Multi-Agent Based Military Health System for the Future Battlefield

Pieter Joan van Voorst Vader
Master Thesis Information Studies
Track: Business Information Studies
Student number: 11812559
pieter.joan.vanvoorstvader@student.uva.nl
Thesis Supervisor: dr. Alexander W.F. Boer
a.w.f.boer@uva.nl
Thesis Examiner: prof. dr. Tom M. van Engers
vanEngers@uva.nl
Universiteit van Amsterdam
Date: July 6, 2018

Abstract: Contemporary advancements in non-obtrusive sensor technologies and derivable physiological vital signs create good opportunities for remote sensor based triage in a military setting. Diagnosable symptoms can be modelled with descriptive logic in the declarative programming language AgentSpeak. Recent advancements have been made in dealing with the uncertainty of valid sensor measurements for diagnoses and can be simulated with CArTaGo Artefacts. With the JaCaMo framework that integrates the BDI Agent modelling language AgentSpeak and CArTaGo, a Military Health System is modelled with the Prometheus AEOlus methodology. The modelled and developed software application is presented with explained requirements, design choices, functional specifications. The modelled medical ontology is developed with *high level reasoning concepts* for distributed medical triage with weighted diagnosed conditions and a costs model in a military setting for *rescue initiation* counsel.

Keywords: Military Health System, Multi-Agent Systems, Agent based modelling, Beliefs Desires Intentions (BDI) Agents, JaCaMo, AgentSpeak, Distributed Diagnosable System

Introduction

Military Health Systems are of contemporary interest with recent developments in sensor technology, advancements in derivable physiological measurements and benefits for the military and her personnel. The Military Health System can be modelled in the form of a diagnosable system (de Boer, 2017). Moreover, this research suggested the demonstration of an automated solution that incorporates the possibility for generation of faulty sensor data, failing equipment, including an applicable medical ontology for remote military triage with valuation of diagnoses, a monetary costs model, and applying a set of suitable sensors that meet Dutch Army requirements.

These mentioned suggestions inspired this research with the goal to combine these suggestions into an Agent Based Modelling proposal for distributed diagnose and triage. Furthermore, this modelled proposal resulted in a developed pragmatic and realistic Multi-Agent based Military Health System that is presented in this Master Thesis.

Therefore, several research questions have been developed to support these goals and are stated in the next section. Thereafter, the Methods section also presents supportive research topics that are derived from the research questions. The next presented Literature Review results in the suitable *grammar* for the actual Multi Agent Military Health System implementation.

The following Results sections present the reasoning algorithms handling multiple sensor configurations and handling several uncertainties concerning sensor data. The proposed modelled medical ontology with *high level* Agent reasoning methods is also presented, to enable Agents in diagnosing these modelled medical conditions for distributed triage while applying methods for weighted condition valuation and a *rescue initiation counsel* costs model.

These algorithms, medical ontology and *high level* reasoning methods are presented with Source Code excerpts in the *Belief, Desire, and Intention (BDI) Agent* programming language AgentSpeak (Rao, 1996). This Multi-Agent System (MAS) is modelled with the Prometheus AEOlus (Uez & Hübner, 2014) methodology that provides a graphical overview of the modelled MAS and Agent plan capabilities. The supporting *environmental* Agent *tooling* Artefacts implement the simulation of physical sensors, a statistical regression package for dynamic threshold calculation, and a database for historic evaluation of recorded diagnosed

conditions.

The Evaluation section describes modelling challenges, programmed algorithm evaluation, high volume MAS simulation functionalities, and evaluated research questions.

The Discussion section elaborates about made design choices, and limitations of this research are stated. The concluding section also contains suggestions for future efforts, besides final thoughts on this research project.

The Appendices Section provide additional modelling diagrams, graphical user interfaces of the developed and automated Military Health System, Source Code Excerpts of presented algorithm implementations, and an elaborate discussion about additional medical ontology choices in *Towards a medical ontology*.

Research Questions

The suggestions of de Boer (2017) have inspired the following two main research questions:

- How can a military health system be modelled with *Agent Based Modelling* for distributed diagnose and triage in a military setting for rescue initiation?
- How can this automated military health system include contemporary advancements in diagnosis capabilities, algorithms, a medical ontology, feasible sensor equipment, and creating methods for handling uncertainties in compliance with military requirements?

The resulting derived sub-questions are listed below:

- How could applicable diagnosable conditions be modelled in a declarative programming language?
- How could this modelled system be simulated in an automated demonstrable graphical environment for system evaluation purposes, including multiple sensor configurations and scenarios presenting the weighted diagnosed conditions and a costs model?

Accordingly to these questions, that created the foundations for applicable research methods and researchable topics, are described in the following discussed section.

Methods

The nature of research questions directed a qualitative approach, starting with a literature review guided by all stated research questions. Within the scope of these questions, supportive topics can be distinguished.

The first topic concerns general requirements for medical health care systems with robust diagnostic capabilities. The second topic contains the physiological vital sign sensor measurements in a military compliance setting, with sensors that are commercially available or presented in research. This military compliance requires sensor equipment that is unobtrusive in the physical and mental duty of military personnel (de Boer, 2017).

The third topic of review concerns applicable risk zones for available sensor equipment that could be applied in military symptom diagnoses and triage scenarios. The fourth topic is an applicable reasoning mechanism for distributed diagnosis of sensors and medical data. The fifth topic relates to symptoms that can be diagnosed with only sensory inputs, without further human intervention, and are applicable in a military setting.

The sixth topic is about mechanisms and algorithms that could be applied for sensor data evaluation, including the possibility of *faulty* data generation, failing and redundant sensors. This sixth topic of algorithms also includes multiple data points over time and timeliness related components for algorithmic reasoning about measured data.

Furthermore, the suitable *Agent Based Modelling techniques* for a Military Health System created in a Multi-Agent System environment (MAS) are researched. The last literature review subject includes possible improvements over existing Military Health Systems with contemporary advancements in the field, that could be proposed for a pragmatic *BDI Agent* implementation.

The referenced literature is mainly sourced from the University of Amsterdam Library, and guarantees that referenced articles and books have an approved academic value and quality. However, selected symptoms and values are sourced from `https://medlineplus.gov/` and `https://en.wikipedia.org/wiki/`. This combination of *url referenced* websites symptom descriptions and according necessary values of those symptoms. These mentioned websites have published references to academic literature in the *symptoms* related pages, mostly not available in the UvA Library (e.g. printed books). Furthermore, *medlineplus.gov* is an official affiliated website of the U.S. National Library of Medicine `https://www.nlm.nih.gov/`.

The resulting combination of all researched topics are modelled accordingly to the specified requirements indirectly stated in the research questions. Combined with produced graphical models, algorithmic logic, and data values for physiological vital signs provide means for an *BDI Agent* based MAS implementation of a Military Health System.

The resulting software application will be evaluated, confirming the feasibility of modelling, programming the multi-agent system and verifying the correctness of reasoning about the stated hypotheses. With this evaluation the main and remaining sub-questions are answered accordingly, while providing insights in the applied logic and modelling methodologies.

Literature Review

The literature review provides insights on topics of the research questions, however not represented in that spe-

cific order.

Military Health Systems. In addition to the military requirements of unobtrusiveness, power and communication for sensor based health monitoring, mentioned by de Boer (2017), further explicit requirements are described concerning mobile health systems in a more general sense. These findings are described in the next two paragraphs.

Sensor Based Personal Health Systems. Personal health systems consist of a biomedical sensors and several data processing approaches to reason about conditions, symptoms and diagnoses that can be deducted from sensor based readings without further human interpretation or intervention. Interpreting the measurement data is challenging due to failing of the hardware, unreliable data, limited processing power, and energy restrictions.

Mobile Medical System Requirements. A fault tolerant architecture for vital medical based wearable computing is critical. Current solutions apply smartphone and wearable devices that imply resource constraints and have *failure sensitivity*. However, these appliances are applicable in the military and should consider *fault tolerance*. Within a possible three tier architecture consisting of wearable, mobile and more central *cloud computing* nodes, different limitations apply.

The wearable and mobile nodes have processing and computing power challenges and should be able to sustain adverse conditions. Hardware faults could be mitigated with redundant devices combined with algorithms that supply means for a controlled failover or negative health status in case of a sensor failure. More central oriented mobile devices can implement a 'watchdog protocol' that checks the health of computing devices and communication devices and is able to failover to a still working node, for a fault tolerant structure of appliance (Abdali-Mohammadi, Bajalan, & Fathi, 2015).

Sensors. Koydemir and Ozcan (2018) published a review of commercially available wearable and implantable biomedical sensors, in addition to a review of Qi et al. (2017) on that subject, and provide insight in the contemporary miniaturised technological possibilities. Some of these appliances might be suitable for military use due to non-obtrusive measuring methods that do not pose dangers or have limitations in a military context (e.g. earpieces might not be allowed, fingertip sensors are quite impractical when operating a firearm).

Moreover, innovative deductive measurement methods are developed and sensors are integrated into fabric, combined creating heart rate sensors without the necessity of a wired power source. The demonstrated solution consists of a passive RFID tag with an ECG sensor, that is based on signal interruption (Vora, Dandekar, & Kurzweg, 2015) and then calculates the ElectroCardioGram (ECG) spikes with a 99 percent accuracy with a Machine Learning (ML) Algorithm (Vora & Kurzweg, 2016).

The peripheral capillary oxygen saturation (SpO_2) is a physiological parameter that can be measured optically and is based on the light absorption of two light wavelengths generated by a energy efficient Light Emitting Diode (LED). Besides the typical application areas of the finger tip, earlobe and forehead placement, recent research established other locations on the human body where these sensors can be applied. The upper pectoral and calf provided reliable measurement points, even under intense respiratory motion and measurement deviations are within the FDA approved accuracies of ±2 percent. This solution is therefore competitive compared to commercial available PulseOximeters (Kramer, Lobbestael, Barten, Eian, & Rausch, 2017).

Acceleration or Gyroscopes could be applied for sensing movement of the body and support extra opportunities for hypothesis confirmation or rejection.

Medical deductible physiological values. Contemporary research described relations between measurements of SpO_2 sensors and Heart Rate(HR), Systolic Blood Pressure (SBP) and Pulse Transmit Time (PTT). These deducted values can be made via one optic and a single wire ECG sensor placed on a single arm (Zhang, Zhou, & Zeng, 2017). This ECG solution could logically be replaced by the solution of (Vora et al., 2015). Johnston and Mendelson (2004) created methods for deducing the Respiratory Rate (RR) from a SpO_2 sensor.

Estimations of human core temperature with thermal sensors measuring skin temperature are feasible with at least 85 percent accuracy with a precision of ±0.5 °C (Buller, Tharion, Hoyt, & Jenkins, 2010). These innovations provide a pragmatic set of sensor equipment that complies with military requirements for unobtrusive measurement methods. This results in the following applicable or required sensor types: *temperature, ECG, SpO_2, and Acceleration or Gyroscope* sensors.

Sensor Diagnosable Conditions and Triage methods. With the above described non obtrusive physiological parameter measurement equipment, medical triage is possible for battlefield situations without human intervention. Applicable pure sensor based triage methodologies and diagnosable conditions are described in the following paragraphs.

Sensor Based Triage Methodologies. With this pure sensor based approach the data processing can be data driven and knowledge based combined for an automated diagnosis, where the data driven approach with simple and possibly less accurate algorithms might suffice for local sensor data analysis. This strategy seems suitable due to the power and processing limitations, and form the basis for the knowledge based representation of a diagnosis. This ontology based Description Logic (DL) is based on subjective agreements of the *medical institution* and military guidelines (Qi et al., 2017).

The measurements are classified and the available

symptoms are diagnosed into *diagnosed conditions* in a non exclusive manner. This Descriptive Logic method is used to describe the conditions that are feasible in a military setting. These sensor measurable symptoms are described in the next paragraph.

Triage for military situations. Different methods for medical triage exist and many of those incorporate the Glasgow Coma Scale (GCS). GCS requires human interaction with the subject and is therefore unsuitable for unobtrusive sensor based remote triage. However, the sensor measured Systolic Blood Pressure and Heart Rate provide a pragmatic tool for triage in combat situations by dividing the HR through the SBP that results in the *Shock Index (SI)* (Pasquier, Tourtier, Boutonnet, Falzone, & Mérat, 2012). Furthermore, according to Pasquier et al. (2012) "A calculated SI $\geq$ 0.9 results in the worst outcomes for patients", and therefore this non-human intervention triage method is selected.

Further triage methods can be found in the ACSCOT physiologic criteria. Within ACSCOT, an alarming Respiratory Rate is either a RR of below 10 or above 29 breaths/minute. This ACSCOT criteria set also includes the unsuitable GCS method and the applicable sensor measurable SBP $\leq$ 90 mmHg (millimeters of mercury) is considered critical (Newgard et al., 2010).

However, in military settings a SBP $\leq$ 85 mmHg is more commonly applied for triage, and also with use of GCS (Falzone et al., 2017). These differences show the subjectiveness of the agreements of the context dependent agreements in the *institution*.

A more suitable military triage solution proposed by Wendelken, McGrath, and Blike (2003), they suggest that an $SpO_2 \leq 85$ is enough for further investigation, below 70 percent requires immediate intervention and above 70 percent is adequate for the military. They also defined a HR of below 40 bpm (beats per minute) a reason for immediate attention.

The AMON military health solution of Anliker et al. (2004) provides risk levels for physiological parameters for the SBP, SpO_2 and HR. These levels of High Risk, Risk, Deviant Zone and Normal Zones translate into threshold parameters suitable for automated vital sign classification reasoning, and could be transformed into an analogous colour-coding scheme of red, orange, and green. All implemented sensor types are classified according to this *risk zone* method. Furthermore the symptoms concerning *respiratory issues, temperature related, a likely dead, a possibly shot, having impaired mental capabilities and a being probably unconscious* conditions have been specified. *Sources:* `https://medlineplus.gov/` *and* `https://en.wikipedia.org/wiki/`*).*

The hypotheses set of diagnosable symptoms consists of the combination of these latter mentioned symptoms and the risk zone classification method. The Tables 1 and 2 in the Results section, show the chosen selection of symptoms and risk zone values. These tables represent the hypothesis set, and is in compliance with the *sensor measurable* requirements for diagnosis. A single diagnose of a symptom depends on the availability of valid sensor data, and is coded with the triage flag colour methodology.

With this hypotheses set of diagnosable symptoms, multiple concurrent symptoms can be diagnosed in a non-exclusive manner. However, when sensor data is unavailable where a hypothesis requires this specific data, the affected diagnosable symptoms are unavailable at that moment. This selection of diagnosable symptoms is made considering the military applicable sensors within the scope of this research, and possesses a possible value in a military context.

The next paragraphs present the results on the selected Agent Based Modelling method that are applicable to this research.

Agent Based Modelling. The following Agent Based Modelling related paragraphs will introduce the appropriate terms to describe a *BDI Agent* modelled implementation, combined with recent developments in this field. With *BDI Agent based modelling* the dimensions Agent, Environment, Interaction, Organisation and User of the VOWEL methodology are considered (Bourbia, Dib, & Benrazek, 2016). The next paragraphs will elaborate on each of the five dimensions, starting with the Agent dimension.

Agent Beliefs, Desires and Intentions. *BDI Agents* speak out in AgentSpeak(L) BDI (Beliefs, Desires and Intentions) semantic grammar (Rao, 1996). The Agents' beliefs, are the perceived state of the world that contains information about the environment (Ricci, Piunti, & Viroli, 2011). These beliefs are the basis for the typical *state based reasoning* of a BDI Agent. The *desires* or goals are the objectives to be accomplished. The currently chosen *intentions* are the *course of action* of an Agent. These *intentions* are a representation of a selected list of actions *to be executed*. Where the *plans* represent *the means* to achieve future world states (Padgham & Winikoff, 2004).

Beliefs. When an Agent would like to remember something for later reasoning, a *mental note* could be made in the *Belief Base (BB)* of the Agent and is remembered until explicitly removed. *Percepts* from the environment result in an updated percept including a *perceivable event* that could result in a plan execution of an Agent, a reaction on that perceived event. *Speech Acts* or messages of other Agents remain in the Belief Base until specifically removed. (Bordini, Hübner, & Wooldridge, 2007)

Declarative Goal Programming. In the AgentSpeak implementation Jason *(Source:* `http://jason.sourceforge.net`*)*, a *declarative goal* programming pattern could be implemented. The main described forms are: *declarative goal (DG), backtracking declarative goal (BDG), exclusive backtracking declarative goal (EBDG)*. Moreover, Agent commitment strategy patterns are defined to model

common behaviour of a social Agent, these are: *blind commitment (BC), single minded commitment (SMC), relativised commitment (RC), and open minded commitment (OMC).*

These strategies differ in the possible selection of *applicable plans* of an Agent, and whether the Agent believes it can acquire some *desired world state* or completion of a plan. In addition to the described goal programming patterns, more typical strategies are: *maintenance goals (MG) and sequenced goal adoption (SGA).* These described patterns have Jason templates or *pre-processing directives* defined. Furthermore should be noted, that the AgentSpeak language resembles the ProLog syntax with accompanying rule expansion syntax (Bordini et al., 2007).

Conjunctive Goals. Within the Agent reasoning process conjunctive goals have a ProLog syntax that can be applied for context testing, and is considered a more complex approach that increases debugging complexity (Bordini et al., 2007).

However, this method does have potential for the constructive hypothesis knowledge base or *hypotheses set*, where a compound diagnose can be checked for unification with partial rules of a more complex hypothesis. This conjunctive reasoning then simplifies the construction of the validity of the environmental context for plan selection. Moreover, this reduces potential code repetition in context validation.

Count-As methodology. The Count-As rule construction, a more advanced *institution* mechanism, is a relation description methodology that combines *brute facts* with subjective institutional agreements in combination with a context that results in *count-as* rule. This methodology is based on the *Social Reality Theory* of John Searle. Within this count-as reasoning concept, an event or state can result in some conclusion in a social system (e.g. when no sensor events are perceived should *count-as* all sensors are malfunctioning, or when several valid sensor measurements are perceived *count-as* a confirmed hypothesis based on a medical ontology).

These institutional agreements can be semantically programmed in the AgentSpeak language in this manner, and combine subjective and hard facts into a conclusion based on a social agreement (e.g. a hypothesis condition) (Boissier, Bordini, Hübner, Ricci, & Santi, 2013). This provides a formalised semantic reasoning method applied to the medical hypothesis domain and should be considered an appropriate method to describe relations within the context of this research. Within the domain of BDI Agents and Multi-Agent Systems, multiple programming orientations exist. These orientations are introduced in the following paragraphs.

Agent Oriented Programming. In Agent Oriented Programming (OOP) or Agent Oriented Software Engineering (AOSE), there is an orthogonal relation between Agents and the environment. According to Ricci et al. (2011), in this *separation of concerns*, the Agents are the *first-class abstraction* to design and program autonomous parts of a software system to fulfil an individual or collective goal. These Agents can use a blackboard concept that provides actions and percepts of the environment. The Agents implemented in Jason interact via the FIPA ACL (standardised Agent Communication Language) for their Speech Acts (Ricci et al., 2011).

Plan Failures. When an executing plan fails, a plan failure event is generated. The resulting failure handling can also depend on the perceived state of the world or belief base of the Agent, to select appropriate failure handling plans. It can be possible to reason about what went wrong, a form of BDG or EBDG, and apply different measures (plans), for repairing that condition. When an Agent has reasons to believe that a desired goals is not achievable anymore, the intention could be abandoned, that could be either RC or OMC.

An example would be the failure of transmitting a data frame to another Agent and backtracking where the perceived state of the world was inconsistent with the actual state. Accordingly the Agent could select plans to reset the transmitting device, just retry to transmit or abandoning the intention at all. This abandonment could possibly happen when the reset action did not have the desired outcome (Hübner & Bordini, 2007).

Environment Oriented Programming. With the notion of environment and the concept of Artefacts, the software product of CArTagO *(Source:* `http://cartago.sourceforge.net`*)* was developed based on the Agents & Artefacts (A&A) model that implements passive Artefacts. These reactive entities are in charge of services and functions that make autonomous individual pro-active Agents work together in a Multi-Agent System. Based on Activity Theory, Workspaces create the conceptual containers for Agents and Artefacts. The Activity Theory is a psycho-sociological framework with the notion of Artefact and is implemented in the social representation that Agents simulate (Omicini, Ricci, & Viroli, 2008).

With this Artefact construction *cognitive stigmergy* is realised, that means the Artefacts are tools for populating and structuring the environment, for perceiving agents that share these Artefacts, and rationally use for their goals. These Artefact helping Agents in their activity and knowledge about practices (percepts and functions) (Omicini et al., 2008). Following this concept *stigmergic coordination* could happen, that means coordination without communication in a MAS, by using an Artefact. A digital degrading pheromone could be used for signalling, an analogy of the ants' use of pheromones in nature (Ricci et al., 2011).

These Artefacts building blocks are a *first-class abstraction* for a MAS designer that supports the *distributed cognition* that agents perform with the social system cooperation forms. These Artefacts have observable properties, and changes generate events for *focussed* Agents. Artefact

functions or *actions*, can be triggered by an Agent sending a command to an Artefact (Bordini et al., 2007).

In theory, Artefacts support Manual definitions and these are supported by the *Common Artefact infrastructure for Agent Open environment (CArtAgO)*. However, a universal ontology for manual definition remains an open issue (Ricci, Piunti, Viroli, & Omicini, 2009). The Artefact is *the* Agents' body in addition to the Agents' mind.

An expansion on Environment Oriented Programming is the Organisation Oriented Programming (OOP) and subject of the next paragraph.

Organisation Oriented Programming. The Organisation of a MAS concerns three interlinked dimensions: functioning, structure and norms (Hubner, Sichman, & Boissier, 2007). The high level norms dimension is also called the *deontic dimension* that consists of the norm based reasoning for Agents on the Organisational level.

These three dimensions form the Organisational Specification (OS), and when an Agent adopts a role or joins an Organisation an Organisation Entity (OE) is created dynamically with the Moise framework `http://Moise.sourceforge.net`. Within the Moise Structural Specification (SS) are three levels defined: The responsibilities of an agent are on the *Individual Level*. *Acquaintance, Communication and Authority links* between roles, are considered on the *Social Level*.

Role aggregation in groups is defined on the *Collective Level*. Further possible constraints are compatibility relations between roles. In the Functional Dimension, the Functional Specification (FS) is a set of schemes that represent the MAS *global* Organisational goals, decomposed in *plans* and distributed by *missions*.

Furthermore, every mission can also have *cardinality constraints* and defines the limits of the amount of Agents that should and can commit to a mission. In the Deontic Dimension is defined what *is* permitted and obligated in an Organisation, and results into the Deontic Specification (DS). Within Moise there are soft and hard constraints, and are located in the Deontic Dimension. Hard constraints are: *cardinality relations, role compatibility, commitment of permitted and obligated missions, acquaintance, communications links and creation of groups and schemes*.

The soft constraints are *not* guaranteed by Moise and emphasises the autonomy of an Agent. This can result in the Agents' decision to violate a norm. This could be an authority link, mission commitment or goal achievement.

The Moise Organisational framework follows the declarative programming paradigm of AgentSpeak (Hubner et al., 2007). The Normative Organisation Programming Language (NOPL) translates the Moise XML specification to provide the organisational structure for Organisational Oriented MAS (Boissier et al., 2013).

Interaction Oriented Programming. In the more recent development of Interaction Oriented Programming (IOP), is the conception of interaction is a 'first-class abstraction' with a MAS consisting of Agents, Environment, Organisation and Interaction. Interaction can occur by percepts, actions and speech acts.

However, *stigmergic interaction* is feasible with the implementation of a specified protocol only based on actions and percepts. This means that for example the well known *Contract Net* protocol can be implemented by autonomous Agents, or Agents and Environmental Artefact, or Agents, Artefacts and Organisation, or with the explicit incorporation of Interaction Protocols.

This latter version has the advantage of less spreading the protocol specification throughout the MAS. However MAS performance suffers in the current implementation (Zatelli & Hübner, 2014).

JaCaMo framework. The JaCaMo framework `http://jacamo.sourceforge.net` implements the concepts and programming orientations of Agent-Environment, Agent-Organisation and Agent-Interaction for high level reorganising capabilities. JaCaMo is the combination of Jason, CArtAgO and Moise into one Eclipse IDE *(Source: `https://www.eclipse.org`)* plugin for a *BDI Agent Integrated Development Environment* with all of the above described BDI Agent related concepts (Boissier et al., 2013).

The Organisation is practically implemented on top of the CArtAgO with Organisational Artefacts in the ORA4MAS project and gives Agents the ability to observe the *states* of an Organisation (Hübner, Boissier, Kitio, & Ricci, 2010).

With the Agent concepts and programming orientations explained, a BDI Agent specific modelling method is introduced in the next paragraph.

Agent Based Modelling Methodology. The general applicable iterative Agent Design methodology Prometheus (Padgham & Winikoff, 2004) has been extended with Prometheus AEOlus to fit the JaCaMo concept of Agent, Environment and Organisation structure when designing a MAS, and implement the created methodology models (Uez & Hübner, 2014). This methodology is *not* to be implemented exactly, and aims to help structuring a MAS design. Within Prometheus AEOlus four phases are defined: System Specification, Architectural Design, Detailed Design and Implementation phase.

Prometheus follows the BDI concept with more detailed concepts that describe: Actions, Percepts, Events, Goals, Beliefs, Plans, Messages (or Speech Acts), and Protocols that specify interaction rules. These rules are usually associated with the achievement of goals. Combined, these concepts form an *Intelligent Agent* (Padgham & Winikoff, 2004). The extended methodology also includes the more recent Interaction dimension more explicitly in the *work prod-*

uct diagrams that provide a graphical overview of a complex MAS with all the VOWELS dimensions described in detailed tables (Roloff et al., 2014).

With the use of the graphical Prometheus Design Tool (PDT), skeleton code can be generated for Jason Agents, but this tool is not suitable for JaCaMo environments yet. The Prometheus extension proposal (Uez & Hübner, 2014) created the possibility for generating skeleton code and is proposed by Cunha, Billa, and Adamatti (2017). However, this implementation has not been developed yet.

Simulation. A multi-agent system created with the JaCaMo framework has some support for developing simulations. However, JaCaMo does not have an integration with *off the shelf* Agent Based Simulation (ABM) platforms, especially integration with Artefacts and underlying hardware for simulation purposes is not available. Many ABM platforms incorporate a kind of time-stepped simulation clock to orchestrate the integrated simulation (Singh, Padgham, & Logan, 2016). This form of automated demonstration also depicts the agents capabilities for adjusting their behaviour during the *durative actions* provided by the ABM orchestration.

These capabilities could also be demonstrated by programming a limited set of conditions that simulate sensor values, refreshed periodically, grouped in multiple *staged* scenarios. The graphical interface, implementable with a CArTagO Artefact, could visually present some of the behavioural properties of Agents, the current applied values, an overview of the diagnosed conditions and various other features of the fully implemented Multi-Agent System of this research project.

Results

The following paragraphs describe the resulting functional specifications, modelling, implementation and evaluation phases of the proposed BDI Agent modelled Military Health System. The descriptions are represented with the *grammar* described in the *Literature Review* section. Furthermore, the resulting developed application is available on this Master Thesis' Github Repository (van Voorst Vader, 2018).

Functional Requirements Specification of the BDI Agent Based Military Health System. The designed and demonstrated Multi-Agent Military Health System contains an autonomous sensor network controlled by BDI agents. The main goal of the system is to provide counsel on initiating a rescue mission based on aggregated weighted medical triage data and a costs model in a Military Battlefield Setting, in accordance with the main research questions.

To perform medical triage, decisions have to be made about faulty, unreliable or missing sensor data, combination of the same type of data into a single value, and failure of equipment to produce a valid measurement for symptom diagnosis. To streamline the information distribution between Agents, considerations for the implementation of an Agent Knowledge Management System are presented.

Agent Knowledge Management System. With an integration between Knowledge Management (KM) and Business Process Management (BPM), an ontology driven KM system can be developed with three main components: an Enterprise Knowledge Repository (EKR), Knowledge Intensive Workflows or Tasks (KIW/KIT) and a Knowledge Exchange Infrastructure (Toledo, Bordini, Chiotti, & Galli, 2011). This Agent based Knowledge Management architecture can also be applied on the Military Health System (MHS).

Knowledge Management. Within the domain of Knowledge Management of Medical Condition hypotheses in the terms of Agent Based Modelling, these definitions are subjective and defined by the notion of *institution*, and thus socially agreed upon. This leads to the possible construction of *count-as* rules that combine the *brute facts* of measured valid sensor data and subjective hypotheses, into conclusions in the *count-as* construction when conditions or *context* apply. Because of the medical hypothesis library that should be considered *Knowledge Intensive*, Knowledge Management is a suitable strategy to support the Knowledge Intensive Task of a single diagnose of the created hypotheses set.

These before mentioned knowledge intensive tasks are the result of breaking down the complex problem of military health system requirements into an ontology based medical diagnose, sensor data validity and applying a suitable weighted diagnose value and costs model. This is implemented including adjustable personalised thresholds.

The validity time of a measurement is a configurable parameter and is defined by the institution, or socially agreed upon. The costs thresholds have the capability to be adjusted during a mission, according to changing mission conditions or between several monitored missions. Lastly the thresholds for symptoms and diagnoses can be altered during a mission if some thresholds are deemed too strict or an environmental condition causes sensor measurement interference, and could be mitigated by threshold changes.

Knowledge Intensive Workflows. The sensor values and threshold are part of the Knowledge Intensive Workflows (KIW) necessary for Medical Triage of a single person, a team, and with costs included in the main goal of *Rescue Initiation Counsel*. This KIW incorporates the Knowledge Library defined in Agent plans, and are discussed below with accompanying Source Code excerpts to explain the implementation further.

Sensor Data Flow. The challenging medical triage task can be split into several subtasks that assess those challenges. Firstly Sensors with their data are an issue to reason about, handle missing and invalid data, and decide whether to propagate this measured data for further processing. Several sensors can be available in one package, even of different

types. One sensor package is handled by a single Sensor Agent that reasons about those sensors, before propagating valid and plausible results to a *Body* level Agent.

A body level agent has an algorithm that combines similar typed data into a single value for symptom diagnose reasoning. These diagnoses are then propagated to the first level of displayable aggregated triage information in a *team* agent. The displayed results are also relayed towards the command centre in unmodified form, this has the effect of every agent that processes this information, does so autonomously and independently.

The command centre agents also incorporate the costs model for rescue initiation. That costs model reasons on the unmodified diagnose data of each team members' body agent. The not modifying body-agent data has the advantage that this information can also be stored for long term, reanalysis for newly set parameters or retrospective analysis of received diagnose data over time. This mechanism should be considered part of the Knowledge Exchange Infrastructure.

Interruption of Data Flow. When a mission requires a period of *radio silence*, a parameter can be set via the GUI or direct Agent message. This message is communicated between the Team and Command Centre Agent for synchronisation purposes. This effectually stores al diagnoses received by the Team Agent in an Artefact. After ending the radio silence at either Agent and a new diagnose has been received, all data will be sent to the Command Centre Agent.

Some data will already be obsolete for display, since more recent diagnoses have been received. This data is still recorded in a database for later evaluation. The 'most recent acknowledged diagnose' construction ensures that in either diagnose GUI only the most recent diagnoses can be displayed.

Threshold Adjustment Data Flow. Also the threshold percepts of each Agent have the capability to be updated during a mission. The Team or Command Centre Agent can either initiate these updates, where the Team Agent has this ability due to *radio silence* or *data transmission* issues. Either way these values will propagate through Speech Acts towards all team members, to be applied in the Threshold Artefact. This knowledge exchange creates the possibility to adjust values more appropriate to mission conditions.

Personalisation of Thresholds. Inter-personal differences create the base for different *normal* values of physiological parameters. Therefore, some threshold values can be adjusted for a better match with the human subject. Another reason for personalisation is the compatibility of the sensor with a subject, this is also called *inter-personal sensor variability*.

High-risk thresholds have the tendency to be physically harmful for every person and reach physical human limits, those values should not be allowed to be adjusted. The positive effect is that less semi-alarming conditions, Orange triage flags, could be diagnosed less often, and provide a more realistic and less polluted diagnose result.

The personalised thresholds are configured in the *.jcm MAS configuration file*, and applied after every change of thresholds. These personalised values have the format of a $\pm$*value* to the *per team* provided threshold values.

Sensor Calibration. Each individual sensor produces slightly different values, even when they are from the same manufacturer. To solve this issue, when not performed at the supplier, every Sensor Agent has the possibility to adjust the measured values in either positive or negative direction with decimal precision. Therefore each sensor could be calibrated by hand if deemed necessary. These adjustments can be made in the *.jcm* configuration file or via the Sensor Console at runtime, depicted in Appendix B in Figure B2.

Maximum age of a Sensor Reading. A valid measurement has only a limited time of value for diagnosis, and could be considered a *digital pheromone* (Weyns, Omicini, & Odell, 2007) that degrades over time. Therefore every type of sensor has a Belief Base setting for the maximum age in the Body Agent. The limited time of diagnosis value is exemplified by the following examples; *A Respiratory Rate can change rather much during one minute, and the core temperature of a human body gradually changes.* Therefore, an evaluated valid measurement of core temperature has a longer configurable lifetime than a valid RR measurement.

The Body Agent performs pruning amongst the Belief Base incorporating these Sensor Type maximum age thresholds, before the measurement will be removed from the Belief Base.

This lifetime method expands the possible diagnose value of the system, when sensor equipment failures occur or fail to produce reliable measurements. This has the effect that *valid* historic measurements can still be used for diagnosis, even if the other sensor types produce values that are considerably more recent. However, these threshold values should be agreed upon by the military staff and are configured in the *.jcm MAS configuration file*.

Sensor Data reasoning. The sensor packages are indirectly controlled by the Sensor Agents who control the Artefacts for controlling the actual hardware. In this simulated configuration the Artefact implements functions to produce *observable properties* of the *sensor status*, thus if the hardware is functioning properly and according to normal parameters.

This Artefact observable property also enables the Sensor Agent to disable a specific sensor when this sensor is known to produce faulty or unreliable values. When the Health Status is *Not Ok*, the sensor percept value will not be considered *valid* for propagation. A simulated hardware failure of the sensor could also trigger this status during the lifetime of the running system.

Furthermore, the Sensor Status is provided, this differs from the Health so differentiation can be made about a functioning of the general controlling hardware and the actual sensor. An *Offline* status also disables further consideration of any value produced. This mechanism can also be applied to disable high power consuming sensors for power conservation and redundancy purposes of energy inefficient Sensor Packages.

A package can also be set *Offline* for failover consideration combined with a *sensor swap* algorithm applied for periodic extra measurements, strain conservation and reliable redundancy failover (Abdali-Mohammadi et al., 2015).

Sensor anomaly detection with Majority Voting and Regression Algorithm. Moreover, the *sensor readings* are analysed by a regression algorithm, ideally Sequential Minimal Optimization (SMO) regression, to only predict the next sensible value, based on the last 30 readings. According to Haque, Rahman, and Aziz (2015) this sliding window and majority voting provide a viable dynamic threshold the predict a true reading or a sensor anomaly, with the prediction of the next value a dynamic threshold for anomaly detection and a majority voting algorithm that combines all sensor measurements. This method excludes measured outliers to decide on the actual value that will be used for the medical symptom diagnoses.

This regression algorithm should be implemented in the Artefact because of the already available *legacy code* implementations. The less accurate linear regression (Haque et al., 2015) will be applied due to processing power restrictions, and might suffice for this purpose according to Qi et al. (2017).

The Sensor Artefact can thus *predict* the next non-anomalous value based on the sensor readings, and has a threshold limitation built-in. The dynamic threshold consists of the *standard error* and *linear regressed predicted value*. When the measured value is within the dynamic threshold, the value is propagated towards the body agent. This measured value evaluation is perceivable via the *sensor readings observable property*.

The body agent can perform *outlier detection*, to exclude a reading if at least three valid measurements exist within the allowed lifetime, to ignore a value if it seems an outlier (e.g. one value is more than 2 σ of the mean). The majority, *outlier excluded mean*, then *wins* the election by means of the statistical *mean* calculation and provides their *consented* value for Triage.

Sensor Data Reasoning Code Example. This Agent evaluation of the sensor readings is presented in Source Code 1. The Artefact observable properties in the *context* evaluation are set by functionality simulations. The status of *sensor readings* is set according to the implemented regression algorithm that predicts the range of next possible read values. When a sensor measured value is within the calculated thresholds the *OK* value is set. The *sensor health* property serves the purposes of an internal sensor workings check, that could be implemented in Java, and to set the status of preferably excluded sensors within a sensor package to the status *broken*.

Consistent unreliable sensors in a combined package can be excluded in this manner. Furthermore, the sensor has a more generic status of *sensor status* that can be either *offline* or *online*. This feature enables to automatically disable sensors at runtime, and excluding perceived values from further processing, because there is a fault within the more general sensor controlling hardware (e.g. cpu errors, software faults, internal communication issues). These latter reasons are extending the previously mentioned reasons for *strategically* set statuses.

These perceivable statuses of *sensor health, status, and valid measurements* are verified in the context of an Agents' goal. The *getAlive* Artefact operation updates an observable property for active communication verification in the Sensor Artefact for responsiveness verification purposes. When this action does not respond or fails, the plans for communicating this perceived measured sensor value are interrupted. A *non responsive* situation would cause the plan to be *suspended* until removal of the intention off the intention-stack, the active failure situation would set the *activeSensor* belief to the *MissingResult* status, and the sensor reading would be rejected by the plan for communicating the value to the *Body Agent*.

```
+!evalValidMeasurement(X,Id,Name,Time) :
  ↪   sensorHealth("Ok")[artifact_name(Id,Name)] &
  ↪   sensorReadings("Ok")[artifact_name(Id,Name)] &
  ↪   sensorStatus("Online")[artifact_name(Id,Name)]
<-  getAlive(Ali)[artifact_id(Id)];
    !checkAliveSensor(Id,Ali,Name);
    !updateBeliefBodyAgent(X,Id,Name,Time).
+!evalValidMeasurement(X,Id,Name,Time).
-!evalValidMeasurement(X,Id,Name,Time).

+!checkAliveSensor(Id,A,Name) : A == ("Alive")
<-  -activeSensor(Name,Id,_)[source(_)];
    +activeSensor(Name,Id,A).
+!checkAliveSensor(Id,A,Name) : true
<-  -activeSensor(Name,Id,_)[source(_)];
    +activeSensor(Name,Id,"MissingResult").

+!updateBeliefBodyAgent(X,Id,Name,Time) : myBodyAgent(Ag) &
  ↪   sensorType(SensorType)[artifact_name(Id,Name)] &
  ↪   activeSensor(Name,Id,"Alive")
<-  .send(Ag,tell,vM(X,Name,Time,Id,SensorType)).
-!updateBeliefBodyAgent(X,Id,Name,Time).
```

Source Code 1: Sensor Validity Reasoning

Hypotheses Reasoning Logic Knowledge Library. The knowledge repository consists of Artefacts containing thresholds for diagnosable symptoms, weighted costs calculation for initiating a rescue mission, and also physiological sensor parameter risk zone thresholds for single sensor triage colour classification. Diagnosed conditions combining multiple types of measurements into one condition, have more

than one decision path to a concluded diagnose, see Tables 1 and 2 for an overview of these thresholds, risk zones and diagnosable symptoms.

These mentioned thresholds are perceived by the Agents who reason with those from the plan libraries, the agents' capabilities, for *Sending valid sensor data, Combining received data and maintaining only measurements within the maximum age of a sensor, Reasoning about Symptoms, Diagnoses and Triage flags, and Applying a costs based reasoning model for rescue initiation*. These knowledge based plan libraries, that are also called *Agent Capabilities*, are a form of *description logic* based reasoning and are implemented in the form of declarative goal programming. Examples of the capability models are depicted in Figures 4, 5, and Appendix A. Additional AgentSpeak source code excerpts of the implementation is also provided in Appendix C. The reported thresholds are implemented in the *Threshold Artefact*, that has *observable properties* for all displayed values, and has *Artefact Actions* to apply adjusted received thresholds.

The GUI for this Artefact is depicted in Appendix A in Figure A3. These values are perceived in the BB of the Agent and are used in plans and conjunctive *count-as* rule validation for symptom diagnosis. An example of a hypotheses set shown in Source Code 2, where the ProLog rules provide a *True Unification* when the stated rule is verified and used in the selection of the *applicable* medical diagnosis plans of each *Body Agent*.

```
resp_apnea(SV) :- apnea(X) & SV <= X.
resp_bradypnea(SV):-apnea(X) & bradypnea(Y) & SV > X & SV < Y.
resp_tachypnea_hypervent(SV):-tachypnea_hypervent(X) & SV>=X.
capabilities_unconsciousness(SV):-unconsciousness(X) & SV<=X.
capabilities_impaired_mental(SV):-impaired_mental(X) & SV<=X.
heat_hyperthermia(ST, SV) :- hyperthermia(X) & SV >= X.
heat_hyperpyrexia(ST, SV) :- hyperpyrexia(X) & SV >= X.
inShock(HR,SBP) :- shockSi(SiX) &  HR / SBP >= SiX.
isShot(ST, SBP) :- shotSbp(SX) & SBP < SX & ST == ("sbp").
isShot(ST, SBPold, SBPnew) :- ( SBPold - SBPnew )  > 0 & ST ==
 ↪   ("sbp").
isShot(ST, SBP, SD, SBPold, SBPnew) :- SD \== 0 & shotSd(SdX)
 ↪   & jia.my_multiply(SD,SdX,Result) & Result <= ( SBPold -
 ↪   SBPnew ) & isShot(ST, SBPold,SBPnew).
```

Source Code 2: Symptom Hypotheses Set Rules

Body Agent Medical Condition Diagnosis. After the sensor data reducing majority voting process, a statistical mean value calculation and available outlier detection, this *valid single measurement* is a *mental note* in the belief base of the Agent with a *timestamp* for maximum lifetime and BB pruning purposes.

The single sensor type diagnoses classify a value based on threshold zone specifications, via Threshold Artefact percepts, and calculate the triage flag colour, observed symptom combined with appropriate naming conventions and risk-zone classification. The more complex conditions, for instance *Likely dead* and *Possibly Shot*, combine several *count-as* ProLog style rules for evaluating the current perceived state of the world, or *context*.

The plans evaluating these conditions combine several rules that create a better overview when reasoning about one diagnosed condition. This stacking of rules creates a reasoning hierarchy, based on deductive logic. When only two sensors produce valid values, like HR and RR, this could lead already to accepting the hypothesis for the *Likely Dead* symptom, when these values are zero. When more different types of measurements are available that support the same diagnosed condition, these values can be added to the existing *minimal rule set* with clear overview of all possible conditions. This ProLog logic reasoning is also understandable for medical personnel with little programming experience (Singh et al., 2016), an advantage of the AgentSpeak language, for hypothesis algorithm validation. These multiple rules for one symptom are shown in Table 2.

```
+!checksbp : avSen("sbp",SBP,SBPdev) &
 ↪   sbp_lower_th_highrisk(X) & SBP < X
<-  +diagnosed("sbp","hypotension_highrisk",SBP,3).
+!checksbp : avSen("sbp",SBP,SBPdev) &
 ↪   sbp_lower_th_highrisk(Y) & SBP >= Y &
 ↪   sbp_lower_th_risk(X) & SBP < X
<-  +diagnosed("sbp","hypotension_risk",SBP,2).
+!checksbp : avSen("sbp",SBP,SBPdev) & sbp_lower_th_risk(X) &
 ↪   SBP >= X & sbp_high_th_risk(Y) & SBP < Y
<-  +diagnosed("sbp","deviant_normal_zone",SBP,1).
+!checksbp : avSen("sbp",SBP,SBPdev) & sbp_high_th_risk(X) &
 ↪   SBP >= X & sbp_high_th_highrisk(Y) & SBP < Y
<-  +diagnosed("sbp","hypertension_risk",SBP,2).
+!checksbp : avSen("sbp",SBP,SBPdev) & sbp_high_th_highrisk(X)
 ↪   & SBP > X
<-  +diagnosed("sbp","hypertension_highrisk",SBP,3).
+!checksbp.

+!diagShot : avSen(ST,SV,SD,SBPold,SBPnew) & ST == ("sbp") &
 ↪   isShot(ST, SV, SD, SBPold, SBPnew)
<-  SBPold - SBPnew = Drop;
    .concat(" sbp: ", SV," drop_sbp(old-new): ",Drop,"
     ↪   std_dev:  ",SD," ",Values);
    +diagnosed("shot","likely_shot",Values,3).
+!diagShot : avSen(ST,SV,SD,SBPold,SBPnew) & ST == ("sbp") &
 ↪   isShot(ST, SV)
<-  //multiple values found thus an SBPold/new is calculated,
 ↪   diff with single sensor
    .concat(" sbp: ", SV," std_dev: ",SD," ",Values);
    +diagnosed("shot","likely_shot",Values,3).
+!diagShot : avSen(ST,SV,SD) & ST == ("sbp") & isShot(ST, SV)
<-  .concat(" only_singlevalue_sbp: ", SV," ",Values); //No
 ↪   StandardDeviation because of only one value
    +diagnosed("shot","likely_shot",Values,3).
+!diagShot : avSen(ST,SV,SD) & ST == ("sbp")
<-  .concat(" sbp: ", SV," ",Values); //No StandardDeviation
 ↪   because of only one value
    +diagnosed("shot"," -=No=- ",Values,1).
+!diagShot.
```

Source Code 3: Observed value classification and hypothesis diagnose

These diagnosed conditions are individually stored in the Belief Base with *kind of condition, status, values, triage colour code*. When a triage maintenance plan is completed, with the modelled plan depicted in Figure 3, a *timestamp* is generated and added in combination with the Body Agent' name and team name. This combination of values is to be

	Risk Zones:				
Sensor Types:	*L HighRisk {3}*	*L Risk {2}*	*Normal-Deviant {1}*	*H Risk {2}*	*H HighRisk {3}*
Systolic Blood Pressure (SBP) in mmHg	≤60	61 - 80	81 - 160	161 - 180	≥181
Pulse Oxygenation (SpO_2) in %	≤80	81 - 92	93 - 100		
Heart Rate (HR) in beats/minute	≤45	46 - 50	51 - 120	121 - 180	≥181
Respiratory Rate (RR) in breaths/minute	≤10		11 - 28		≥29
Temperature in °C	≤35		35.1 - 38.2	38.3 - 39.9	≥40.0

Triage Colour Flags: Red {3}, Orange {2}, Green {1}

Table 1 *Risk Zones and Triage Flags*

	Triage Colour Levels	
Diagnosable Symptoms	***Red {3}***	***Orange {2}***
Likely Dead:	*Likely Dead: (and Acceleration ≤ 0.4m/s, if available)* HR ≤ 0 & RR ≤ 1 HR ≤ 0 & RR ≤ 1 & SBP ≤ 1 HR ≤ 0 & RR ≤ 1 & Temp ≤ 28 HR ≤ 0 & RR ≤ 1 & SBP ≤ 1 & Temp ≤ 28	*Still Moving - Quite Abnormal:* Acc: > 0.4 m/s *(if available,* *and confirmed hypothesis* *of Likely Dead)*
Shot	*Likely Shot:* SBP ≤ 50 SBP Drop≥ 0 & SBP Drop ≥ (2.0*standard deviation) SBP ≤ 50 & SBP drop≥ 0 & SBP Drop ≥ (2.0*std.dev.) *(SBP Drop = oldest measured value - most recent value)*	
Shock	*In Shock:* HR / SBP ≥ 0.9	
Undercooled	*Hypothermia:* Temp ≤ 35	
Heat Illness	*Hyperpyrexia:* Temp ≥ 40.0	*Hyperthermia:* Temp 38.8 - 40.0
Respiratory Issues	*Apnea:* RR ≤ 0 *Bradypnea:* RR ≤ 12 *Tachypnea or Hyperventilation:* RR ≥ 29	
Oxygenation related	*Impaired Mental Functions:* SpO ≤ 65 *Unconsciousness: SpO ≤ 55*	

Only available sensor measurements are used in hypothesis evaluation,
one line per hypothesis sensor combination, reported threshold values are adjustable

Table 2 *Diagnosable Symptoms and Triage Flags*

sent in a batch of Speech Acts to the Team Agent. After this batch has been sent, another *most recent diagnose* message is composed to notify the Team Agent that all values have been transmitted and could start processing those received values. The stored *symptom values* in the belief differ in amount, according to the type of diagnose and availability of multiple sensor measurements in a diagnosed symptom (e.g. the 'Likely Death' condition can consist of only HR and RR values, and optional are: SBP, Temp and Acceleration, when available).

Diagnosis on pure sensor data has many uncertainties, and in many cases human participation for further triage is needed to determine exact causes or apply GCS. Therefore, every symptom and ailment diagnose is evaluated individually, to have the potential supplying many possible diagnoses that are feasible and can be deducted from only sensor measurements. Possible causes for symptoms are not reported because of the great uncertainty and amount of causes to report. Furthermore, most causes can only be excluded with human intervention.

Medical Diagnosis reasoning example. The code excerpt Source Code 3 shows the single sensor classification with the before mentioned use of observed current thresholds. The depicted *!checksbp* goal compares the calculated mean value of all sensors of a Body Agent with the *risk zone* thresholds, and creates a new belief containing all necessary values. The last *!checksbp* plan option is the *applicable plan* in case of a non-existent sensor value. The second part of Source Code 3 shows the plans for the *Likely Shot* diagnosed condition, where the *isShot(....)* variants are implemented from the hypotheses rules, shown in Source Code 2.

Command Centre Weighted Triage. The unaltered received Body Agent data is reprocessed by the Command Centre Agent for evaluating every diagnosed condition with conditional weights and a costs model, based on the Triage Colour-Coding scheme. The medical conditions are partitioned in two categories, *weighted sensor risk zone and*

weighted medical diagnose classifications. Both categories have an adjustable threshold value. With every individual reading having an appointable weighted value and the category thresholds are adjustable, the potential exists to fully neglect a category. Moreover, some conditions can be made more important for final rescue initiation counsel.

```
rescueThresholdReached(TotalRescueCostAmount, TeamMemberValue,
  ↪   FinalCount) :-  totalRescueCosts(Amount) &Amount >
  ↪   TeamMemberValue.
@planTeamReport[atomic]
+!getUpdatedTeamReport :
  ↪   totalRescueCosts(TotalRescueCostAmount) &
  ↪   teamMemberHumanLifeValue(TeamMemberValue) &
  ↪   doCountLikelyDeathStatus(DoCountDeath)
<-  .findall(mostRecentAck(AgentString, TimeStamp,
  ↪   TeamName),mostRecentAck(AgentString, TimeStamp,
  ↪   TeamName), MRL);
    .abolish(viableRescueCount(_));
    .abolish(deathCounted(_));
    .abolish(deathCountedThresholdsNotMet(_));
    +viableRescueCount(0);
    +deathCounted(0);
    +deathCountedThresholdsNotMet(0);
    !processDiagnosedPerAgent(MRL);
    ?deathCounted(DeathCount);
    ?viableRescueCount(RescueCount);
    !calculateRescueCounsel(TotalRescueCostAmount,
      ↪   TeamMemberValue, DoCountDeath, DeathCount
      ↪   ,RescueCount).

+!calculateRescueCounsel(TotalRescueCostAmount,
  ↪   TeamMemberValue, DoCountDeath, DeathCount, RescueCount)
  ↪   : DoCountDeath == false
<-  TeamMemberValue * RescueCount  = TeamMemberRescueCosts;
    !rescueCostsEvaluation(TeamMemberRescueCosts,
      ↪   TotalRescueCostAmount).
+!calculateRescueCounsel(TotalRescueCostAmount,
  ↪   TeamMemberValue, DoCountDeath, DeathCount, RescueCount)
  ↪   : DoCountDeath == true
<-  RescueCount - DeathCount = FinalCount ;
    TeamMemberValue * FinalCount  = TeamMemberRescueCosts;
    !rescueCostsEvaluation(TeamMemberRescueCosts,
      ↪   TotalRescueCostAmount).

+!rescueCostsEvaluation(TeamMemberRescueCosts,
  ↪   TotalRescueCostAmount) : TeamMemberRescueCosts >=
  ↪   TotalRescueCostAmount
& doCountLikelyDeathStatus(DoCountDeath) & deathCounted(X) &
  ↪   viableRescueCount(Y) & deathCountedThresholdsNotMet(Z)
<-    printCounsel(true,DoCountDeath,X,Y,Z).
+!rescueCostsEvaluation(TeamMemberRescueCosts,
  ↪   TotalRescueCostAmount) : TeamMemberRescueCosts <
  ↪   TotalRescueCostAmount
& doCountLikelyDeathStatus(DoCountDeath) & deathCounted(X) &
  ↪   viableRescueCount(Y) & deathCountedThresholdsNotMet(Z)
<-    printCounsel(false,DoCountDeath,X,Y,Z).
```

Source Code 4: Excerpt of *Final Rescue Counsel* plans

Lastly the summing of category threshold scores, both reduced to one when the thresholds are reached, have an accumulative threshold that varies from one to two. This creates the differentiation for having the requirement to have positive evaluations on both categories, before a complete individual weighted diagnose counts in the *total rescue count*. This counting system also has one subtraction capability, that can be set either *On* or *Off*, and subtracts the *Likely Death* count from the category evaluation. With this construction, a person can score positively on the count to be rescued, but because the person is *likely not alive*, then this *rescue count* is visibly subtracted from the *total team rescue count*.

Command Centre Costs Evaluation. After the cumulative rescue count has been calculated, in either form, this value is multiplied by a customisable amount for the *team member human life value*. This produces a *total amount of possible loss* expressed in a numerical value. A rescue mission also has a cost value, the *total rescue costs* consisting of a *rescue equipment or operational cost*, and a *human life value for rescue personnel* multiplied by the amount of personnel required for a rescue mission. When the possible loss of human life is greater than the rescue costs, a positive counsel is provided to initiate a rescue mission. To provide insight in this calculation, the actual *total rescue count* is provided, and the *death count* is visible. Examples of this counsel are depicted in Appendix B, Figures B9 and B10.

Final Counsel reasoning example. The plans shown in Source Code 4 show the an *atomic* plan label, this method guarantees that after this plan has been initiated, no new percepts or messages are processed until the plan execution has finished. This method was chosen because of the many needed reasoning cycles for looping through the several *.findall(....)* list structures needed for calculation of reached thresholds of every *team member* and all individual triaged conditions. These several loops are in Object Oriented programming terms, multi dimensional arrays. The Diagnose Console initiated *+!getUpdatedTeamReport* event perception explicitly resets all counters and applies *test goals* for retrieving the beliefs for the final Artefact Action *printCounsel()* is executed by the Agent, that updates the GUI of the Command Centre Agent.

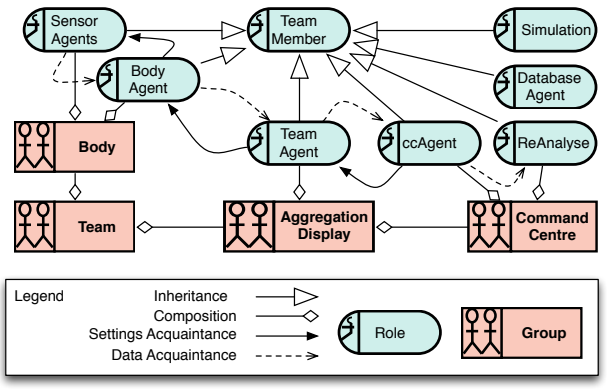

Figure 1. AEOlus Structural Overview with Acquaintance Relations

Multi-Agent Structural Overview. Figure 1 depicts the Structural Diagram modelled with the extended Prometheus AEOlus methodology (abbreviated to AEOlus). This figure shows that each Agent inherits from the role *Team Member*. The Agents are also hierarchically grouped for organisational purposes. The acquaintance relations are also

depicted, and visually show the restrictions of communication between the several Agent Roles.

The Sensor and Body Agents form the Body Group and share a workspace and Artefacts. A more complex relationship exists between the Agents in the Command Centre Group, where the Aggregation Group functionalities and capabilities are expanded with the Rescue Counsel displayed in a Team Agent *Diagnose Console*, depicted in Appendix B, Figure B8 and methodology Legend in Figure A1. The Re-Analyse Role has only a one-way acquaintance with the Command Centre Agent in the receiving direction, an Authoritative relation. The Team and Body Agents also have two Authoritative relations, from the Body towards the Team Agent for Diagnosed Condition purposes. Vice versa this relation exists for the adjustment of diagnose costs thresholds.

The Simulation Role is an artificial construction that has permission to change and set all available thresholds and all individual measured sensor values for demonstration purposes. This communication is performed with *Speech Acts*.

The Database Agent role only inherits from the Team Member role and does not have acquaintance relations since all perceived data is retrieved from a database Artefact.

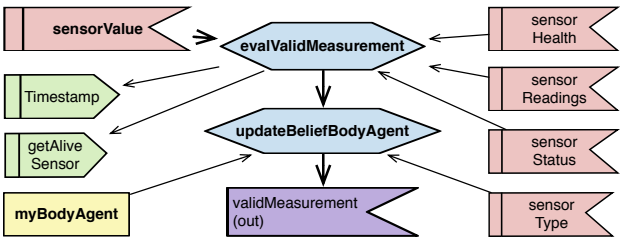

Figure 2. Artefact triggered Sensor Agent evaluation capability

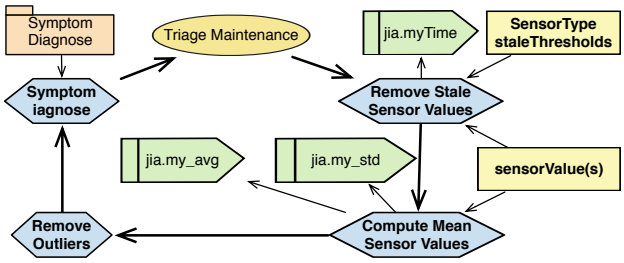

Figure 3. Body Agent Triage Maintenance plan.

Agent Roles and Capabilities. Indirectly explained above, the six types of Agent have been described. These Agents have also been modelled in AEOlus. The Sensor Agents' Capability diagram is depicted in Figure 2 showing the flow of a received measurement event by the Sensor Artefact ending in a Speech Act towards the Body Agent.

The Body Agent has a Triage *maintenance goal* depicted in Figure 3 with the actual schematic overview of the

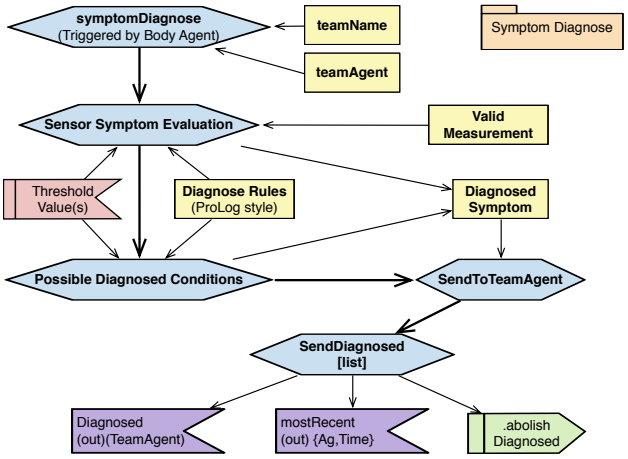

Figure 4. Body Agent Symptom Diagnose Capability.

implementation of the Medical knowledge library into the Agent plans, depicted in Figure 4, where this Knowledge Intensive Workflow actively uses the information from two parts of the Knowledge Library to produce the symptom diagnoses.

The Team Agents' *Flag and Diagnose* capability for GUI display has been fully implemented in AgentSpeak and this depicted from receiving the *Most Recent Diagnose* of any Body Agent towards the *Update GUI* action in Figure 5. The Costs Diagnose capability of the Command Centre Agent is displayed in Figure A2, and initiated by clicking the *Update TeamReport* button. Additional modelled capabilities are provided in Appendix A.

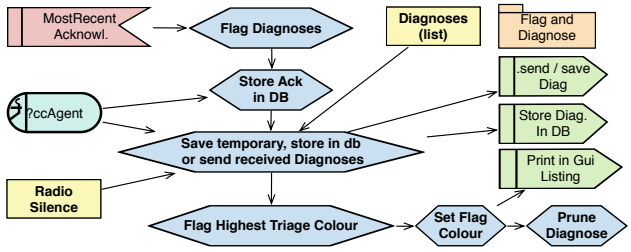

Figure 5. Team Agent – Flag and Diagnose Capability

Weighted Costs Simulation. An Agent in the Re-Analyse or Database role are the only types in the MAS that have finite goals and do not handle *live* data. The Re-Analyse Agent can be initiated from a Command Centre *Diagnose Console* Artefact and will receive the current thresholds and all currently available relevant beliefs of the Command Centre Agent. This process could be modelled with a Moise Organisation, because the task is finite.

However, no further cooperation with other agents is necessary. The main function of this simulation is to adjust weighted diagnosed symptom and costs thresholds on the *static* dataset, to acquire insight into optimal parameters for

a mission counsel, before actually apply those tested settings into the *live* environment.

Evaluating Mission Data. Furthermore, all data received by the Command Centre Agent is stored in a Java HyperSql database, like mentioned before. This data is accessible for a *per team, mission, agent, timestamped* base for display via the Database Agents' *Database Console and Database Diagnose Console*, depicted in Appendix B in Figures B1 and B7. This data is displayed by a newly started *helper* Database Agent that processes the data retrieved for inspection. Every recorded session of the MAS and team causes the mission identifier to increment for separation purposes. The fine grained per timestamped result set display method enables *in detail* review of a missions' sensor data analysis.

Command Centre Agents. The above-described Multi-Agent System describes the functionality for only one team. However, multiple teams can be monitored concurrently by adding a new set of all Agents with another unique Team name and Command Centre Agent name. This initialises another GUI Artefact for Rescue Initiation Counsel, together with independent settings from every other concurrently running team configuration.

Evaluation

With the modelled and developed Multi-Agent based Military Health System described in the *Results* section above (van Voorst Vader, 2018), encountered software implementation challenges are described in the following paragraphs. Thereafter, the research questions are evaluated that also validate the functioning of the modelled medical ontology, algorithms, weighted diagnosable triage conditions, the costs model and general MAS functionalities.

Graphical User Interface Implementations. Multiple GUI Artefacts have been created for Users to set values of the various components in the MAS. The Figures B9 and B10 depict examples of the costs evaluation calculation in the combined GUI for individual agent diagnose viewing and resulting team rescue counsel. In this GUI Artefact the Body Agent list in the left column is updated when a new diagnose is processed by the Command Centre Agent. The displayed age of the last diagnose is also recalculated for every agent in the list, and in a glance the importance of a diagnose can assessed due to the applied Triage Flag Colour Scheme. This scheme ranges from Red, Orange, Green, and Black. The Black colour is appointed when the Body Agent does function nominally and perceived no valid measurements for diagnosis.

Within the also alphabetically sorted drop-down menu, all agents displayed can be selected for further inspection of the full diagnoses report. When there is an absence of valid sensor measurements, a 'NaN' (Not Available Now) value is displayed. This creates the opportunity for Command Centre Agent Users to possibly inference more conclusions than are displayed. Furthermore, since all available sensor types are shown, it is quickly derivable what sensors where unavailable for the more complex diagnoses.

AgentSpeak Internal Functions. During the implementation of all functionality of the MAS in JaCaMo, challenges were encountered that are described in the following paragraphs.

The calculations for the average, standard deviation, multiply and divide of sensor data sets have been implemented in *Jason Internal Actions*. This functionality consists of four separate functions that have a precision of one tenth of the measured unit. These functions rely heavily on the Unification, Transition System and other principles of unique to Jason. To reliably implement the conversion of negative String formatted values into Double formatted values, the functionality to directly call a *Java class from within AgentSpeak* code has been applied to parse these values into an AgentSpeak calculable format. This method mitigated issues with Artefacts that did not respond in the high volume Agent simulations. Furthermore, these conversions were not possible with custom written Internal Functions.

Agent listing functions. The listing of diagnosed Agents were in need of a few other peculiarities. Firstly an AgentSpeak given number of '1' or '0' would be passed to the Java interpreter in 'Byte' format, while a '2' would be an Integer. To solve these inter-interpreter issues, the Java function constructors received those values in 'Object' format, use the toString() function and then parse that value to an Integer or Double. This implementation was necessary for the GUI Colour-coding of the Agent Diagnose listings and returning input values to Agents. To sort the *Agent Diagnose list* in alphabetic order a custom *AgentDiag* model has been developed, with an addition to create a simplified compare on only the *name* part of the Object model for a match. This enabled a search and replace in the ArrayList implementation that produces the actual displayed *Agent Diagnose list*. The called for-loop called inexplicitly the function *toString()* of the model to actually display html-formatted text, this construction is initiated by the Agent after processing a new perceived result.

The deferred sending Actions utilised a similar custom model construction, and the actual communication is left to the autonomous Agent and is instructed via the *signal* Artefact construction, filled with the command and actual values. For possible *ConcurrentModificationExceptions* a temporary list is created with each actually sent item, to ensure that every message is only sent once, is removed from the list, and the list is still accessible by the agent for concurrently adding more items to that list. The Artefacts have besides GUI buttons what link to *@INTERNAL_OPERATION* also Actions *@OPERATION* that perform that function, and are also used for value propagation by the Agents.

Testing the MAS. The per *Agent Symptom Simulator GUI* incorporates individual sensor value settings for *risk zones*, and the more complex symptoms have scenario buttons. This was developed for verifying the correct functioning of the implemented medical hypothesis rules and plans. A button press signals the *Symptom Agent* to retrieve all known sensors of the Agent Body Group from the belief base and set the desired sensor value via the Artefact *action*. Since the Agents are reactive and pro-active autonomous entities, the *signalling* triggered events in the Sensor Agents of a Agent Body Group, and the Knowledge Intensive Workflow was initiated towards the end result of a Rescue Initiation Counsel.

High Volume Simulation and Testing. This design was expanded for the high volume MAS simulations that created the desired functionality of a Agent Based Modelling Simulation platform implemented in JaCaMo. This simulation Agent Console has five stages that form the *stages* in the AgentSpeak developed scenario profiles. These scenario profiles can be configured on a per agent basis. A scenario can therefore show several states of a single agent confirming the valid algorithm implementation according to specifications. This simulation Agent is contacted by each Body Agent groups' Symptom Agent, and communicates the current *active stage* of the simulation to each of the Symptom Agents. Each Symptom Agent then applies the scenario profiled plans and set each available sensor of the Body Agent group accordingly. These values are refreshed automatically for demonstration purposes by the Symptom Agents. These *set* sensor values initiate the reactive behaviour of all the Agents in the MAS hierarchy, to ultimately display the *Final Rescue Counsel*.

High Volume Simulation Challenges. The final version of the implemented product initialised 243 Agents in a single MAS for simulation of 40 Body Agents, one Team Agent, and one Command Centre Agent. The autonomic nature of Agent plan execution and extremely rapid agent initialisation showed a few flaws in the previous implementation iterations of this modelled MAS solution. The artefacts became sometimes unresponsive, this issue was resolved by introducing internal action *wait* function with the use of *math.random* functions at various Artefact related plans in all Agents. Furthermore, the Agent initialisation order is also random and due to the randomly ordered plan execution, Agent announcements towards other Agents of their existence was redesigned into a *maintenance plan* programming pattern executing until an acknowledged belief of the targeted Agent was in the Belief Base of the initiating Agent. Furthermore, the *percept, action, reasoning* cycles have been increased and *pooled* in eight threads, since a four core, 8 threaded cpu was used for the simulation, and provided a more predictable and stable execution of the Agents plans and GUI updates.

The following paragraph will describe the answers to the research questions, building on the sub-questions.

Research Question Evaluation. The main research question of *"How can a military health system be modelled with Agent Based Modelling for distributed diagnose and triage in a military setting for rescue initiation?"* is answered by the MAS descriptions and models presented in the *Results* section that describe this modelled military health system, distributed diagnosis capabilities and selected triage methods.

Furthermore, the presented implementation description, that incorporates topics of the literature review, answers the second research question *"How can this automated system include contemporary advancements in diagnosis capabilities, algorithms, a medical ontology, feasible sensor equipment, and creating methods for handling uncertainties in compliance with military requirements?"*.

The first sub-question *"How can the applicable diagnosable conditions be modelled in a declarative programming language with descriptive logic?"*, is answered by the description of the applied modelling methods, algorithms with accompanying explanations and Source Code excerpts.

The second sub-question *"How could the resulting methods be simulated in an automated demonstrable graphical environment for system evaluation purposes, including multiple sensor configurations and scenarios presenting the weighted diagnosed conditions and a costs model?"*, is answered by applying the researched simulation methods for a Agent Based Modelling simulation in the presented implementation. This simulation demonstrates the functioning of multiple sensor configurations within the staged symptom simulation profile environment.

These *staged symptom simulation profiles* that are per agent configurable, represent the validation of all implemented algorithms. These results can be visually verified by evaluating the configured threshold settings and displayed sensor values. Combined with the capabilities to alter thresholds during the simulation, the effects of these changes can be evaluated. This includes the thresholds for the diagnosed conditions, the individual weighted value for each diagnosable condition, and the costs model implementation.

At any moment in the main simulation interface, a static dataset can be generated in a separate interface for evaluation of desired weighted triage and costs model results. This separate simulation does not interfere with the main simulation. The database recording of each Body Agent timestamped result set enables further evaluation of an individuals' condition over time, in a separate interface, including historic missions and all current mission data.

Further modelling and design considerations are presented in the following Discussion section.

Discussion

Considerations that are made during the modelling and implementation of the Military Health System are discussed below.

Design considerations. The possibility existed to implement Moise in the whole system. Due to the many maintenance tasks, large amount of communication that the Moise implementation generates, and the continuous character of sensor analysis and reporting, the choice was made to implement the *live* agents with *normal* maintenance goals and an Agent-Environment implementation. Furthermore, little coordination is necessary between agents due to the fault tolerant design of the MAS, while coordination is one of the main purposes of Organisation with Moise.

Protocols in the design have been implemented pragmatically by sending a batch of messages, with confirmation or acknowledged end of batch messages. This implementation has been done for synchronizing reasons, manageable data structure within the belief base, and handling the unsure course of action of any Agent in the MAS. The JaCaMo Interaction implementation has not been implemented for performance reasons.

Only the final Rescue Initiation Counsel has been made an *atomic* function. This creates the obligation for the Agent to execute that plan and subsequent plans, before doing anything else. This created theoretic certainty then after the User initiated *Update TeamReport* request, this was handled with that *state of the world* and not contaminated by a Belief Base update due to an incoming Speech Act. Any perceived event has the potential to disrupt the full team report calculations that involve factually the use of looping through multidimensional arrays implemented in AgentSpeak. This looping consumes many reasoning cycles and is quite fast. However, when 40 Body Agents concurrently propagating values in the Command Centre direction, the amount of disturbance is quite considerate.

Medical Hypothesis and Triage Considerations. In line with qualitative research, a more humble position was regarded towards the implementation of medical hypotheses. Only values found in published literature were included. Furthermore, a conservative stand towards possible diagnoses was implemented. This can be experienced in the following reasoning example: “When the required sensors for some complex diagnose did not provide valid measurements within the maximum evaluation time span, then such diagnose should be excluded from reasoning”. This means that if unsure, do not suggest it. Many more elaborate diagnoses require human intervention and interaction with the subject of measurement.

The requirement of those interactions exclude any possible diagnose on those hypotheses or underlying causes. Therefore these interaction requiring typed symptoms and all causes are excluded from consideration and reporting. This includes the, also in the military field widely used, Glasgow Coma Scale for military triage that is substituted by the *Shock Index* method. The several in the military Triage Methods include the use GCS scale for reliability reasons, and has human interaction requirements (e.g. pinching someone, and watch for some physical reaction). Therefore educated guesses, based on published material have been made and are referenced to, in the literature review above. Appendix D presents a more detailed discussion on the creation of the medical ontology depicted in Tables 1 and 2

This resulted in five displayed sensors, possibly with only three or four physical sensors required, while six are implemented in the Artefacts. These individual readings are *Triage Risk Zone Coloured* classified, Furthermore, eight more complex conditions that are not directly derivable from risk zone analysis, are included in the hypotheses set evaluation algorithms implemented in AgentSpeak. Another issue with medical conditions is the disagreement in published research on quite common symptoms (e.g. hyperpyrexia lower threshold differs from 40.0 °C to 41.5 °C). This medical disagreement and reliance on institutional related factors, also motivated the design choice to have *in mission* adjustable thresholds that are propagated towards all Agents.

Further considerations have been made upon the flagging method for risk zone and diagnosed symptom scores. Multiple existing triage methods use a colouring scheme based on four colours (Black, Red, Yellow, Green) (Wendelken et al., 2003; Jentsch, Ramirez, Wood, & Elmasllari, 2013; Falzone et al., 2017). However, Yellow has been substituted for Orange for visibility reasons in bright sunlight conditions and viewing a diagnosis on a screen in the field. Yellow is a hard to read colour in very bright conditions, therefore the more distinguishable colour Orange has been selected. Orange is a logical substitute in every day use and is intuitively scaled in between Red and Green, analogous to the meaning of Yellow in widely used triage methods. The colour-coding Black has not been implemented in this triage meaning, due to uncertainty of a certain death based on pure sensor data, this code is usually given to a subject that is already dead or too severely wounded to be rescued. Only the substitute use of the Shock Index for non GCS-based triage has been directly referenced in literature for military use. The other diagnosable conditions are used in GCS-based triage, and might provide value in the presented modelled military triage method. Furthermore, these diagnosed conditions are considered dangerous conditions and are flagged according to the implemented three colour-coding scheme.

The weighted diagnosed condition system and costs model create the possibility to value each condition in a more fine-grained manner, for counsel purposes, than is possible with the colour-coding triage. This could help differentiate the urgency considerations of conditions that require quite urgent attention and immediate attention, where the differen-

tiation between Orange and Red does not suffice. In some mission conditions an *undercooled* diagnosis might require immediate attention, while in another mission, the value of this diagnosis differs according to set mission parameters and survival chance of the subject. Therefore, the choice has been made to implement this weighted diagnosis system upon the more common colour based triage method, for flexibility reasons and pragmatic appliance in the demanding military settings.

Medical Condition Remediation. Within the scope of this research only one remedial action can be performed from the command centre, the presented *final rescue counsel* informs personnel to initiate a rescue mission upon the set parameters and perceived team diagnosed conditions.

However, a team interface without the costs and weighted diagnosed symptoms is available to the team agent, and could be the team medic. With this interface a more immediate intervention could be performed, possibly with further *human intervention required* triage steps.

These interventions could also be initiated from the command centre, possibly with the use of an individuals historic *timestamped diagnose sets* that could provide insights in the diagnosed symptom progression in real-time.

Pragmatic Appliance. Since the JaCaMo framework does support JADE natively, that in turn supports the https-protocol, a secure connection between Agents over public networks is feasible. However the non-JADE implementation has been used for development and also support network communication, a switch to JADE is only a configuration parameter in the MAS configuration, and has been tested with this implementation with a successful result.

The actual sensor controlling of physical sensors has been left out of the implementation, due to the sensor specific code that has to be implemented for each sensor in their own language. Moreover, the physical sensors were not available, and the goal was to provide a pragmatic set of *feasible* sensor within the context of this research. Another limitation is the mechanism that disables a sensor automatically when the sensor requires skin contact, this issue could be mitigated with the inclusion of a proximity sensor in those skin contact sensor packages. The Sensor Artefact implementation could implement this proximity threshold condition.

The presented solution provides adjustable properties to modify all threshold setting according to missions, disagreements on algorithmic evaluation of the proposed symptoms should be solved in AgentSpeak. The BDI Agent concepts are intuitive to non-programmers and allows them to engage in the development and refinement of the hypotheses set (Singh et al., 2016). Therefore Medical and Military professionals also have the potential to further develop these algorithms and symptom hypotheses specifications. The complete source code of this research is available at `https://github.com/pieterjoanUvA/Master_Thesis__Agent_Based_Modelling__Military_Health_System`, for the interested reader.

External Validity. The presented Military Health System could also be applied in different fields. This includes triage in civilian situations, with a disabled costs model. The presented MHS can already be adjusted to civilian suitable thresholds for symptom diagnosis. Examples of civilian appliance scenarios would be *elderly care* or *mass casualty* situations that occur with natural disasters or aircraft accidents.

Conclusion

This proof of concept solution for a modelled, Knowledge Managed, Multi-Agent Military Health System provides a basis for the appliance of contemporary available sensor equipment, dealing with the accompanying uncertainties and reasoning towards reliable sensor measurement conclusions. These *valid measurements* are thereafter employed in an modelled medical, developed in AgentSpeak with *descriptive logic*, and partially with a ProLog rule syntax. These *valid measurements* produce per Agent sets of *diagnosed conditions* that are evaluated with a triage colour-coded schema. According to the adjustable weights of each condition, a human subject is either counted or not in the *total rescue counts* according to agreements of the *institution*.

Lastly a possible controversial concept is the valuation of a human life in the form of an economic cost, with the potential capability to appoint a different value to a soldier and rescue personnel. If the cost of potential human life losses is greater than the possible costs, a positive counsel for *rescue initiation* is displayed. The Agent based modelling of the specified Military Health System combined with the actual developed software application (van Voorst Vader, 2018), demonstrating the automated simulation goal of this Multi-Agent modelled Military Health System. With the development of a demonstrable software application, this research is considered successfully completed.

Future Research. Future efforts could be made in refining analysis possibilities, cooperation with an army supplier or a country's national military division. Also implementations with real sensor equipment would be valuable, especially when cooperating with an actual military division. Dynamic threshold and outlier algorithms could be adjusted on a per sensor base, proximity sensor trigged package disablement tested, and possible implementation of a team medic signalling method for more immediate triaged critical health condition mitigation. Possible diagnose extensions could be researched in cooperation with medical and military professionals and could consider *fatigue* conditions and signalling these on an individual or team level with augmented reality methods *(e.g. vibrating wristbands or smart glasses)* The military requirements can also be quite remarkable, moreover the price differences between commercially available and military sufficient equipment and systems.

References

Abdali-Mohammadi, F., Bajalan, V., & Fathi, A. (2015). Toward a fault tolerant architecture for vital medical-based wearable computing. *Journal of medical systems*, *39*(12), 149.

Anliker, U., Ward, J. A., Lukowicz, P., Troster, G., Dolveck, F., Baer, M., ... others (2004). Amon: a wearable multiparameter medical monitoring and alert system. *IEEE Transactions on information technology in Biomedicine*, *8*(4), 415–427.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., & Santi, A. (2013). Multi-agent oriented programming with jacamo. *Science of Computer Programming*, *78*(6), 747–761.

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in agentspeak using jason* (Vol. 8). John Wiley & Sons.

Bourbia, R., Dib, L., & Benrazek, A.-E. (2016). An agent-based architecture for a distributed mis. In *Computer systems and applications (aiccsa), 2016 ieee/acs 13th international conference of* (pp. 1–2).

Buller, M. J., Tharion, W. J., Hoyt, R. W., & Jenkins, O. C. (2010). Estimation of human internal temperature from wearable physiological sensors. In *Iaai.*

Buller, M. J., Tharion, W. J., Karis, A. J., Santee, W. R., Mullen, S. P., Blanchard, L. A., & Hoyt, R. W. (2007). *Demonstration of real-time physiological status monitoring of encapsulated 1st civil support team-weapons of mass destruction (cst-wmd) personnel* (Tech. Rep.). Building 42 - Kansas Street Natick, MA 01760: Army Research Inst of Environmental Medicine Natick ma Biophysics and Biomedical Modeling Div.

Cunha, R., Billa, C., & Adamatti, D. (2017). Development of a graphical tool to integrate the prometheus aeolus methodology and jason platform. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, *6*(2), 57–70.

de Boer, G. (2017). *Modelling the military wearable sensory ecosystem as a diagnosable system* (Unpublished master's thesis). University of Amsterdam.

Falzone, E., Pasquier, P., Hoffmann, C., Barbier, O., Boutonnet, M., Salvadori, A., ... Mérat, S. (2017). Triage in military settings. *Anaesthesia Critical Care & Pain Medicine*, *36*(1), 43–51.

Haque, S. A., Rahman, M., & Aziz, S. M. (2015). Sensor anomaly detection in wireless sensor networks for healthcare. *Sensors*, *15*(4), 8764–8786.

Hübner, J. F., Boissier, O., Kitio, R., & Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous agents and multi-agent systems*, *20*(3), 369–400.

Hübner, J. F., & Bordini, R. H. (2007). Developing a team of gold miners using jason. In *International workshop on programming multi-agent systems* (pp. 241–245).

Hubner, J. F., Sichman, J. S., & Boissier, O. (2007). Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, *1*(3-4), 370–395.

Jentsch, M., Ramirez, L., Wood, L., & Elmasllari, E. (2013). The reconfiguration of triage by introduction of technology. In *Proceedings of the 15th international conference on human-computer interaction with mobile devices and services* (pp. 55–64).

Johnston, W., & Mendelson, Y. (2004). Extracting breathing rate information from a wearable reflectance pulse oximeter sensor. In *Engineering in medicine and biology society, 2004. iembs'04. 26th annual international conference of the ieee* (Vol. 2, pp. 5388–5391).

Koydemir, H. C., & Ozcan, A. (2018). Wearable and implantable sensors for biomedical applications. *Annual Review of Analytical Chemistry*(0).

Kramer, M., Lobbestael, A., Barten, E., Eian, J., & Rausch, G. (2017). Wearable pulse oximetry measurements on the torso, arms, and legs: A proof of concept. *Military medicine*, *182*(suppl_1), 92–98.

Newgard, C. D., Rudser, K., Hedges, J. R., Kerby, J. D., Stiell, I. G., Davis, D. P., ... others (2010). A critical assessment of the out-of-hospital trauma triage guidelines for physiologic abnormality. *The Journal of trauma*, *68*(2), 452.

Omicini, A., Ricci, A., & Viroli, M. (2008). Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, *17*(3), 432–456.

Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent systems: A practical guide* (Vol. 13). John Wiley & Sons.

Pasquier, P., Tourtier, J., Boutonnet, M., Falzone, E., & Mérat, S. (2012). The shock index: A further simple tool of triage in combat casualties. *Injury*, *43*(7), 1230.

Qi, J., Yang, P., Min, G., Amft, O., Dong, F., & Xu, L. (2017). Advanced internet of things for personalised healthcare systems: A survey. *Pervasive and Mobile Computing*, *41*, 132–149.

Rao, A. S. (1996). Agentspeak (l): Bdi agents speak out in a logical computable language. In *European workshop on modelling autonomous agents in a multi-agent world* (pp. 42–55).

Ricci, A., Piunti, M., & Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, *23*(2), 158–192.

Ricci, A., Piunti, M., Viroli, M., & Omicini, A. (2009). Environment programming in cartago. In *Multi-agent programming:* (pp. 259–288). Springer.

Roloff, M. L., Stemmer, M. R., Hübner, J. F., Schmitt, R., Pfeifer, T., & Hüttemann, G. (2014). A multi-agent system for the production control of printed circuit boards using jacamo and prometheus aeolus. In *Industrial informatics (indin), 2014 12th ieee international conference on* (pp. 236–241).

Singh, D., Padgham, L., & Logan, B. (2016). Integrating bdi agents with agent-based simulation platforms. *Autonomous Agents and Multi-Agent Systems*, *30*(6), 1050–1071.

Toledo, C. M., Bordini, R. H., Chiotti, O., & Galli, M. R. (2011). Developing a knowledge management multi-agent system using jacamo. In *International workshop on programming multi-agent systems* (pp. 41–57).

Uez, D. M., & Hübner, J. F. (2014). Environments and organizations in multi-agent systems: From modelling to code. In *International workshop on engineering multi-agent systems* (pp. 181–203).

van Voorst Vader, P. J. (2018). *Source code – multi-agent based modelled military health system* (Master's thesis, University of Amsterdam). Retrieved from `https://github.com/pieterjoanUvA/Master_Thesis_Agent_Based_Modelling__Military_Health_System/`

Vora, S., Dandekar, K., & Kurzweg, T. (2015). Passive rfid tag

based heart rate monitoring from an ecg signal. In *Engineering in medicine and biology society (embc), 2015 37th annual international conference of the ieee* (pp. 4403–4406).

Vora, S., & Kurzweg, T. (2016). Modified logistic regression algorithm for accurate determination of heart beats from noisy passive rfid tag data. In *Biomedical and health informatics (bhi), 2016 ieee-embs international conference on* (pp. 29–32).

Wendelken, S., McGrath, S., & Blike, G. (2003). A medical assessment algorithm for automated remote triage. In *Engineering in medicine and biology society, 2003. proceedings of the 25th annual international conference of the ieee* (Vol. 4, pp. 3630–3633).

Weyns, D., Omicini, A., & Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, *14*(1), 5–30.

Zatelli, M. R., & Hübner, J. F. (2014). The interaction as an integration component for the jacamo platform. In *International workshop on engineering multi-agent systems* (pp. 431–450).

Zhang, Q., Zhou, D., & Zeng, X. (2017). Highly wearable cuff-less blood pressure and heart rate monitoring with single-arm electrocardiogram and photoplethysmogram signals. *Biomedical engineering online*, *16*(1), 23.

Appendix A
Prometheus AEOlus diagrams

Additional Prometheus AEOlus (Uez & Hübner, 2014) Agent modelled diagrams are depicted in Figures A2, A3, A4, A5, A6, A7, A8, with the legend available in Figure A1. These models were created during the iterative process of designing the MAS. These models depict the use of various elements in plans with a stream of subsequent plans, *Internal and Artefact actions* that are called from within plans, and *outgoing Speech Acts* that trigger *incoming Speech Act* plans at the receiving Agent.

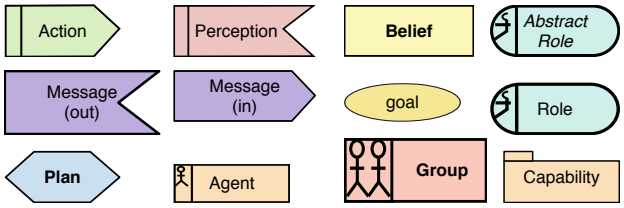

Figure A1. legend of Prometheus AEOlus diagrams *(Uez & Hübner, 2014)*

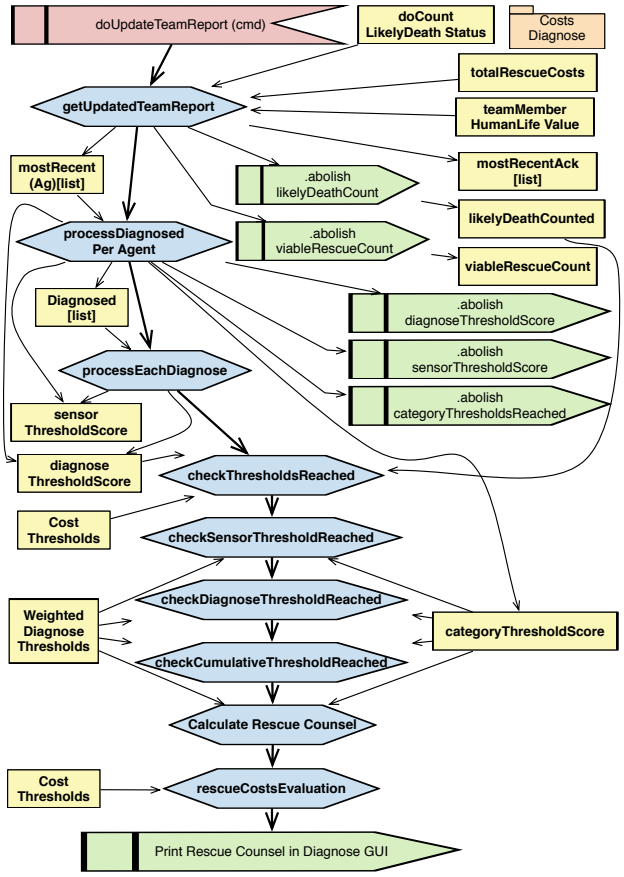

Figure A2. Command Centre Agent – Costs and Thresholds capability

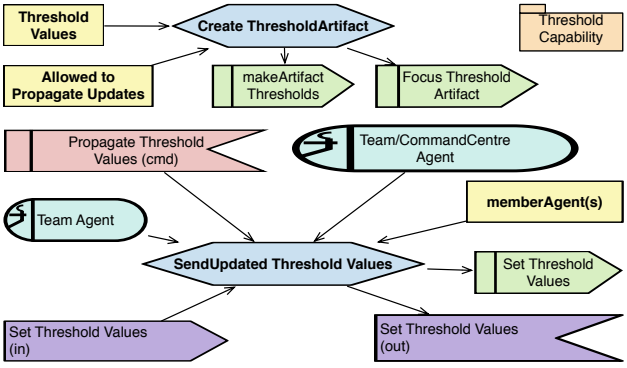

Figure A3. Threshold Capability model

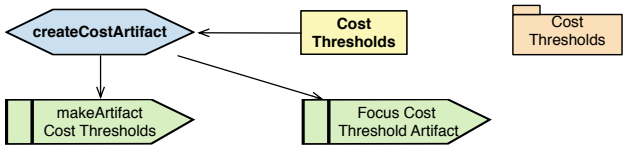

Figure A4. Weighted diagnosed symptom thresholds and costs model thresholds initialisation plans

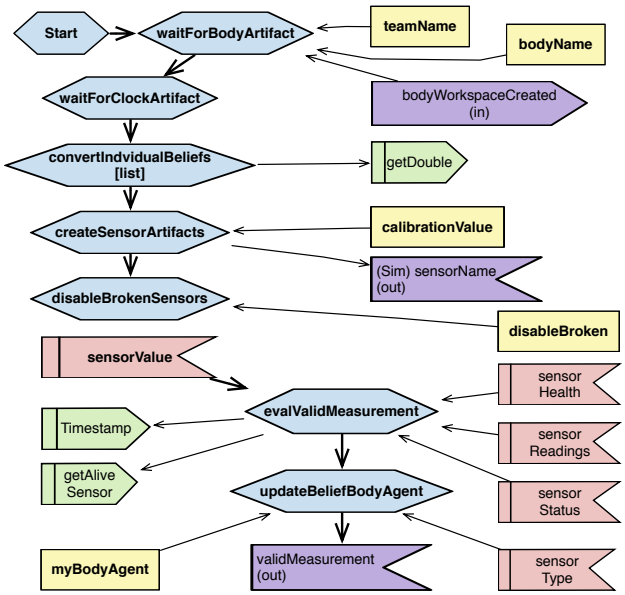

Figure A5. Sensor Agent model

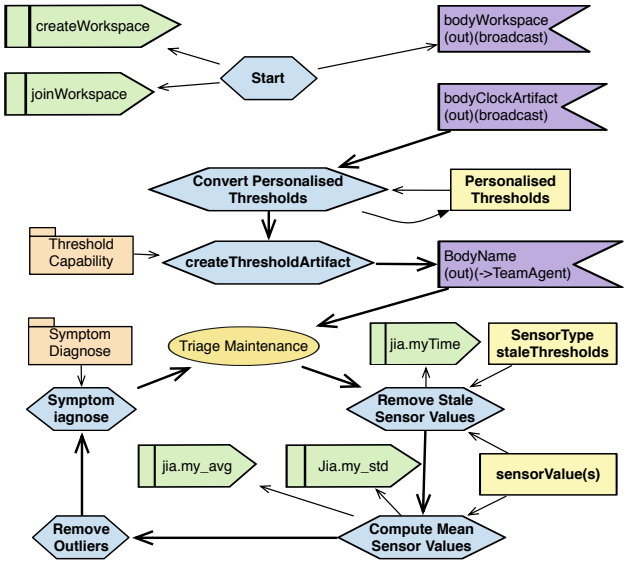

Figure A7. Body Agent plans

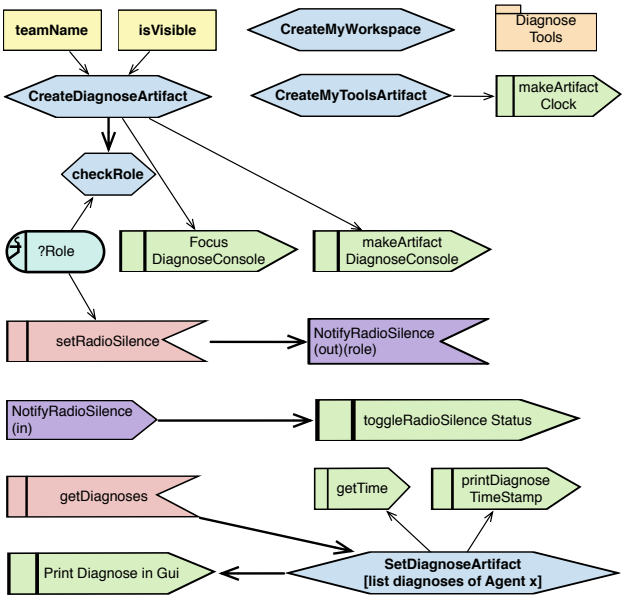

Figure A6. Diagnose Capability model

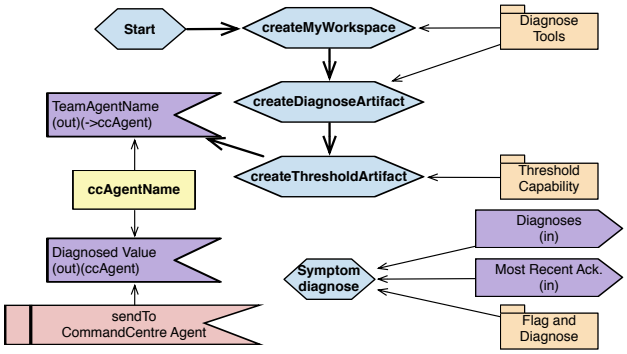

Figure A8. Team Agent Overview model

Appendix B
Graphical User Interfaces

Graphical User Interfaces of the Military Health System, that are implemented via CArtAgO Artefacts, are depicted in Figures B1–B12, where Figure B7 is the resulting view of the *GetResult* button in Figure B1. The depicted *Symptom Console* (Figure B5) is only used during development and validation of the individual diagnosable symptoms. The various implementations (Figures B7, B8, B9, B10) of the *Diagnose Console* show differentiations of one Artefact according to the Agents' role in the MAS.

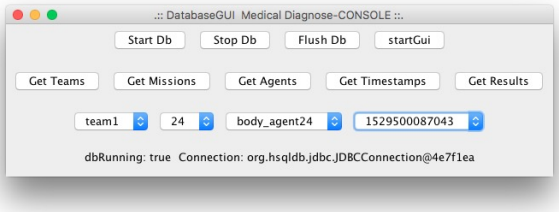

Figure B1. Database datapoint selection interface

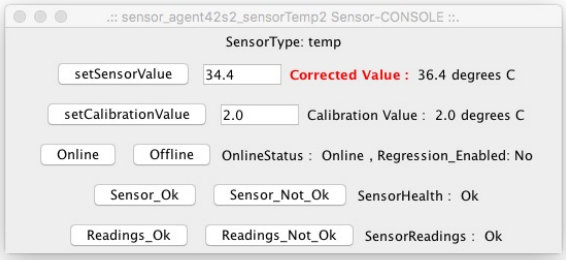

Figure B2. Sensor Console without enabled dynamic threshold evaluation.

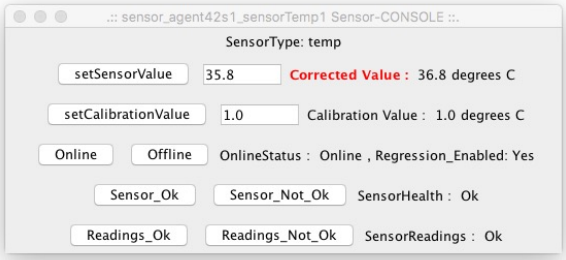

Figure B3. Sensor Console with enabled dynamic threshold evaluation and value within dynamic threshold.

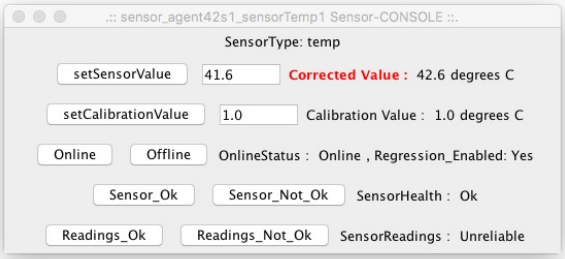

Figure B4. Sensor Console with dynamic threshold evaluation enabled and unreliable measured value, a value larger than dynamic threshold calculation.

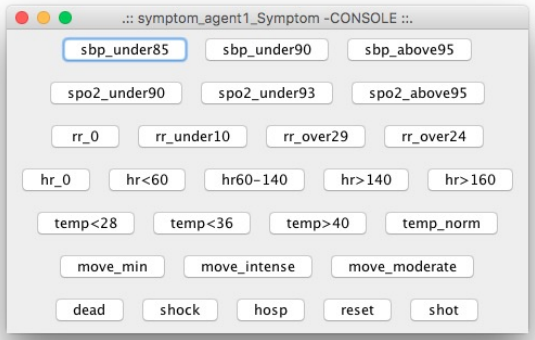

Figure B5. Symptom Console implementation.

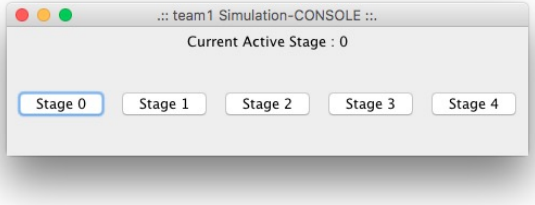

Figure B6. Staged scenario simulation console

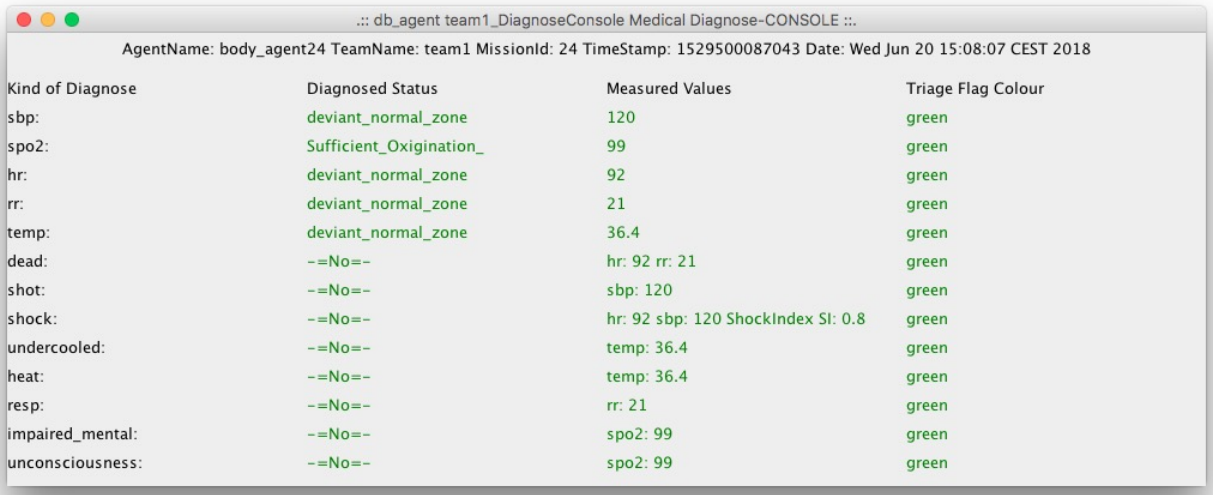

Figure B7. Database Agent timestamped resultset interface.

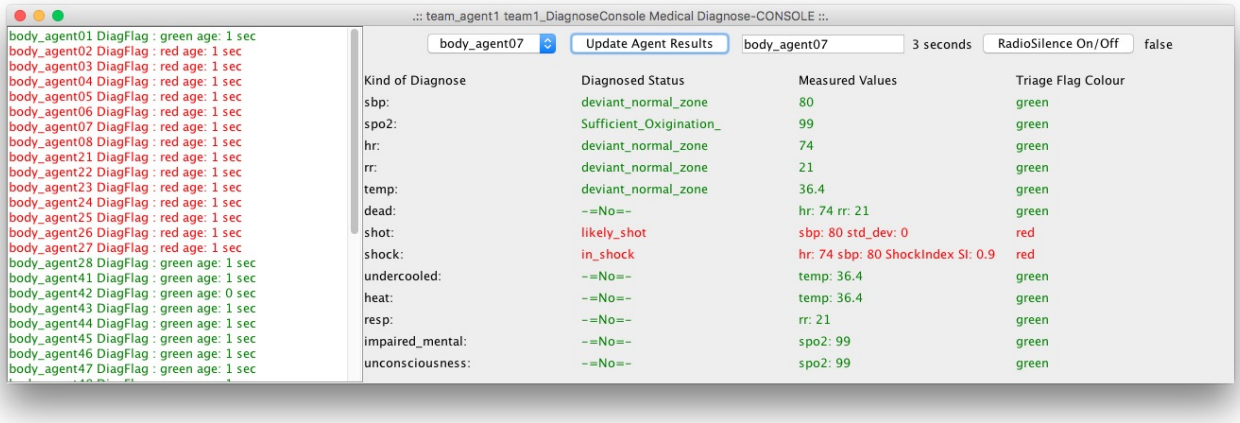

Figure B8. Diagnose Console of a Team Agent.

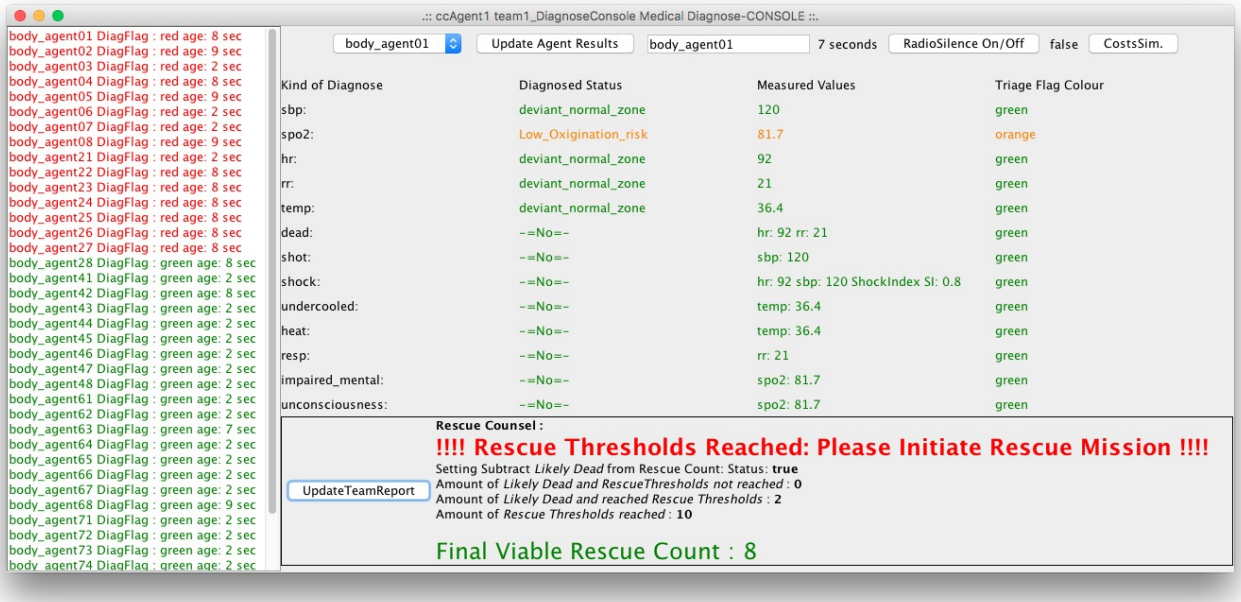

Figure B9. Diagnose Console of a Command Centre Agent, with Final Rescue Counsel.

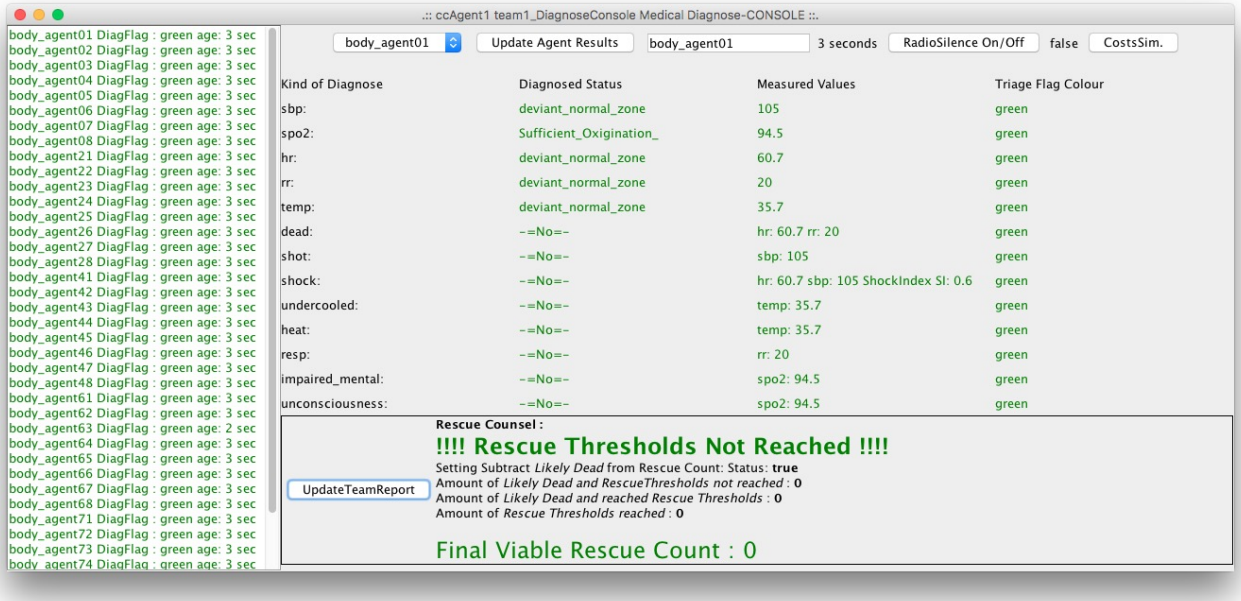

Figure B10. A Command Centre Agent Diagnose Console with a complete green result set.

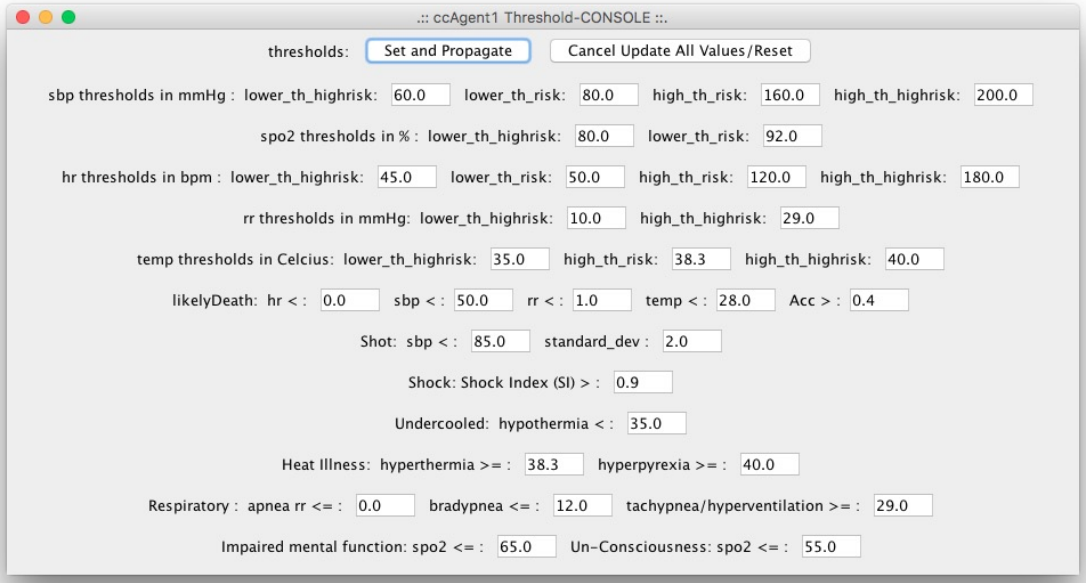

Figure B11. The Sensor Risk Zone and Symptom diagnosis threshold settings.

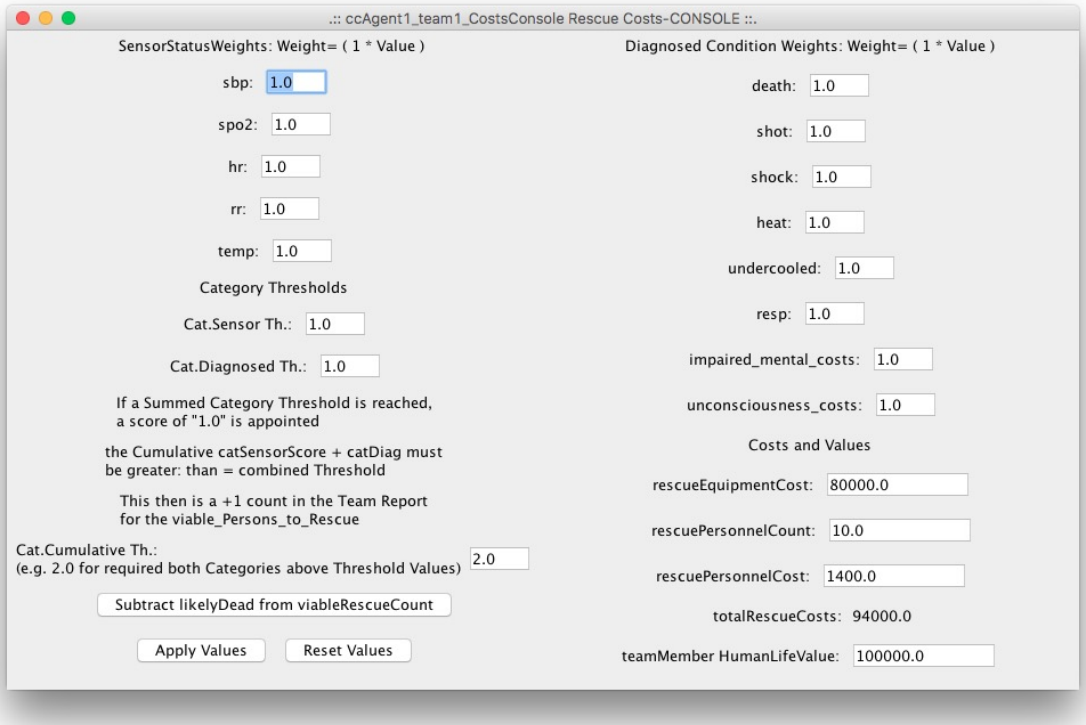

Figure B12. Weighted Symptom and Costs model threshold console.

Appendix C
JaCaMo Implementation

Outlier Detection AgentSpeak. Source Code C 1 depicts the plans for converting the *.jcm configuration file parameters* into Doubles with the use of *cartago.invoke_obj....*, this previously discussed method proved reliable in the high volume testing. Furthermore the *outlier removal* functionality is presented that implements the custom internal action *jia.my_std_eval(....)* for possible outlier detection and evaluation. For demonstration purposes this functionality has been restricted to the *temperature sensor*, and has a per Sensor Agent configurable belief *outlierRemove(true)*. This plan evaluates every value individually, removes the outliers, and recalculates the new *outlier excluded* averages. These values are then the input for the *Diagnose Capability* plans, available in the Master Thesis GitHub repository (van Voorst Vader, 2018).

```
/*  convert personalised thresholds from .jcm file from string to Double, since negative values not allowed */
+!convertPersonalisedThresholds : .findall(personalised(Pname, Pvalue),personalised(Pname,Pvalue),L)
& .length(L,Length) & Length >= 1
<- !convertPersonalised(L).
+!convertPersonalisedThresholds.

+!convertPersonalised([]).
+!convertPersonalised([personalised(Pname,Pvalue)|T]) : true
<-  cartago.invoke_obj("java.lang.Double",parseDouble(Pvalue),Y);
-personalised(Pname,Pvalue)[source(_)];
+personalised(Pname,Y);
!convertPersonalised(T).

+!computeMeanSensorValues : true
<-  .findall(ST,stale(ST,Th),L);
    .abolish(avSen(_,_,_));
    .abolish(avSen(_,_,_,_,_));
    !computePerType(L).
+!computePerType([]).

/* jia.my_std_eval(SensorValue, MeanValue, Desired_std_dev_Threshold, Calculated_std_dev_Value, min_threshold_Value(absolute value),
  ↪   ReturnString) */
+!removeOutliers([],Avg,StdDev,ST) :true <- !recalculateAverage(Avg,StdDev,ST).
+!removeOutliers([V|T],Avg,StdDev,ST) : jia.my_std_eval(V,Avg,1.1,StdDev,2,"positive")
<-  jia.myTime(TimeStamp); .wait(4);
    +evalPositive(V,ST,TimeStamp);
    !removeOutliers(T,Avg,StdDev,ST).
+!removeOutliers([V|T],Avg,StdDev,ST) : jia.my_std_eval(V,Avg,1.1,StdDev,2,"negative")
<-  !removeOutliers(T,Avg,StdDev,ST).
+!recalculateAverage(Avg,StdDev,ST) : .findall((V),evalPositive(V,ST,TimeStamp),ReCalcList) & .length(ReCalcList,X) & X > 1
<-  jia.my_avg(ReCalcList,AvgNew);
    jia.my_std(ReCalcList,StdDevNew);
    +avSen(ST,AvgNew,StdDevNew).
+!recalculateAverage(Avg,StdDev,ST).
/* not suitable for sbp drop because that drop is an outlier, specific for temp sensor demo profile */
+!computePerType([ST|T]) : .findall((V),vM(V,SN,MTime,SId,ST),ML) & outlierRemove(true) & .length(ML,X) & X > 1 & ST == ("temp")
<-  jia.my_avg(ML,Avg);
    jia.my_std(ML,StdDev);
    .abolish(evalPositive(_,ST,_));
    !removeOutliers(ML,Avg,StdDev,ST);
    !computePerType(T).

+!computePerType([ST|T]) : .findall((V),vM(V,SN,MTime,SId,ST),ML) & ST == ("sbp") & .length(ML,X) & X > 1
<-  .findall(v(MT,Valu),vM(Valu,SNm,MT,SIdm,ST),MinMaxL);
    .min(MinMaxL,v(MT,SBPold));
    .findall(b(MTm,Valum),vM(Valum,Snmm,MTm,SIdmm,ST),MaxL);
    .max(MaxL,b(MTm,SBPnew));
    jia.my_avg(ML,Avg);
    jia.my_std(ML,StdDev);
    +avSen(ST,Avg,StdDev);
    +avSen(ST,Avg,StdDev,SBPold,SBPnew);
!computePerType(T).
+!computePerType([ST|T]) : .findall((V),vM(V,SN,MTime,SId,ST),ML) & ML \== []
<-  jia.my_avg(ML,Avg);
    jia.my_std(ML,StdDev);
    +avSen(ST,Avg,StdDev);
    !computePerType(T).
/* Plan for Empty set of Valid Measurements*/
+!computePerType([ST|T])  <- !computePerType(T).
```

Source Code C 1: Excerpt of *Sensor Agent Outlier detection and valid measurement* plans

Outlier Detection Custom Internal Action. Source Code C 2 shows the custom Jason internal action implementation of the outlier evaluation function. The input values given in AgentSpeak are received in *args[0] – args[4]*, the return value is implemented in *args[5]* that maps directly to the *positive or negative* value in the context evaluation of the *+!removeOutliers(....)* plans. The evaluation function also includes a configurable minimum threshold value to allow a minimum variation in values. When quite stable values have been measured for a longer period, that logically implies a very low standard deviation, the minimum threshold value allows a new deviant value to be evaluated *positive* and is therefore included in the majority voting process of the Agents' plans.

```
package jia;

import jason.*;
import jason.asSemantics.*;
import jason.asSyntax.*;

public class my_std_eval extends DefaultInternalAction {

@Override
public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
    // execute the internal action
    final double value = (double)((NumberTerm)args[0]).solve();
    final double meanvalue = (double)((NumberTerm)args[1]).solve();
    final double desired_std_th = (double)((NumberTerm)args[2]).solve();
    final double std_dev_value = (double)((NumberTerm)args[3]).solve();
    final double min_threshold = (double)((NumberTerm)args[4]).solve();

    double difference = meanvalue - value;
    if (difference == 0 | difference > 0 ) {
        // nothing to do
    } else if (difference <= 0) {
        difference = difference * -1;
    }
    if (std_dev_value == 0) {
        return un.unifies(args[5], new StringTermImpl("positive"));
    }
    if (difference < (min_threshold) | difference < (std_dev_value * desired_std_th) ) {
        return un.unifies(args[5], new StringTermImpl("positive"));
    } else if (difference > (std_dev_value * desired_std_th) ) {
        return un.unifies(args[5], new StringTermImpl("negative"));
    }
        return un.unifies(args[5], new StringTermImpl("negative"));
    }
}
```

Source Code C 2: Custom internal action *Outlier Detection*.

Dynamic Threshold. The linear regressed dynamic threshold, predicted next value is presented in Source Code C 3. The first function *setSensorValue(....)* checks the *observable property* of the sensor if the regression functionality is enabled. When this sensor functionality is enabled the function *evalRegression(value)* is called, that returns a Boolean value used in the *Sensor Readings* observable property. The *evalRegression* functionality then *predicts* the next value, while incorporating the last 30 stored values. For handling sudden change in measured values after a *steady* period, the minimum threshold value is applied to create a useful minimum allowed bandwidth. The Sensor Console with disabled dynamic threshold can be viewed in Appendix B Figure B2, while versions with enabled regression are depicted in Appendix B Figures B3 and B4.

```
import org.apache.commons.math3.stat.regression.*;

@OPERATION void setSensorValue(double value) {
    String regressionStatus = getObsProperty("regressionEnabled").getValue().toString();
    if (regressionStatus.equals("No")) {
        display.sensorValueDisplay.setText(String.valueOf(value));
        double new_input = value - Double.parseDouble(display.calibrationValueDisplay.getText());
        display.valueToInput.setText(""+new_input);
        getObsProperty("sensorValue").updateValue(value);
    } else if (regressionStatus.equals("Yes")) {
        boolean setSensorReadingBool = evalRegression(value);
        if (setSensorReadingBool == false) {
            setSensorReadings("Unreliable");
        } else if (setSensorReadingBool == true) {
            setSensorReadings("Ok");
        }
        display.sensorValueDisplay.setText(String.valueOf(value));
        double new_input = value - Double.parseDouble(display.calibrationValueDisplay.getText());
        display.valueToInput.setText(""+new_input);
        getObsProperty("sensorValue").updateValue(value);
    }
}
boolean evalRegression(double value) {
    int toPredictCounter = 0;
    double stdErr = 0;
    double myPrediction = 0;
    double myLastValue = 0;
    double minThreshold = 2;
    boolean validValue = false;

    SimpleRegression regression = new SimpleRegression();
    /* initial list filling function removed .... */
    toPredictCounter++;
    myPrediction = regression.predict(toPredictCounter);
    stdErr = regression.getSlopeStdErr();
    double rawDifference = value - myPrediction;
    if(rawDifference < 0) {
        rawDifference = rawDifference * -1;
    }
    double diffLastCurrentValue =  value - myLastValue;
    if(diffLastCurrentValue < 0) {
        diffLastCurrentValue = diffLastCurrentValue * -1;
    }
    if (rawDifference < stdErr | rawDifference < minThreshold ) {
        validValue = true;
    }
    if (diffLastCurrentValue < stdErr | diffLastCurrentValue < minThreshold ) {
        validValue = true;
    }
    double dynamicThreshold = myLastValue - myPrediction;
    if(dynamicThreshold <0) {
        dynamicThreshold = dynamicThreshold * -1;
    }
    dynamicThreshold = dynamicThreshold + stdErr;
    if (rawDifference < dynamicThreshold | diffLastCurrentValue < dynamicThreshold) {
        validValue = true;
    }
    myRegressionValues.remove(0);
    myRegressionValues.add(value);

    return validValue;
}
```

Source Code C 3: Excerpt of *Sensor Artefact Console* implementation.

Outlier detection and dynamic threshold general considerations. Sudden large changes in both outlier detection and the dynamic threshold calculation could reject measured values for a prolonged period. This will have the effect that diagnoses considering these measured values will be unavailable. However, since measured values are still stored for either their maximum valuable lifetime (outlier detection) or last 30 values (dynamic threshold), this effect will resolve over time and diagnoses will become available again for display. The modelled minimum threshold value is set at a value of two, and should be validated with medical specialists that know *normal* change of values over time of the specific sensors. However, if this value is in fact too small, the only effect is that more often a diagnosable condition is temporary unavailable.

Diagnose Console. The partially depicted Diagnose Console Artefact implementation in Source Code C 4, shows the temporary *saveMsg* function in case of an active *Radio Silence* situation. This *saveMsg* function implements the already discussed custom *Diagnosed* model to store each Body Agents' diagnoses. When the event *+radioSilence(false)* is perceived by the Team Agent, the *sendingMsg* action is called within the Team Agents' plans. Each individual diagnose is *signalled* towards the *focussed* Team Agent, that perceives this event in the form of *+cmd("sendToCC",Kind,Status,Values,Flag,Name,Team,TimeStamp)*, and reacts accordingly to the available plans for this event. The *printMsg* action is executed when the plans for *mostRecentAck* have been finished, this function then updates the Agent drop-down menu and left column *Agent Diagnose* list in alphabetic order. The drop-down list update implements the before mentioned reduced model matching on *AgentDiag(agentnames)*. The left column listing also implements the *List<AgentDiag>* model to store all *most recent* aggregated colour flagged team triage results. The resulting visual representation can be viewed in Appendix B in Figure B10.

```
List<AgentDiag> listAgentDiag = new ArrayList<AgentDiag>();
List<Diagnosed> listDiagnosed = new ArrayList<Diagnosed>();
@OPERATION void saveMsg(String kind, String status, Object values, Object flag, String name,  String team, long timestamp) {
    String s_values = values.toString();
    String s_flag = flag.toString();
    listDiagnosed.add(new Diagnosed(kind,status,s_values,s_flag,name,team,timestamp));
}
@OPERATION void sendingMsg(boolean bool) {
    /* for preventing a ConcurrentModificationException */
    List<Diagnosed> templist = new ArrayList<Diagnosed>();
    for(Diagnosed diag : listDiagnosed) {
        String name = diag.getName();
        String kind = diag.getKind() ;
        Integer flag = Integer .parseInt(diag.getFlag());
        String status = diag.getStatus();
        String values = diag.getValues();
        String team = diag.getTeam();
        long timestamp = diag.getTimestamp();
        templist.add(diag);
        signal("cmd","sendToCC",kind,status,values,flag,name,team,timestamp);
    }
    listDiagnosed.removeAll(templist);
}
@OPERATION void printMsg(Object msg, String colour, long timestamp, long currtime){
    String agentnames = msg.toString();
    AgentDiag search = new AgentDiag(agentnames);
    /* check if item already exists in JComboBox, otherwise add */
    if(display.model.getIndexOf(msg) == -1) {
        display.myBox.addItem( msg );
    }
    /* remove new sent agent_result from ArrayList<AgentDiag> (customModel) */
    listAgentDiag.remove( search);
    listAgentDiag.add(new AgentDiag(agentnames, colour, timestamp));
    Collections.sort(listAgentDiag, new AgentDiagChainedComparator(new AgentDiagNameComparator()) );
    String message = "";
    /* ArrayList Printing Loop, inexplicitly calls toString(); of model_class */
    for (AgentDiag name : listAgentDiag) {
        message = message + name;
    }
    display.myLabel.setText(message);
}
```

Source Code C 4: Excerpt of *Diagnose Artefact Console* implementation.

Appendix D
Towards a medical ontology

The medical ontology is restricted by many factors that limit the amount of feasible diagnoses. Firstly, the available sensors that are in compliance with the military requirement of unobtrusiveness. This requirement limited the available sensors types to HeartRate, Respiratory Rate, Temperature, Pulse Oximeter, acceleration and gyroscopes.

With the available sensor types that could be applied in a military setting and are unobtrusive for soldiers, a research was performed to gather values that are actually used in medical triage, symptom definitions and topics described in the *Methods* section.

For each sensor applicable critical values *(high-risk zone)* are described and mapped to a *Red* Flag, the values between deviant and critical *(risk zone)* are flagged *Orange* when available. The availability of Orange values for sensor types varies, when such values where not available, these values were not included in the Table 1. Not every critical value mapped to a specialist medical term like *hyperpyrexia*. All values in the Normal and Deviant zones are considered to be not alarming at all, and are therefore aggregated in the *Green* flagged zone. Further diagnosable conditions are flagged *Red* when a person is likely to be incapacitated or could pose a threat to themselves or the team in a military setting.

After this aggregation of sensor types and sensor value zones, according to literature sources, a search was performed for additional symptoms that can be diagnosed without human intervention in this remote triage setting.

The exact values for these diagnosable conditions is still debated upon in the field, and when the specialists do not agree, a value is arbitrary chosen within that bandwidth of disagreement values. However, these values can be easily adjusted via the Artefact Consoles according to the institutional norms.

Composed Conditions. The impaired mental function and unconsciousness conditions are based on the assumption that the vast majority of humans suffers of these ailments with the applied SpO_2 values. However, this is not certain for everyone, and could be mitigated by threshold adjustments. According to Wendelken et al. (2003), $SpO_2 >$ 70% is sufficient, while Anliker et al. (2004) states that < 80% is alarming.

The heat related ailments also are up for discussion, and the arbitrary bandwidth selection of these named ailments have been made.

The alarming shock condition of SI $\geq$0.9 is mostly agreed upon, also in the military field.

The shot condition is partly based on the research of de Boer (2017) for the *sudden blood pressure drop* hypotheses. The alarming value for immediate medical care is argued upon and differs from 85mmHg to 90mmHg. However, the condition is called *Likely Shot*, since measurements do suggest that the sudden drop or dangerously low SBP a puncture wound. This condition is likely in a military setting, however this condition could also happen due to knife wounds or broken bones puncturing the skin or even severe internal bleeding without external punctures. The low blood pressure condition remains severely alarming though.

The *Likely Death* scenario is based on common sense, not explicitly reference in literature. This condition excluded the Oxygen saturation measurements, since when blood flow stops, when the hart stopped beating, the available oxygen in the blood vessels does not have to be consumed by the surrounding tissues. This condition applies sensible values like nil values for heart rate, blood pressure, respiratory rate and a body temperature below the limit of *undercooled*. The Table 2 also shows that at least a mandatory HR and RR must be zero. This minimum has the following argumentation: A RR of zero, can be voluntary *apnea*, and can be observed while someone submerges in water while holding their breath. Therefore this condition alone should not be considered enough to conclude that someone is dead.

Furthermore, when the heart rate is measured with a nil value, can have multiple causes that could be temporary. The combination of these RR and HR measurements do provide a basis to diagnose a person *Likely Dead*, based on sensor measurements alone. To improve the chance of detecting disconnected sensors and the reliance on a non-bodycontact sensor like acceleration or gyroscope, that also are less sensitive to failure, actually disable the possibility for a positive evaluation of this *Likely Dead* condition when there is still movement above some minimal threshold. This implies that there is something else wrong. This diagnosis is also distinctly diagnosed with the flag *Orange* and is therefore not counted in any rescue counsel variation.

Modelled diagnosable respiratory related conditions, that include shallow breathing symptoms are aggregated or excluded, since breast band based sensors are unsuitable for an eight-hour working day, with comfortability and perceived job hindering issues (Buller et al., 2007). Therefore, only absolute Respiratory Rate conditions are included and not ones that rely on very subtle acceleration variation measurements in addition to RR counting. Therefore, the conditions tachypnea and hyperventilation are aggregated and are indistinguishable without a breast band sensor.

Possibly, other conditions could be constructed with the proposed sensor types with more specialised medical knowledge. This expansion of conditions should be considered feasible since the before mentioned *intuitiveness* of the AgentSpeak language and concepts for medical specialists with little programming experience.

With this argumentation of the already presented medical ontology, a pragmatic diagnosable set of conditions have been created that can be modelled in a *diagnosable system* for military health triage scenarios.