add pieces of XML file (some fixed taken by inputs and some
   generated by the code from the CSV files) to create the global
   declaration needed to compile the mission UPPAAL file
4: vector_string_main =
   add_string(vector_string_main, f_print_txt_file(open_global_dec))
5: vector_string_main =
   add_string(vector_string_main, f_print_txt_file(global_dec_pre))
6: vector_string_main =
   add_string(vector_string_main, ad_GloDec_sta(global_dec_ext))
7: vector_string_main =
   add_string(vector_string_main, ad_globDec_dyn_format())
8: vector_string_main.push_back(close_gloDec)
9: //Phase 2 - Add the system declaration part at the vector_string_main
   variable
10: //As in the previous phase add pieces of XML file to create the system
    declaration
11: ://Operations to create the vector_string_system_dec variable
12: vector_string_main=
   add_string(vector_string_main, vector_string_system_dec)
13: vector_string_main =
   add_string(vector_string_main, f_print_txt_file(mission_sys_dec_instances)
14: vector_string_main.push_back(close_sys_dec)
15: //Phase 3 - Create the file in the folder with the name set
16: //Generate the main.xml file in the folder declared in the JSON file
17: write_to_file(vector_string_main, absolute(xml_main_folder + BB +
   xml_main_name))

```

tion XML part. This vector is created by functions, which transform the data declared for the parameters in CSV file in UPPAAL syntax. It generates a code as reported in Listing 5.4 (the complete version of the code is illustrated in Appendix A.2). In which every parameter needed to instantiate the pattern is declared. Specifically for the **arg** vector, the **arg_ad** vector, the **SEEA0** matrix, the **obj_xye** matrix the **importance_sense** vector and the **importance_entity** vector for all the patterns added to the mission and defined in the `mission_config.csv` file as explained in Section 5.2.

As explained in Section 5.1, when the `main.xml` file is generated by the automated extended pattern `main.xml` generator, the pre-existing UPPAAL code generator can use it to generate the UPPAAL model. It replaces the placeholder inserted in the `main.xml` to add the parts needed to make the UPPAAL model to describe correctly the desired mission. Specifically, it adds the patterns template (local declaration and the location of the pattern) and their instances. In Listing 5.5, there is an example of instances, in which 2 **Human_Follower** templates of name `h_1` and `h_5` are instantiated. They have different arguments, specifically for the **SEEA0**, **importance_sense**, **importance_entity**, **obj_xye**, **arg** and **arg_ad** parameters. Their addition in the arguments part of the instance is created by the pre-existing code, considering the information inserted in the `mission_config.csv` file. These 2 instances refer to the **Human_Follower** extended pattern with parameters declared by the automated `main.xml` code.

```

1  /* HF_1 */
2  /* args HF_1 */
3  /* vecArg */
4  const double arg_us_HF_1[n_arg_all] = {
5  50.0, /* upd_weight_ref [0] */
6  .....
7  2.0 /* id_person [22]*/
8  };
9  //////////
10 /* args_ad HF_1 */
11 const int arg_ad_HF_1[nr_argAd]={
12 35, /* nr_seeao */
13 .....
14 7 /* nr_obj */
15 };
16 //////////
17 /* matrix seeao num HF_1 */
18 const int SEEAO_HF_1[nr_seeao_max][nc_seeao]={
19 {1,1,1,1,1},
20 .....
21 {0,0,0,0,0}
22 };
23 //////////
24 /* matrix_obj_xye_num HF_1 */
25 const double obj_xye_HF_1[nr_obj_max][nc_obj] = {
26 {400.0,270.0,2.0},
27 .....
28 {4000.0,700.0,3.0}
29 };
30 //////////
31 /* importance_matrix_num HF_1 */
32 /* sense importance matrix */
33 const double importance_sense_HF_1[nr_impSense_max] = {
34 10.0,
35 .....
36 0.0
37 };
38 /* entity importance matrix */
39 const double importance_entity_HF_1[nr_impEntity_max] = {
40 10.0,
41 .....
42 5.0
43 };
44 //////////
45 /*//////// HL_2 //////////*/
46 \* it continues for the other patterns with the same structure of the
   HL_1 but with the possibility to have different parameters.*
47 .....

```

Listing 5.4: Example of parameters declaration in the system declaration

```
1      h_1 = Human_Follower(1, 26.0, 2, -1, -1, -1, arg_us_HF_1,  
    arg_ad_HF_1, SEEA0_HF_1, importance_entity_HF_1,  
    importance_sense_HF_1, obj_xye_HF_1);  
2      .....  
3      h_5 = Human_Follower(5, 26.0, 2, -1, 1, -1, arg_us_HF_5,  
    arg_ad_HF_5, SEEA0_HF_5, importance_entity_HF_5,  
    importance_sense_HF_5, obj_xye_HF_5);
```

Listing 5.5: Example of `Human_Follower` instances with different parameters declared for the 2 patterns, passed as arguments

6 | Experimental Validation

After having defined the structure of the model and implemented it, doing all the needed tests to verify the correctness of the functions developed and their operations, the extended human patterns were used to simulate a scenario, similar to what was simulated with the pre-existing models. To make a comparison in terms of time spent to have a result on the same queries, and to analyse the effects of the additional parameters that do not exist in the pre-existing models, some simulations were done to try to understand what are the effects of these new parameters on the model understanding what is their meaning and what is representing in a real environment.

Hence in this chapter, first is analysed the mission in which the models are used, and then the results obtained from varying different parameters are analysed and lastly, a comparison with the pre-existing models is considered.

6.1. Mission analysed

The mission is operated on the layout illustrated in Figure 6.1, where the entrance and recharge stations are in orange, cupboards are in green (CUP1 and CUP2), examination/waiting room entrances are in red (R1a, R1b, and R2), and doctors' offices in blue (OFF1, OFF2, and OFF3).

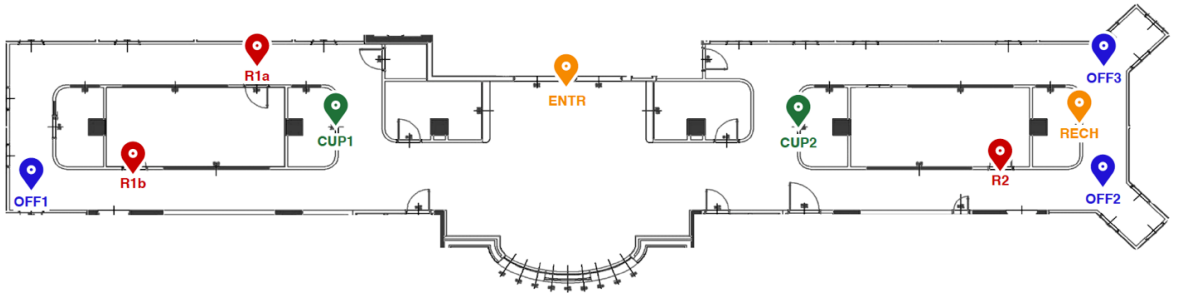


Figure 6.1: Point of interest of the layout from [16]

On the other hand in Figure 6.2 represents the layout abstraction as a set of rectangular areas, with wall size([m]) and intersection points between areas marked by \times symbols.

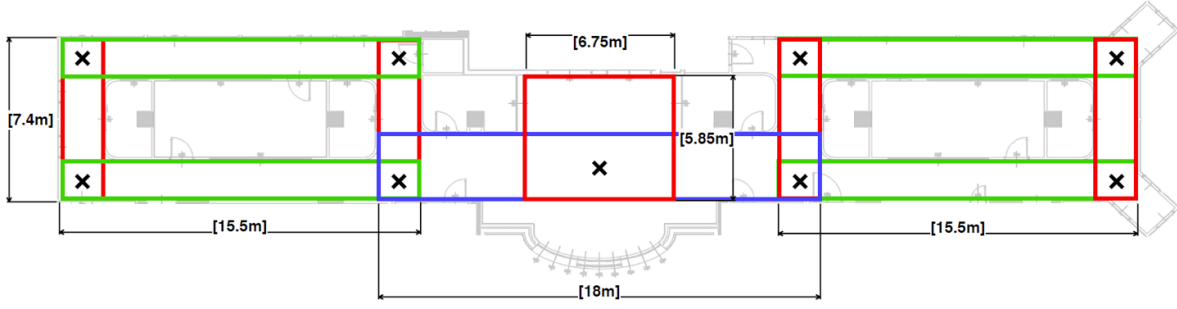


Figure 6.2: Layout abstraction as rectangular areas from [16]

The scenario's mission is defined by the interaction of a robot with a patient and a doctor. The robot (Tbot) serves a patient-doctor pair (Follower/Leader, respectively). The robot meets the patient at the entrance (ENTR) and leads them to the waiting room (R1b) to wait for the doctor to visit them. The robot follows the doctor to CUP1 where they fetch the required tools and follows them back (carrying the tools) to the examination room (R2) where the patient will receive the treatment. Finally, the robot returns to R1b and escorts the patient to R2 where the doctor is waiting.

6.2. Parameters analysed

The focus of the experimental results was based on analysing the effects of the **distance_threshold**, and the **delay** parameters on the mission success probability. Also, the effects of the different number of entities in the environment are analysed, specifically considering the different cases of **case_obj** parameter, when the pattern perceives the fixed object and the robot and when it perceives all the entities in the environment (adding also the humans).

For each analysis, the same parameters (**SEEA0**, **importance_entity**, **importance_sense** and **obj_xye**) for the pattern characterisation were used. To have data depending only on the variations of the previous parameters and not on the human declaration. In the Appendix A.3 are reported all the values of the selected parameters, changing the values of **distance_threshold**, **delay** and **case_obj** for the mission and query analysed (see Table 6.1 and Table 6.1).

6.3. Analysis varying delay and distance parameters

In this section, the results obtained by simulations are reported and discussed. The queries analysed (Table 6.1) are based on the success probability ($\text{Pr}[\leq \tau; 20] (< \text{scs})$) of 20

mission simulations at different time instants (τ), specifically at 300 [s] (Q1), 350 [s] (Q2) and 400 [s] (Q3). For all the analysis the decision-making process defines the decision considering the entities in its neighbour, to make possible the analyses of the model setting the `distance_threshold` parameter.

| Code | Query |
|------|--|
| Q1 | $\text{Pr}[\leq 300; 20](<> \text{scs})$ |
| Q2 | $\text{Pr}[\leq 350; 20](<> \text{scs})$ |
| Q3 | $\text{Pr}[\leq 400; 20](<> \text{scs})$ |

Table 6.1: Queries verified, they represent the properties that are used to verify the mission success probability ($<> \text{scs}$) of 20 simulations at different time instants (300, 350, 400 [s])

The missions analysed are the same as described in Section 6.1 but with different characteristics for the entities perceived by the pattern for the decision-making process. In the $DpaExt_1$ (`case_obj` = 1) mission, patterns consider the fixed objects (7) and the robot (1), while in $DpaExt_2$ (`case_obj` = 3) mission, there is the addition of the other humans (4) (the other patterns), which are involved in the mission. Table 6.2 illustrates a recap of the decision-making process setting for the missions considered.

| Code | Decision-Making setting |
|------------|--|
| $DpaExt_1$ | Mission in which the human pattern perceives the fixed objects (7) and the robot (1) for the decision-making process. <code>case_event</code> = 2, <code>case_obj</code> = 1 and <code>case_fixed_obj</code> = 1 (see Table A.1) |
| $DpaExt_2$ | Mission in which the human pattern perceives the fixed objects (7), the robot (1) and the humans (4) for the decision-making process. <code>case_event</code> = 2, <code>case_obj</code> = 3 and <code>case_fixed_obj</code> = 1 (see Table A.1) |

Table 6.2: Recap Decision-Making setting for the missions analysed.

In the **first analysis (A1)** (Table 6.3) the `distance_threshold` is set up to 400 [cm], while the `delay` varies its values in the time intervals [1,150] [s]. The entities used to make a decision are the fixed objects considered by their presence in the scenario and by the robot that interacts with the humans ($DpaExt_1$). Looking at the data obtained by this analysis reported in Table 6.4, it is possible to say that the highest mission probability is

| Code | Name | Mission | Parameters |
|------|-----------------|------------|---|
| A1 | First Analysis | $DpaExt_1$ | $delay \in [1, 150]$ [s] $distance_threshold = 400$ [cm] |
| A2 | Second Analysis | $DpaExt_2$ | $delay \in [1, 150]$ $distance_threshold = 400$ [cm] |
| A3 | Third Analysis | $DpaExt_1$ | $delay = 20$ [s] $distance_threshold \in [25, 2000]$ [cm] |
| A4 | Fourth Analysis | $DpaExt_2$ | $delay = 20$ [s] $distance_threshold \in [25, 2000]$ [cm] |

Table 6.3: Recap analyses.

obtained for the **Q3** query, in which it overcomes the 50% for different values of the **delay** (e.g. for **delay** $\in [20, 100]$), while for the same **delay** interval for the **Q2** and **Q1** queries do not reach in all values the 40%. Hence, considering the data related to the **Q3** query, it is possible to say that for lower values of **delay** (up to 12 [s]), the success probability, is worst concerning the other **delay** values and that the success reduces again when this parameter increases over 80 [s]. This is because, for **delay** ≤ 12 , the success probably does not reach the 30% (e.g. for 1, 6 and 12 [s]), while for **delay** $\in (12, 80]$, the probably reaches values over the 50% (e.g. for 20, 40, 80 [s]). With values of **delay** greater than 100 [s], the success probability decreases and for **delay** = 150 [s], there is a 31 % of probability of success. This is also confirmed considering the average success probability of the maximum values of the success probability for each query, in **Q3** 41 %, in **Q2** 28 % and in **Q1** 21 %. This data can characterise different human characters, specifically, 2 opposite human behaviours: undecided when the human continuously changes their decision (**delay** < 12 [s]); decided, when the action decided lasts for a long time (**delay** > 80 [s]). Analysing the data between these 2 behaviours it is possible to say that for **delay** $\in [12, 80]$ ordinary people's behaviour is represented.

The **second analysis** has the same objective as the first one but in this case, also the humans in the scenario are considered for the human decision (**case_obj** = 3). This is done to make a comparison and analyse what may happen if humans have more distractions (more entities on which operates their decisions). In Table 6.5, the data of this second analysis are reported. Also in this case it is possible to notice the same behaviours that were defined in the previous analysis (A1) are present. Specifically, the mission probabilities between the queries. Also, in this case, the **Q3** query reaches the highest values of probabilities over the **delay** a lot of success probabilities are greater than the 40%, while in the **Q2** all the values do not reach it (37% is the maximum), and in the

| <i>Delay</i> [s] | Q1 | Q2 | Q3 |
|------------------|---------------------|----------------------|---------------------|
| 1 | [0,0.139108] | [0,0.139108] | [0,0.139108] |
| 6 | [0,0.139108] | [0.001265,0.248733] | [0.001265,0.248733] |
| 10 | [0,0.139108] | [0.001265,0.248733] | [0.032071,0.378927] |
| 12 | [0,0.139108] | [0.001265,0.248733] | [0.001265,0.248733] |
| 20 | [0.012349,0.316983] | [0.086572,0.491046] | [0.118932,0.542789] |
| 40 | [0.012349,0.316983] | [0.032071,0.378927] | [0.153931,0.592189] |
| 60 | [0,0.139108] | [0.012349,0.316983] | [0.57334,0.436614] |
| 80 | [0.032071,0.378927] | [0.032071,0.378927] | [0.153931,0.592189] |
| 100 | [0.001265,0.248733] | [0.0010265,0.248733] | [0.086572,0.491046] |
| 150 | [0,0.139108] | [0.001265,0.248733] | [0.012349,0.316983] |

Table 6.4: Data related to the First Analysis (A1) performed on *DpaExt*₁, with **distance_threshold** = 400 [cm], and **delay** \in [1,150] [s], for the intervals of probabilities for queries **Q1**, **Q2** and **Q3** (see Table 6.1).

Q1 only for the **delay** = 20 [s], there is a success probability greater than 40%. This is also confirmed considering the average success probability of the maximum values of the success probability for each query, in **Q3** 34 %, in **Q2** 30 % and in **Q1** 25 %. Hence, also in this analysis, the maximum probabilities of the **Q3** are considered to understand the pattern behaviour when the **delay** parameter increases. The data has more or less the same structure as analysis 1, in which it is possible to divide into 3 intervals the success probabilities into. In this analysis, for **delay** lower than 10 [s], the probability does not overcome the 24%, which is reached and exceeded by the success probabilities for the **delay** \in [12,80][s] (over the 40%), while it returns to decreases for **delay** greater than 80 [s]. For this mission, it seems that the model has an undecided behaviour when the **delay** < 10 [s], ordinary in [12,60] [s] and undecided for **delay** > 80 [s], hence concerning the A1 analysis of the pattern behaviours are translated to lower values of **delay** parameter.

In Table 6.6, there is a comparison between the first (A1) and second (A2) analyses considering the intervals of probabilities obtained for the **Q3** query over the **delay** parameter values. It is possible to notice how the probabilities of success in the second analysis are lower than the values obtained in the first analysis (A1). The average probability value computed for the **Q3** query for the comparison between the queries returns that the average value for A2 is 34 %, while is 41 % for A1. This can be interpreted that when the pattern considers for the decision-making process more entities when the distance

threshold is fixed and the **delay** varies the success probability decreases.

| <i>Delay</i> [s] | Q1 | Q2 | Q3 |
|------------------|---------------------|---------------------|---------------------|
| 1 | [0,0.139108] | [0,0.139108] | [0,0.139108] |
| 6 | [0,0.139108] | [0.001265,0.248733] | [0,0.248733] |
| 10 | [0,0.139108] | [0.001265,0.248733] | [0,0.139108] |
| 12 | [0.123485,0.316983] | [0.030271,0.378927] | [0.086572,0.491046] |
| 20 | [0.057334,0.436614] | [0.057334,0.08572] | [0.086572,0.491046] |
| 40 | [0,0.139108] | [0.032071,0.378927] | [0.032071,0.378927] |
| 60 | [0.032071,0.378927] | [0.032071,0.378927] | [0.118932,0.542789] |
| 80 | [0.012349,0.316983] | [0.032071,0.378927] | [0.057334,0.436614] |
| 100 | [0.001265,0.248733] | [0,0.139108] | [0.001265,0.248733] |
| 150 | [0.001265,0.247833] | [0.001265,0.248733] | [0.032071,0.378927] |

Table 6.5: Data related to the Second Analysis (A2) performed on *DpaExt₂*, with **distance_threshold** = 400 [cm], and **delay** \in [1,150] [s], for the intervals of probabilities for queries **Q1**, **Q2** and **Q3** (see Table 6.1).

The **third** (Table 6.7) and the **fourth analyses** (Table 6.8) have the objective to analyse what happens when the **delay** parameter is considered fixed at 20 [s], and the **distance_threshold** can vary in a length [cm] interval of [25,2000]. In the third analysis as in the first one, the human can make a decision considering the fixed objects and the robot in the scenario. From the data collected for this analysis (Table 6.7). It is possible to notice, as it was in the previous analysis, that the success probability for the **Q3** query is greater concerning the others. In fact for values of **distance_threshold** greater than 300 [cm] the maximum success probability of **Q3** is greater concerning the others, only for **distance_threshold** = 1000 [cm] the **Q2** value is greater than the **Q3** value. Also considering the average success probability of the maximum value for the queries, it is possible to obtain that in **Q3** is 46 %, in **Q2** is 36 % while in **Q1** is 26 %. Hence considering the data related to the **Q3** query, it is possible to say that for low values (< 600 [cm]) of the **distance_threshold**, there are higher success probabilities concerning greater values (≥ 600 [cm]), specifically, the success probabilities for the first case is greater than the 50%, while for the second, it does not reach the 40% and it decreases. This can be explained by the fact that with a greater **distance_threshold** more entities define the decision of the human, and for that, the decisions may not be focused only on the action declared for the interaction with the robot, and so it is possible to say that the human is more distracted with external entities and not be strictly related to the normal mission

| <i>Delay</i> [s] | Q3 A1 | Q3 A2 |
|------------------|---------------------|---------------------|
| 1 | [0,0.139108] | [0,0.139108] |
| 6 | [0.001265,0.248733] | [0,0.248733] |
| 10 | [0.032071,0.378927] | [0,0.139108] |
| 12 | [0.001265,0.248733] | [0.086572,0.491046] |
| 20 | [0.118932,0.542789] | [0.086572,0.491046] |
| 40 | [0.153931,0.592189] | [0.032071,0.378927] |
| 60 | [0.57334,0.436614] | [0.118932,0.542789] |
| 80 | [0.153931,0.592189] | [0.057334,0.436614] |
| 100 | [0.086572,0.491046] | [0.001265,0.248733] |
| 150 | [0.012349,0.316983] | [0.032071,0.378927] |

Table 6.6: Comparison of mission probabilities between first (A1) and second analysis (A2) for the probabilities of Q3 query (see Table 6.1).

development.

The fourth analysis (A4) concerning the third (A3), considers also the human involved in the decision-making process. Analysing the data reported in Table 6.8, it is possible to say that the success probability is greater in the Q3 query with respect to the others, as happened also in the others analyses. Specifically for the values of `distance_threshold` > 300 [cm], in fact considering the average probability for the 3 queries, it is obtained that for Q3 is 44 %, for Q2 37 % and 30 % for Q1. Hence considering the data related to the `distance_threshold` ≤ 600 [cm], the success probability is greater with respect to the values of the `distance_threshold` > 600 [cm], because in the first case, the success probability is greater than the 40 % with a pick of 64 % while in the second the maximum value is 25 %. This can be interpreted as it was done in the third analysis, hence when the `distance_threshold` the neighbour of the pattern is greater and more entities can be considered the success probability decreases. The comparison of data between the third and fourth analyses of the success probability values for the Q3 query are compared. Considering the average of the maximum success probabilities defined above it is possible to say the third analysis obtained a higher success probability of a 46 % with respect to 44% of the fourth analysis. This makes it possible to state that the decision-making process reduces its success probability when the human is more distracted (more entities, third analysis), concerning the case of fewer entities, when it is less distracted.

| <i>Distance</i> [cm] | Q1 | Q2 | Q3 |
|----------------------|-------------------|--------------------|----------------------|
| 25 | [0,0.13911] | [0,0.13911] | [0,0.13911] |
| 50 | [0.19119,0.63957] | [0.31528, 0.76942] | [0.19119,0.630457] |
| 100 | [0.01235,0.31698] | [0.08657,0.49105] | [0.153909,0.5921189] |
| 200 | [0.05733,0.43661] | [0.27196,0.72804] | [0.271958,0.728042] |
| 300 | [0.08657,0.49205] | [0.15391,0.59219] | [0.315278,0.769422] |
| 400 | [0.01235,0.31698] | [0.08657,0.49105] | [0.153909,0.592189] |
| 500 | [0.00123,0.24873] | [0.05733,0.43661] | [0.230578,0.684722] |
| 600 | [0,0.13911] | [0,0.13911] | [0.0865715,0.491046] |
| 700 | [0.01235,0.31698] | [0,0.13911] | [0.032009,0.378927] |
| 1000 | [0,0.13911] | [0.01235,0.31698] | [0.001209,0.248733] |
| 2000 | [0,0.13911] | [0.00127,0.24873] | [0.012485,0.316983] |

Table 6.7: Data related to the Third Analysis (A3) performed on $DpaExt_1$, with `delay` = 20 [s], and `distance_threshold` \in [25, 2000] [cm], for the intervals of probabilities for queries Q1, Q2 and Q3 (see Table 6.1).

6.4. Comparison with pre-existing models

The comparison data is based on the values obtained by the success probabilities varying the time (τ) required for the verification of $\text{Pr}[\leq \tau(\langle \rangle \text{ scs})]$ (Table 6.10). In Table 6.11, there are different data collected to make a comparison between the mission defined in the previous analysis ($DpaExt_1$ and $DpaExt_2$) and data made without using the extended human pattern (Dpa). As expected the success probability is higher in the query that has the highest value of τ , that is 400 [s] concerning other τ values (350 [s] and 300 [s]). To analyse the data between the mission the property is considered with the $\tau = 400$ [s]. In this case, the success probability for the pre-existing mission (Dpa) has the highest mission probability value equal to 0.933 ± 0.05 (93 % \pm 5 %) concerning the other missions, 0.556 ± 0.05 (56 % \pm 5 %) for $DpaExt_1$ and 0.500 ± 0.05 (50 % \pm 5%) for $DpaExt_2$. This may be possible because the extended pattern can consider behaviours that the pre-existing patterns can not consider like the time between 2 decisions and the distance at which the entities of the environment are perceived for the decision-making process, making the pattern depending on what there are close to it, during its motion over the environment. In addition, focusing on the $DpaExt_1$ and $DpaExt_2$, as explained in Section 6.3, the success probabilities are greater when the pattern is less distracted

| <i>Distance [cm]</i> | Q1 | Q2 | Q3 |
|----------------------|-------------------|-------------------|--------------------|
| 25 | [0,0.13911] | [0,0.13911] | [0,0.13911] |
| 50 | [0.11893,0.54279] | [0.27196,0.72804] | [0.360543,0.80881] |
| 100 | [0.03207,0.37893] | [0.01235,0.31698] | [0.057334,0.37893] |
| 200 | [0.15391,0.59219] | [0.36054,0.80881] | [0.315378,0.68472] |
| 300 | [0.01235,0.31698] | [0.08657,0.49105] | [0.271958,0.72804] |
| 400 | [0.00127,0.24873] | [0.01235,0.31698] | [0.11893,0.54279] |
| 500 | [0.05733,0.43661] | [0.10123,0.31698] | [0.05733,0.43661] |
| 600 | [0,0.13911] | [0.03207,0.37983] | [0.11893,0.54279] |
| 700 | [0.00127,0.24873] | [0,0.13911] | [0.00127,0.24873] |
| 1000 | [0,0.13911] | [0.00127,0.24873] | [0.00127,0.24873] |
| 2000 | [0,0.13911] | [0.00127,0.24873] | [0,0.13911] |

Table 6.8: Data for the fourth analysis (A4, `distance_threshold` \in [25, 2000] [cm] and `delay` = 20 [s]) for the probabilities intervals of the Q1, the Q2 and the Q3 queries (see Table 6.1).

| <i>Distance [cm]</i> | Q3 A3 | Q3 A4 |
|----------------------|----------------------|---------------------|
| 25 | [0,0.13911] | [0,0.13911] |
| 50 | [0.19119,0.63046] | [0.360543,0.80881] |
| 100 | [0.153909,0.5921189] | [0.057334,0.37893] |
| 200 | [0.271958,0.728042] | [0.23058,0.684722] |
| 300 | [0.315278,0.769422] | [0.271958,0.728042] |
| 400 | [0.153909,0.592189] | [0.11893,0.54279] |
| 500 | [0.230578,0.684722] | [0.05733,0.43661] |
| 600 | [0.0865715,0.491046] | [0.11893,0.54279] |
| 700 | [0.03207,0.37893] | [0.00127,0.24873] |
| 1000 | [0.00127,0.24873] | [0.00127,0.24873] |
| 2000 | [0.01235,0.31698] | [0,0.13911] |

Table 6.9: Comparison of mission probabilities for the Q3 query ($\text{Pr}[\leq 400; 20] (> \text{scs})$) between third (A3) and fourth analysis (A4), with `distance_threshold` \in [25, 2000] [cm] and `delay` = 20 [s]

(fewer entities involved $DpaExt_1$), concerning what happens in the $DpaExt_2$, in which the entities considered are 12 (7 fixed objects, 1 robot and 4 humans) instead of 8 (7 fixed objects and 1 robot). Considering the query with $\tau = 400$ [s], the success probabilities are respectively, 0.556 ± 0.05 (56 % \pm 5 %) and 0.500 ± 0.05 (50 % \pm 5%). Also, the time needed for the verification is lower in this case, 44.45 [min] concerning 51.02 [min]. This highlights that in the case of more entities, the success probability decreases while time increases. In both $DpaExt_1$ and $DpaExt_2$, the **distance_threshold** and **delay** were set equal, respectively, 200 [cm] and 20 [s], because the behaviours that it is decided to give to the pattern was related to ordinary people and with a distance that can be feasible in the environment.

| Query |
|--------------------------------------|
| $\text{Pr}[\leq 300](<> \text{scs})$ |
| $\text{Pr}[\leq 350](<> \text{scs})$ |
| $\text{Pr}[\leq 400](<> \text{scs})$ |

Table 6.10: Queries verified for the comparison with the pre-existing models without considering the number of runs over the success probability should be computed, it is defined by UPPAAL, and can be different mission by mission

| Mission | τ | Ver.Time [min] | Success Probability |
|------------|--------|----------------|---------------------|
| Dpa | 400 | 16.36 | 0.933 ± 0.05 |
| | 350 | 54.67 | 0.706 ± 0.05 |
| | 300 | 59.09 | 0.419 ± 0.05 |
| $DpaExt_1$ | 400 | 44.45 | 0.556 ± 0.05 |
| | 350 | 41.30 | 0.463 ± 0.05 |
| | 300 | 34.04 | 0.340 ± 0.05 |
| $DpaExt_2$ | 400 | 51.02 | 0.500 ± 0.05 |
| | 350 | 44.12 | 0.467 ± 0.05 |
| | 300 | 34.36 | 0.313 ± 0.05 |

Table 6.11: Results comparison for the verification time and the success probability of the $\text{Pr}[\leq \tau](<> \text{scs})$ for the pre-existing mission (Dpa) and the extended pattern mission ($DpaExt_1$ and $DpaExt_2$)

6.5. Discussion

The collected data from the different analyses involved in this chapter shows how the new extended patterns can describe different human behaviours, for example, if a human being

is more determined or more distracted. This extends the verification results to missions where human behaviour must be set with more details. Even if these data show that the mission success probability decreases when the human is more distracted or undecided, they need to be validated also in reality, taking data from the same mission with the same actors involved in the framework mission. This step is not always possible given the complexity to recreate the desired mission in reality. Hence, the data obtained should be considered at this moment only indicative and they can not be considered a complete representation of what may happen in reality. This can be also emphasised by the fact that the human layered cognitive decision-making model does not cover the entire subconscious part of the human being, that is the unpredictable part of the human, to the fact that is strictly related to the human self and not general for all, as the conscious part can be. It is necessary to highlight that there are some limitations in its ability to accurately and fully represent human decision-making. It cannot, for example, take into account all the factors that influence human decisions, such as emotions or social influences. Even if the model has the probabilistic characteristic, it can reach a high accuracy but it is not possible to obtain every time the actual human decision. To improve this model, it might be useful to conduct further research to better understand human decision-making and to identify any gaps or limitations. Specifically, it could be possible (if a data set of this type exists) to compare the framework verification result with a specific known mission and improve the model until the results are not close to the mission data, obtained in reality.

7 | Conclusions

Through this work, a more sophisticated decision-making model was developed by adding the cognitive characteristic. The implementation was accomplished with the encoding of the Layered Reference Model of the Brain theory in UPPAAL. This creates a decision-making model with a set of parameters that help to specify human characteristics that were not possible to be described by the pre-existing decision-making model. In fact, with the new decision-making model is possible to consider different human behaviours, such as being distracted or not distracted. This makes verification results to be dependent on the more specific situation. The designer can specify how that human should behave, so how it may think in the mission. The new decision-making model is a starting point to help the mission designer be able to specify more clearly the scenario and obtain verification results closer to what may happen in reality.

The automated generation of the UPPAAL model is a necessary step to make a framework useful to be used also by users that are not close to the verification tools as UPPAAL SMC is. In this thesis, an automated generation was implemented to achieve this objective, it was developed to be the simpler way to declare the human parameters, considering CSV files, that are possible to be created on every pc with a text editor.

The decision-making model developed has the limit to not fully describing and representing a mission where the environment can change over time. This is given by the fact that the parameters remain fixed. Adding dynamic parameters which may change depending on what may happen in the environment permits analysing a more complex scenario where a human may change their behaviour. For example, if the environment becomes dark because there is the possibility that it may happen in the scenario, the human does not have more sight sense but only touch and hearing, so the possible decision may change for this event. On the other hand, it may be possible to consider the importance of an entity changes over time, for example, if the fatigue of the human increases the human may give to a chair concerning a robot concerning human is in a mental state where it should stop and rest. Moreover, as with any model, there are some limitations in its ability to accurately and fully represent human decision-making. It is not able, for example,

to take into account all the factors that influence human decisions, such as emotions or social influences. Even if the model has the probabilistic characteristic, it can reach a high accuracy but it is not possible to obtain every time the actual human decision. To improve this model, it might be useful to conduct further research to better understand human decision-making and to identify any gaps or limitations. Specifically to have access to the data set of HRI, to declare also by the designer side better weights for the entities in the environment, to get verification results that are close to the reality. To simulate robotics applications with more accuracy and be able to improve robot development by having simulated data representing reality.

Bibliography

- [1] G. Agha and K. Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1):1–39, 2018.
- [2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. ISSN 0304-3975. doi: [https://doi.org/10.1016/0304-3975\(94\)00202-T](https://doi.org/10.1016/0304-3975(94)00202-T). URL <https://www.sciencedirect.com/science/article/pii/030439759400202T>. Hybrid Systems.
- [3] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.
- [4] J. R. Anderson. Act: A simple theory of complex cognition. *American psychologist*, 51(4):355, 1996.
- [5] M. Askarpour. How to formally model human in collaborative robotics. *arXiv preprint arXiv:2012.01647*, 2020.
- [6] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass. A systematic approach to model checking human–automation interaction using task analytic models. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(5):961–976, 2011.
- [7] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.
- [8] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. Uppaal smc tutorial. *International journal on software tools for technology transfer*, 17:397–415, 2015.
- [9] F. De Felice, A. Petrillo, and F. Zomparelli. A hybrid model for human error probability analysis. *IFAC-PapersOnLine*, 49(12):1673–1678, 2016. ISSN 2405-8963. doi: <https://doi.org/10.1016/j.ifacol.2016.07.821>. URL <https://www.sciencedirect.com/science/article/pii/S2405896316300000>.

- [com/science/article/pii/S2405896316311041](https://www.sciencedirect.com/science/article/pii/S2405896316311041). 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.
- [10] C. B. Frey and M. A. Osborne. The future of employment: How susceptible are jobs to computerisation? *Technological Forecasting and Social Change*, 114:254–280, 2017. ISSN 0040-1625. doi: <https://doi.org/10.1016/j.techfore.2016.08.019>. URL <https://www.sciencedirect.com/science/article/pii/S0040162516302244>.
 - [11] U. Grenander. Stochastic processes and statistical inference. *Arkiv för matematik*, 1(3):195–277, 1950.
 - [12] H. R. Hartson, A. C. Siochi, and D. Hix. The uan: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems (TOIS)*, 8(3):181–203, 1990.
 - [13] D. J. Jilk, C. Lebiere, R. C. O’Reilly, and J. R. Anderson. Sal: An explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 20(3):197–218, 2008.
 - [14] J. E. Laird. *The Soar cognitive architecture*. MIT press, 2019.
 - [15] L. Lestingi, M. Askarpour, M. M. Bersani, and M. Rossi. A model-driven approach for the formal analysis of human-robot interaction scenarios. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1907–1914. IEEE, 2020.
 - [16] L. Lestingi, D. Zerla, M. M. Bersani, and M. Rossi. Specification, stochastic modeling and analysis of interactive service robotic applications. *Robotics and Autonomous Systems*, page 104387, 2023.
 - [17] F. Paternò. Concurtasktrees: an engineered notation for task models. *The handbook of task analysis for human-computer interaction*, pages 483–503, 2004.
 - [18] S. Pocock, P. Wright, and M. Harrison. Thea-a technique for human error assessment early in design. Technical report, YORK UNIV (UNITED KINGDOM) DEPT OF COMPUTER SCIENCE, 2001.
 - [19] A. D. Swain and H. E. Guttman. Handbook of human-reliability analysis with emphasis on nuclear power plant applications. final report. 8 1983. doi: 10.2172/5752058. URL <https://www.osti.gov/biblio/5752058>.
 - [20] U. University and A. University. UPPAAL, 2008. URL <https://uppaal.org/>.
 - [21] Y. Wang, S. Patel, D. Patel, and Y. Wang. A layered reference model of the brain. In *The Second IEEE International Conference on Cognitive Informatics, 2003. Pro-*

- ceedings.*, pages 7–17. IEEE, 2003. URL <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1225945>.
- [22] Y. Wang, Y. Wang, S. Patel, and D. Patel. A layered reference model of the brain (lrmb). *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(2):124–133, 2006. URL <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1624538>.
- [23] J. Williams. A data-based method for assessing and reducing human error to improve operational performance. In *Conference Record for 1988 IEEE Fourth Conference on Human Factors and Power Plants*,, pages 436–450, 1988. doi: 10.1109/HFPP.1988.27540.

A | Appendix A

A.1. All Model parameters

Table A.1: Cases Parameters

| Name parameter | Value | Effects |
|---------------------------|-------|---|
| <i>case_upd_weight</i> | 0 | Deactivate the case |
| | 1 | To modify the action weights for the actions stored in stm and ltm vectors at every extraction |
| | 2 | To modify the action weights for the actions stored in stm and ltm vectors randomicaly |
| <i>case_weight_entity</i> | 0 | Nothing, the entity weight sum is equal to 1 |
| | 1 | Every entity has a desired weight to give to them different importance declared in the importance_entity vector |
| <i>case_weight_sense</i> | 0 | Nothing, the sense weight sum is equal to 1 |
| | 1 | Every sense has a desired weight to give to them different importance declared in the importance_sense vector |
| <i>case_event</i> | 1 | event_may_happen vector is randomly initialised every time |
| | 2 | event_may_happen vector is set depending on the distance of the entities from the human |

Continue on next page

Continue from the previous page

| Name parameter | Value | Effects |
|---------------------|-------|---|
| <i>case_last_id</i> | 0 | The last_id_output is equal to the last_id_extracted_output at every occasion |
| | 1 | The last_id_output is randomly selected between 1 to 4 at every occasion |
| | 2 | The initial_id is equal to last_id_extracted_output, for next extraction the last_id_out is equal to the last_id_extracted |
| | 3 | The initial_id is equal to last_id_extracted_output, for next extraction the last_id_out is randomly selected between 1 and 4 |
| <i>case_obj</i> | 0 | The pattern for the decision-making considers only the fixed objects |
| | 1 | The pattern for the decision-making considers the fixed (if selected) objects and robots |
| | 2 | The pattern for the decision-making considers the fixed (if selected) objects and Humans |
| | 3 | The pattern for the decision-making considers the fixed (if selected) objects the robots and Humans |
| <i>case_stl_ltm</i> | 0 | The lost of memory is not considered |
| | 1 | The lost of memory is random |
| | 2 | The lost of memory is FIFO and in a random cycle of decision-making |
| | 3 | The lost of memory is FIFO |

Continue on next page

Continue from the previous page

| Name parameter | Value | Effects |
|----------------------|-------|---|
| <i>case_obj_cons</i> | 0 | The fixed objects are not considered also in the case they should be considered to have the case of only robots, only humans or robots and humans |
| | 1 | The fixed objects are considered in the cases of case_obj |

Ends from the previous page

Table A.2: Pattern parameters

| Name parameter | Value | Effects |
|---------------------------------|-------|---|
| <i>upd_weight</i> | > 0 | It is used to give a probably weight for the <i>case_upd_weight</i> = 2 |
| <i>stm_weight</i> | > 0 | weigh to sum at the action stored in the stm |
| <i>ltm_weight</i> | > 0 | weigh to sum at the action stored in the ltm |
| <i>delay</i> | > 0 | It is used to set the time interval between 2 decisions |
| <i>last_id_extracted_output</i> | > 0 | It is used to set the id of the last output for the starting extraction and in case of no presence of event |
| <i>distance_threshold</i> | > 0 | It is used to set the distance at which the entity is considered for the decision-making process |
| <i>n_stm</i> | > 0 | It is used to set the max of action that the pattern can memorize in the stm |
| <i>n_ltm</i> | > 0 | It is used to set the max of action that the pattern can memorize in the ltm |
| <i>id_robot</i> | > 0 | It is used to set the id that the user considers for the robot |

Continue on next page

Continue from the previous page

| Name parameter | Value | Effects |
|---------------------|----------|--|
| <i>id_walk</i> | > 0 | It is used to set the id that the user considers for the walk action |
| <i>id_sit</i> | > 0 | It is used to set the id that the user considers for the sit action |
| <i>id_run</i> | > 0 | It is used to set the id that the user considers for the run action |
| <i>id_stand</i> | > 0 | It is used to set the id that the user considers for the stand action |
| <i>id_human</i> | > 0 | It is used to set the id that the user considers for the human |
| <i>distance_min</i> | ≥ 0 | It is used to set the minimum distance at which the entity is considered for the decision-making process |

Ends from the previous page

A.2. System declaration example

```

/* HF_1 */
/* args HF_1 */
// vecArg
const double arg_us_HF_1[n_arg_all] = {
50.0, /* upd_weight_ref [0] */
0.0, /* case_upd_weight_ref [1]*/
1.0, /* stm_weight_ref [2]*/
2.0, /* ltm_weight_ref [3]*/
1.0, /* case_weight_entity_ref [4]*/
1.0, /* case_weight_sense_ref [5]*/
2.0, /* case_event_ref [6]*/
1.0, /* case_fixed_obj [7]*/
1.0, /* n_delay_index_ref [8]*/
1.0, /* last_id_extracted_output_ref [9]*/
2.0, /* case_last_id_ref [10]*/

```



```

0.0, /* case_obj_ref [11]*/
0.0, /* distance_min [12]*/
400.0, /* distance_threshold_ref [13]*/
3.0, /* case_stm_ltm_ref [14]*/
10.0, /* n_stm_ref [15]*/
6.0, /* n_ltm_ref [16]*/
1.0, /* id_robot [17]*/
1.0, /* id_walk [18]*/
2.0, /* id_sit [19]*/
3.0, /* id_run [20]*/
4.0, /* id_stand [21]*/
2.0 /*id_person [22]*/
};
//////////
/* args_ad HF_1 */
const int arg_ad_HF_1[nr_argAd]={
35, /* nr_seeao*/
1, /* n_sense*/
11, /* n_entity*/
11, /* n_event*/
20, /* n_action*/
4, /* n_output*/
7 /* nr_obj*/
};
//////////
/* matrix seeao num HF_1 */
const int SEEAO_HF_1[nr_seeao_max][nc_seeao]={
{1,1,1,1,1},
{1,1,1,2,4},
{1,1,1,3,2},
{1,1,1,5,4},
{1,1,1,4,1},
{1,2,2,1,1},
{1,2,2,2,4},
{1,2,2,3,2},
{1,2,2,5,4},
{1,3,3,1,1},

```

[illegible]

```

{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0},
{0,0,0,0,0}
};

//////////
/* matrix_obj_xye_num HF_1*/
const double obj_xye_HF_1[nr_obj_max][nc_obj] = {
{400.0,270.0,2.0},
{4000.0,270.0,2.0},
{1400.0,450.00,5.0},
{1200.0,680.0,2.0},
{3000.0,450.0,5.0},
{200.0,200.0,3.0},
{4000.0,700.0,3.0}
};

//////////
/* importance_matrix_num HF_1*/
/* sense importance matrix */
const double importance_sense_HF_1[nr_impSense_max] = {
10.0,
10.0,
10.0,
0.0,
0.0
};

/* entity importance matrix */
const double importance_entity_HF_1[nr_impEntity_max] = {
10.0,
10.0,
10.0,

```

```

5.0,
5.0,
5.0,
5.0,
5.0,
5.0,
5.0,
5.0
};
//////////
/*///// HL_2  /////*/
.  \\ it continues for the other patterns with the same structure of the
    HL\_1 but with the possibility to have different parameters.
.

```

A.3. Analysis Parameters

```

%
/* HF_1 */
/* args HF_1 */
// vecArg
const double arg_us_HF_1[n_arg_all] = {
50.0, /* upd_weight_ref [0] */
0.0, /* case_upd_weight_ref [1]*/
1.0, /* stm_weight_ref [2]*/
2.0, /* ltm_weight_ref [3]*/
1.0, /* case_weight_entity_ref [4]*/
1.0, /* case_weight_sense_ref [5]*/
2.0, /* case_event_ref [6]*/
1.0, /* case_fixed_obj [7]*/
1.0, /* n_deelay_index_ref [8]*/
1.0, /* last_id_extracted_output_ref [9]*/
2.0, /* case_last_id_ref [10]*/
1.0, /* case_obj_ref [11]*/
0.0, /* distance_min [12]*/
400.0, /* distance_threshold_ref [13]*/
3.0, /* case_stm_ltm_ref [14]*/

```

```

10.0, /* n_stm_ref [15]*/
6.0, /* n_ltm_ref [16]*/
1.0, /* id_robot [17]*/
1.0, /* id_walk [18]*/
2.0, /* id_sit [19]*/
3.0, /* id_run [20]*/
4.0, /* id_stand [21]*/
2.0 /*id_person [22]*/
};
//////////
/* args_ad HF_1*/
const int arg_ad_HF_1[nr_argAd]={
35, /* nr_seeao*/
1, /* n_sense*/
11, /* n_entity*/
11, /* n_event*/
20, /* n_action*/
4, /* n_output*/
7 /* nr_obj*/
};
////////////////////////////////////
/*matrix seeao num HF_1*/
const int SEEAO_HF_1[nr_seeao_max][nc_seeao]={
{1,1,1,1,1},
{1,1,1,2,4},
{1,1,1,3,2},
{1,1,1,5,4},
{1,1,1,4,1},
{1,2,2,1,1},
{1,2,2,2,4},
{1,2,2,3,2},
{1,2,2,5,4},
{1,3,3,1,1},
{1,3,3,2,4},
{1,3,3,3,2},
{1,3,3,5,4},
{1,4,4,1,1},

```

{1,4,4,6,2},
{1,4,4,3,4},
{1,5,5,1,1},
{1,6,6,7,1},
{1,6,6,15,4},
{1,6,6,8,1},
{1,6,6,9,4},
{1,7,7,10,4},
{1,7,7,11,1},
{1,7,7,12,4},
{1,8,8,7,1},
{1,8,8,15,4},
{1,8,8,8,3},
{1,8,8,9,4},
{1,9,9,10,4},
{1,9,9,11,1},
{1,9,9,12,4},
{1,10,10,13,3},
{1,10,10,14,4},
{1,11,11,19,4},
{1,11,11,20,1},
{2,1,12,1,1},
{2,1,12,2,4},
{2,1,12,3,4},
{2,1,12,5,4},
{2,1,12,4,1},
{2,2,13,1,1},
{2,2,13,2,4},
{2,2,13,3,4},
{2,2,13,5,2},
{2,3,14,1,1},
{2,3,14,2,4},
{2,3,14,3,4},
{2,3,14,5,4},
{2,3,14,16,1},
{2,4,15,1,1},
{2,4,15,6,2},

```

{2,4,15,3,4},
{2,5,16,1,1},
{3,10,17,13,3},
{3,10,17,14,4},
{3,11,18,18,4},
{3,11,18,17,1}
};

////////////////////////////////////
/*matrix_obj_xye_num HF_1*/
const double obj_xye_HF_1[nr_obj_max][nc_obj] = {
{400.0,270.0,2.0},
{4000.0,270.0,2.0},
{1400.0,450.00,5.0},
{1200.0,680.0,2.0},
{3000.0,450.0,5.0},
{200.0,200.0,3.0},
{4000.0,700.0,3.0}
};

////////////////////////////////////
/*importance_matrix_num HF_1*/
/*sense importance matrix*/
const double importance_sense_HF_1[nr_impSense_max] = {
10.0,
10.0,
10.0,
10.0,
10.0
};

/*entity importance matrix*/
const double importance_entity_HF_1[nr_impEntity_max] = {
10.0,
10.0,
10.0,
5.0,
5.0,
5.0,
5.0,

```

```

5.0,
5.0,
5.0,
5.0
};
////////////////////////////////////
/*//////// HL_2 //////////*/
.  \\ it continues for the other patterns with the same structure of the
    HL\_1
.
.

```

A.4. Function Algorithms

This section reports the description of the remaining functions not described in the text in Section 4.1.2. Specifically the functions for the extractions the *entity_id_extraction()* (Algorithm A.1), *event_id_extraction()* (Algorithm A.2), *action_id_extraction()* (Algorithm A.3) and the function for the computation of the events that may happen *update_event_may_happen()* (Algorithm A.4).

The functions for the extraction have respectively the objective to extract the entity, the event and the action depending on the previous ids extracted, following how the **SEEA0** matrix is declared for the pattern. They have more or less the same structure as the *sense_id_extraction()* function described in Section 4.1.2 with the Algorithm 4.2. They have similar operations but different data related to the elements for which the function is developed. They start to reset some variables (Phase 1) to proceed to the selection of the possible elements to be extracted (Phase 4) considering the **SEEA0** matrix (Phase 2) and the variables created (Phase 3).

Instead, the *update_event_may_happen()* (Algorithm A.4) is a different function concerning the previous ones, because it is used to define which are the possible event that may happen considering the possible active cases (Phase 2). For example, consider the case in which the human has to perform its decisions perceiving the entities in its neighbour. This function has the objective to compute the distance of the entities in the environment (for the selected one) and compare it with the **distance_threshold** and **distance_min** set and if the entity has a distance greater than the **distance_min** but lower than the **distance_threshold**, it is possible to add it to the neighbour of the pattern and then activate (set to 1) all the events declared in the **SEEA0** matrix that has as

a parent that entity (Phase 3). The vector of the event that may happen is used in the *update_weights()* function to compute the weights of only the sense, entity and action that is related to the events that may possible **event_may_happen** vector.

Algorithm A.1 *entity_id_extraction()*

```

1: Input: weight_sum, sense_id_extracted, weight_vector_entity,
          interval_vector_entity, selected_vector, id_trovato, n_selected,
          id_selected, n_action, n_rows, n_dim
2: //Phase 1 - Reset to zero some variables
3: id_selected, selected_vector, weight_sum = 0
4: //Phase 2 - Select for the extraction only the possible entity that
   are possible to be extracted by the sense extracted
5: for m = 0 to n_rows do
6:   if SEEAO[m][0] == sense_id_extracted then
7:     selected_id = SEEAO[m][1]
8:     for k = 0 to n_entity do
9:       if id_vector_entity[k] == selected_id then
10:        selected_vector[k] = selected_vector[k] + 1
11:      end if
12:    end for
13:  end if
14: end for
15: //Phase 3 - Create variables for extraction
16: for k = 0 to n_entity do
17:   if selected_vector[k] ≥ 0 then
18:     id_selected[j] = k + 1
19:     j = j + 1
20:     n_selected = n_selected + 1
21:   end if
22: end for
23: for q2 = 0 to n_selected do
24:   for q3 = 0 to n_entity do
25:     if id_vector_entity[q3] == id_selected[q2] then
26:       weight_vector_entity[q2] = entity_weight_vector[q3]
27:     end if
28:   end for
29: end for
30: //Phase 4 - Extraction
31: Normalise the weights as done in phase 3 of the Algorithm 4.1
32: Create the variables for the entity as done in phase 2 of the
   Algorithm 4.2
33: Extract id_extracted as done in phase 3 of the Algorithm 4.2 with entity
   variables
34: return id_extracted

```

Algorithm A.2 *event_id_extraction()*

```

1: Input: weight_sum, sense_id_extracted, entity_id_extracted,
           weight_vector_event, interval_vector_event, selected_vector,
           id_trovato, n_selected, id_selected, n_event, n_rows, n_dim
2: //Phase 1 - Reset to zero some variables
3: id_selected, selected_vector, weight_sum = 0
4: //Phase 2 - Select for the extraction only the possible entities that
   are possible to be extracted by the sense extracted
5: for m = 0 to n_rows do
6:   if SEEO[m][0] == sense_id_extracted && SEEO[m][1] ==
     entity_id_extracted then
7:     selected_id = SEEO[m][2]
8:     for k = 0 to n_event do
9:       if id_vector_event[k] == selected_id then
10:        selected_vector[k] = selected_vector[k] + 1
11:       end if
12:     end for
13:   end if
14: end for
15: //Phase 3 - Create variables for extraction
16: It follows the same steps described in phase 3 in the Algorithm A.1
17: //Phase 4 - Extraction
18: //Normalise the weights as done in phase 3 of the Algorithm 4.1
19: //Create the variables for the entity as done in phase 2 of the
   Algorithm 4.2
20: //Extract id_extracted as done in phase 3 of the Algorithm 4.2 with
   entity variables
21: return id_extracted

```

Algorithm A.3 *action_id_extraction()*

```

1: Input: weight_sum, sense_id_extracted, entity_id_extracted,
   event_id_extracted, weight_vector_action, interval_vector_action,
   selected_vector, id_trovato, n_selected, id_selected, n_action, n_rows,
   n_dim = 0
2: //Phase 1 - Reset to zero some variables
3: id_selected, selected_vector
4: //Phase 2 - Select for the extraction only the possible actions to be
   extracted by the extracted sense, entity and event
5: for m = 0 to n_rows do
6:   if SEEAO[m][0] == sense_id_extracted && SEEAO[m][1] ==
   entity_id_extracted && SEEAO[m][2] == event_id_extracted then
7:     selected_id = SEEAO[m][3]
8:     for k = 0 to n_dim do
9:       if id_vector_action[k] == selected_id then
10:        selected_vector[k] = selected_vector[k] + 1
11:      end if
12:    end for
13:   end if
14: end for
15: //Phase 3 - Create variables for extraction
16: //It follows the same steps described in phase 3 in the Algorithm A.1
17: //Phase 4 - Extraction
18: //Normalise the weights as done in phase 3 of the Algorithm 4.1
19: //Create the variables for the entity as done in phase 2 of the
   Algorithm 4.2
20: //Extract id_extracted as done in phase 3 of the Algorithm 4.2 with
   entity variables
21: return id_extracted

```

Algorithm A.4 *update_event_may_happen()*

```

1: Input: case_event, event_may_happen, event_from_entity, neigh,
   case_fixed_obj, case_obj, humanPositionY, humanPositionX,
   distance_min, distance_threshold, distance, nr_object_neigh,
   nr_obj_tmp, obj_xye, distance_obj_rob, id_robot, distance_obj_hum,
   id_person, humanPositionX, humanPositionY, nr_object, SEEO,
   n_human = 0
2: //Phase 1 - Reset to zero some variables
3: event_may_happen, event_from_entity
4: //Phase 2 - Define the cases of the function
5: //All the cases of case_event, case_fixed_obj and case_obj parameters
   that are illustrated in Table A.1
6: if case_event == 1 then
7:   :// Randomly event may happen
8: elseif case_event == 2
9:   if case_obj then //other cases for case_obj and case_fixed_obj
10:  :
11: else case_obj == 3 //pattern perceives human, robot and fixed obj
12:   nr_object = nr_obj_tmp + n_human + n_robot
13:   j = 0 //index for the human vector
14:   for i = nr_obj_tmp to nr_obj_tmp + nr_human do //piece of code for
     the computation of the human distance, that is similar for the robot
     and fixed objects
15:     distance_obj_hum[j] =
       pt_dist(humanPositionX[id-1], humanPositionX[j], humanPositionY[id-
         1], humanPositionY[j])
16:     //verify if the human is in the neighbour of the pattern
17:     if (distance_obj_hum[j] ≤ distance_threshold && distance_obj_hum[j] ≥
       distance_min) then
18:       neigh[i] = id_person
19:     else
20:       neigh[i] = -1
21:     end if
22:     j ++
23:   end for
24:   ://The computation for robots
25: end if
26: end if
27: //Phase 3 - Set to 1 the event that may happen for the entity in the
     neighbour defined in the previous phase, updating the event_may_happen
     vector

```

Acknowledgements

May 4, 2023, my experience as a master's student at the Politecnico di Milano ends here. My journey has been full of events that have made me grow and have a new point of view to see, face and live my future.

This was possible thanks to the trials I had to overcome and the people I had by my side on this journey.

Thanks to all the **Professors** met in the 14 courses addressed, in particular to my Supervisor, Matteo Giovanni Rossi.

Thanks to Livia Lestingi who helped me with the thesis work.

Thanks to my **parents** who allowed me to study, to all **my family, friends** and to those who, with a smile, have stood by me in this endeavour and will be so in my future commitments.

Thank you with all my heart.

Ringraziamenti

4 Maggio 2023, si chiude qui la mia esperienza come studente magistrale del Politecnico di Milano. Il mio percorso è stato ricco di eventi che mi hanno fatto crescere ed avere un nuovo punto di vista di vedere, di affrontare e di vivere il mio futuro.

Questo è stato possibile alla prove che ho dovuto superare e alle persone che ho avuto al mio fianco in questo viaggio.

Grazie a tutti i **Professori** incontrati nei 14 corsi affrontati, in particolare al mio Relatore, Matteo Giovanni Rossi.

Grazie a Livia Lestingi che mi ha aiutato nel lavoro di Tesi.

Grazie ai miei **genitori** che mi hanno permesso di studiare, a tutta la mia **Famiglia**, agli **amici** e a chi, con il sorriso, mi è stato accanto in questa impresa e lo sarà nei prossimi impegni a venire.

Grazie di cuore.

