Implementing *GliderAgent* – an agent-based decision support system for glider pilots

Jan J. DOMAŃSKI [a], Radosław DZIADKIEWICZ [a], Maria GANZHA [b,c,1], Andrzej GAB [d] and Mariusz M. MESJASZ [a] Marcin PAPRZYCKI [b]

[a] *Warsaw University of Technology, Warsaw, Poland*
[b] *Systems Research Institute Polish Academy of Sciences, Warsaw, Poland*
[c] *University of Gdańsk, Gdańsk, Poland*
[d] *Silesian University in Opava, Opava, Czech Republic*

Abstract.
The aim of this chapter is to discuss practical issues encountered when a skeleton of the *GliderAgent*—a multi-agent system supporting glider pilots in various navigation and pilotage scenarios—was implemented. The GliderAgent implementation integrates two agent environments discussed in other chapters of this book (*Jade* and *MAPS*), as well as the *XCsoar* pilotage software, and the *OpenStreetMaps* GIS system. In addition to presenting the basic design of the GliderAgent system, the technical issues solved during its implementation are discussed. Finally, examples of implemented capabilities are illustrated.

Keywords. Decision support system, software agents, gliding, sensors

Introduction

Today, computer technology supports flight-related functions in commercial air planes, within airline companies, and in air traffic control. However, in the case of gliding (flying sailplanes/gliders), use of computer systems (electronic devices, in general) is limited. There are multiple reasons for lack of acceptance of "electronics" in gliding. Some of them are psychological, e.g. conservatism of pilots, manifesting itself in a belief that gliding requires special skills that are contrary to technological advances. However, important is also the fact that in engineless planes there is limited amount of available electric power, which would not be sufficient for a large number of electronic devices. However, even in gliding, we start to find "smartphones" and PDAs, running programs like the *XCsoar* [12] (or the *WinPilot* [11]), which resemble the vehicle GPS navigation systems and provide information like: (i) display of airspace areas, (ii) altitude required to complete a task, (iii) thermal profile of the area where the glider is located, (iv) depiction of a final glide through the terrain and around multiple waypoints, etc. Furthermore, in

[1]Corresponding Author: Systems Research Institute Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland; E-mail: maria.ganzha@ibspan.waw.pl.

the context of achievement flying (see, below) the so-called *loggers* are used. Loggers store and archive waypoints flown by. These devices have to be approved by the *Fèdèration Aèronautique Internationale* (*FAI*; [1]) to allow pilots to claim sport achievements (the *FAI Record Claim*). While some loggers use only the GPS positioning (simple *Position Recorders*), others have implemented more advanced technologies (e.g. the International Gliding Commission Approved *Flight Recorder*, which includes also a pressure altitude sensor, which allows to store not only waypoints, but also altitude data (for more information, see [5]).

Before proceeding further, let us discuss two gliding accidents, which provide context for our work. Additional accident reports have been discussed in [14], which also provided some of the material used in the initial sections of this chapter.

Both accidents happened in Poland. The first one took place on July 13th, 2009, during the starting procedure, using the so-called winch launching. Unfortunately, the starting cable didn't release properly, and remained attached to the glider. To warn the crew about this dangerous situation, the ground team made the standard procedural sign, as well as tried to radio the information in. Unfortunately, pilots haven't noticed the sign, while the cabin radio malfunctioned. The winch mechanic cut the cable off, to let the glider fly without pulling it down. Next, the glider flying down-wind has touched the electrical line with the hanging cable, making a short circuit. The cabin crew has not noticed the problem until they reached the final glide. Fortunately, they were able to successfully release the cable, and then landed safely. Of course, the primary problem was that pilots did not observe requirements of the starting procedure, which oblige them to observe ground for a while after the start; to check for signs informing them about problems. But we can imagine that if another communication channel was open, the ground team could deliver message about the problem, and this message could be "blinking" and/or "beeping," to demand attention. Obviously, it is possible that both the "PDA" and the radio malfunctioned simultaneously, but probability of such double-malfunction is much lower than that of a single one.

The second incident happened during a competition: the Polish Gliding Championship, Standard Class, on June 10th, 2010. The pilot started normally and flew for more than 3 hours. Unfortunately, after 6 PM the thermals subsided and he had to go back to the airfield. The altitude of the glider was too low to be able to reach the destination. During the landing preparation phase the pilot noticed that, on a closer field, there is a landed glider. So he tried to land on the same field, conjecturing that if one glider has landed there, it should be a good place for emergency landing. He didn't know that the glider on the field was broken because of landing on the terrain that was too "bumpy." Unfortunately, nobody could inform / warn him about this fact, and he has destroyed his glider as well. In this case two forms of help can be envisioned. (1) Informing the contest organizers in advance that there is a glider, which won't be able to reach the original airfield. As a result, organizers could contact the pilot and instruct him where he can find an acceptable place to land. (2) The first glider, which crash-landed on the bumpy terrain, could mark the field as not proper for landing, and send this information to other gliders in its close proximity. This message could then be propagated to other gliders, reaching them all (in a way similar to a query propagation in wireless sensor networks; see [14]).

0.1. Typology of glider flying

Discussion presented thus far indicates that there exist different types of glider flights (e.g. recreation and competition). Let us enumerate them and their main features.

1. *Training flights*, which can be divided into:
   a) *Flights with an instructor*—most of such flights are early-training flights where a student pilot gains basic skills—how to start, to land, how to turn the glider, etc.
   b) *Solo training flights*—conducted by a student alone. During these flights an instructor is located on the ground and should have a possibility to communicate with the student-pilot to correct the pilotage technique, give the advices, or warn about mistakes / dangers. Solo training flights are typically conducted above an airfield, or in a close distance to it. Here, distance of 20 km (or less) is usually treated as a safe one (e.g. for returning to an airfield). Nevertheless, what is considered a "safe distance" depends also on multitude of conditions, e.g. the flight height, and/or the weather.
2. *Recreational flying*—requires higher navigational skills, because of long distances flown away from the start airfield (more than 20 km). Recreational flying is very often conducted over areas not well known to the pilot. Here we can distinguish the following sub-categories:
   a) *Cross-country flying*—the easiest and safest form of recreational flying; typically flown over plains. Here, thermal soaring demands awareness of other gliders (if not flying alone in a single lift) to avoid collisions.
   b) *Ridge flying*—demands high skills, because of flying in close distances to slopes of hills and/or mountains. Due to the terrain characteristics, there could be a sudden variation of lifts and sinks encountered by a glider, and such conditions could be very dangerous. If there are more gliders in the same area, their pilots should be aware of positions of the others.
   c) *Lee-wave flying*—a dynamic phenomenon that helps pilots to gain several kilometers of height and make flights hundreds of kilometers long. Mostly observed in mountains, and thus requiring at least as advanced skills as in the case of ridge flying. However, further experience is required because of the nature of the wave, which typically involves very violent conditions for start, getting to the wave and landing. The main danger of the lee-wave flying is being caught above a "solid cloud." Clouds can develop so quickly that a pilot does not have a chance to come back down without getting through the clouds.
3. *Sport / competitive flying*—focused on achieving sport results according to the FAI or local rules. All sport flights must be conducted in accordance with specified rules and procedures (to be accepted as a sport achievement). Among sport flying events we can distinguish:
   a) *Badges and diplomas*—for example FAI's silver, gold badges, diamonds and diplomas achieved for completing soaring performances of: distance, duration and height gained.
   b) *Localized competitions* (Grand-Prix, Championship, etc.)—pilots meet and fly tasks together starting from the same place.

c) So-called *on-line contests*—pilots conduct flights alone, in their clubs, and then send their flight documentation to the contest organizer to be validated and scored.
d) *Record flights*—gaining national or world records, validated by a National Authority and/or the FAI.

0.2. Supporting glider pilots—preliminary observations

On the basis of the above described incidents, as well as identified types of glider flights, let us now identify some of the most important needs arising in flying gliders.

Training flying. It would be important to provide more information to the student pilot (flying a solo flight). Here we can distinguish the advice made available during the flight, and post-flight analysis. During the flight, the instructor could give some ready-to-use suggestions observing parameters transferred from the distant glider (advising on the basis of observing the glider using binoculars is not sufficient). In a post-flight session, if precise-enough data was collected, instructor and student could go over the completed flight, analyzing each important decision made by the trainee.

Cross-country flying. For this type of flying very useful would be finding thermals, as well as informing the ground team in advance that the glider cannot reach any airfield (primary or secondary); together with its precise location. The latter would help to prepare a glider trailer for fast bringing the aircraft back from the landing site. In case of an injury, it will greatly shorten the time of providing medical assistance; thus increasing pilot safety. Furthermore, pilots could find useful obtaining important meteo change reports, especially warnings about possible storms and thunders, delivered directly to their "smartphones" (and confirming to the ground that the message was received, without the need to use the radio by the pilot).

Ridge flying. This type of flying requires fast informing the ground station about any abnormal behavior or danger. The mountainous area generally is not appropriate for landing outside of existing airfields. Besides the slopes, they are mostly covered by forests and/or shrubs. Furthermore, note that in this case, the time between realizing the danger and the possible crash is much shorter than in cross-country flying. Therefore, the proposed system should provide, among others, autonomous communication from the glider to the ground station (to let the pilot concentrate on the pilotage).

Lee-wave flying. The violent nature of the flights using the *lee-wave* phenomenon, and the possibility of "shutting" the sky below the glider, require that the pilot regularly checks the weather condition over the airfield(s). Even worse, the lee-wave flying involves heights where the pressure is much lower and the oxygen is diluted. Staying above the 4000 meters (above the sea level) could lead to the altitude sickness [2], which is a direct danger for pilot's health (unconscious pilot cannot prevent the crash; see, description of the accident presented in [14]). These features of lee-wave flying suggest the support to be provided by the system, e.g. autonomous communication concerning the state of the glider and its pilot send to the ground station, and various warnings delivered to the pilot.

Sport flying. Finally, in *sport flying* we could encounter all of the above mentioned topics, as the competitions are conducted over the fields, and in the mountains.

Furthermore, the issue of making competitive flying interesting to spectators also becomes important (see, the next Section).

Obviously, this list is not exhaustive, for more information, see Section 1.1.

0.3. Electronic communication equipment in gliding

At the beginning of this chapter, we have stated that on board of gliders one can find devices capable of running "glider-GPS-software" and/or loggers. Obviously, there are also radios, which are the main, and the oldest, electronics on board of gliders. Let us now describe other electronic equipment that can be found in gliding.

There exist efforts to create a general anti-collision system, similar to the one used in commercial aviation, i.e. the Traffic Collision Avoidance System (TCAS) [9]. Here, for instance, the *FLARM* system [4] has been deployed; primarily in parts of Western Europe (German speaking countries and Scandinavia), as well as in Australia, and New Zealand. This system is based on communication between aircrafts equipped with devices designed to warn about a risk of a collision. The limitations to this system are: (a) very high specialization—not suitable for any use other than the collision avoidance, (b) low bandwidth transmission channel—limiting the amount of information that can be transmitted; thus reducing possibility of extending its functionality beyond crash avoidance, (c) very short range—making it unsuitable for avoiding collisions in the case of fast moving aircrafts. Here, note that sport gliders can fly even up to 300 km/h and it remains to be seen if the FLARM system will be capable of effectively dealing with such speeds.

Separately, in recent years, we observe an increasing interest in implementing services usually called "on-line tracking", which visualize in (almost) real-time positions of gliders during various contests (see Section 0.1). The goal of such systems is to make gliding (as a sport discipline) more audience-friendly—letting the audience "feel the competition," and deliver intermediate (temporary) scores (obviously, keeping those away from the pilots participating in the competition). We can differentiate these systems according to the transmission medium they utilize:

1. GSM/GPRS data transmission—e.g. LX GPS trackers [6] adopted from vehicles;
2. GSM/SMS data transmission—e.g. vPtracker [10] developed for the Club Class World Gliding Championship held in Elverum, Norway in 2004 and later commercialized;
3. satellite data transmission—i.e. the Yellowbrick [3] used last during the World Grand Prix Gliding Final in Santiago, Chile, in 2009;
4. APRS—mostly utilized by the OpenTracker+ [7] and the TinyTrak3 [8] solutions—based on the Amateur Radio technology.

Reviewing these approaches the following observations can be made. All GSM-based solutions (1,2) have problem with sending data (even when flying on relatively low altitudes). GSM/UMTS operators fix their BTS antennas to disperse the signal across the serviced *ground* area and are not interested in sending / receiving their signal to / from the airspace "above." Thus, the GSM-based solutions are (i) highly dependent on the GSM range height, (ii) have many breaks in transmission, and (iii) position data is sent relatively rarely—only when receiver antennas can be reached—mostly in intervals of several minutes, which limits usability of such data (the display is "jumpy" at best).

Main drawbacks of the satellite-based systems (3) are their cost and power consumption. The Yellowbrick solution uses the Iridium satellites which are 780 km away and in-flight devices can be rented only for a specified time-frame. The cost and battery life are dependent on the frequency of data transmission [13]. The main drawbacks of the APRS-based approaches (4) are: low or no infrastructure, low bandwidth, and being in an early-deployment phase.

Note also, that all these approaches are designed to deliver only a single, on-line tracking, service; rather than provide support for the glider pilot (see Section 0.2). Furthermore, since they typically have a very low communication bandwidth (as this is all that is needed for tracking), It would be extremely difficult to extend them to support more general functionalities.

1. *GliderAgent*—a glider pilot decision support system—requirements analysis

The *GliderAgent* is conceptualized as an agent-based decision support system: (a) helping a pilot in navigation and pilotage, (b) delivering additional flight information, (c) providing support in emergency situations, (d) facilitating communication, monitoring, logging of events, etc. It is not envisioned as replacing the existing systems (e.g. the XCsoar, or the WinPilot), but to integrate with them. However, it should be able to combine some functionalities, which currently run in separate applications, to facilitate better support for the pilot. Furthermore, the GliderAgent is to become a common platform for implementing services, existing and not yet known. On the basis of the material presented thus far, we can start by presenting an overview of a glider flying infrastructure, which can be found in Figure 1. In this figure, one can see main actors that participate in glider flying and are to be taken into account when creating the GliderAgent system (for a more complete discussion of the content of this Figure, see [14]). Let us now use information depicted in Figure 1 to introduce the requirements analysis (business and technical) for the GliderAgent system.

1.1. Business analysis

The GliderAgent system is to support needs of the glider pilot (see Section 0.2). Let us now state the basic system requirements.

Sharing information between gliders This is one of the key functionalities that the remaining ones depend on. The implemented system should allow for messages to be passed between various entities (see, Figure 1), such as glider to glider message exchange, communication with the ground station(s), and with proxy stations. Here, the proxy stations are understood as an infrastructure designed to forward messages between other entities in the system. Proxy stations are to be placed in specific locations established on the basis of analysis of flight patterns around the airfield; in particular, they are to be used in the mountainous areas.

Pilot state monitoring The system has to be able to assess the state of the pilot. For this purpose, it has to track her/his selected biological parameters during the flight, and compare them to the health-norms. For instance, system should be able to distinguish if the pilot is conscious, semi-conscious, or unconscious, and act accordingly. Measurements are to be performed by using appropriate bio-sensors,

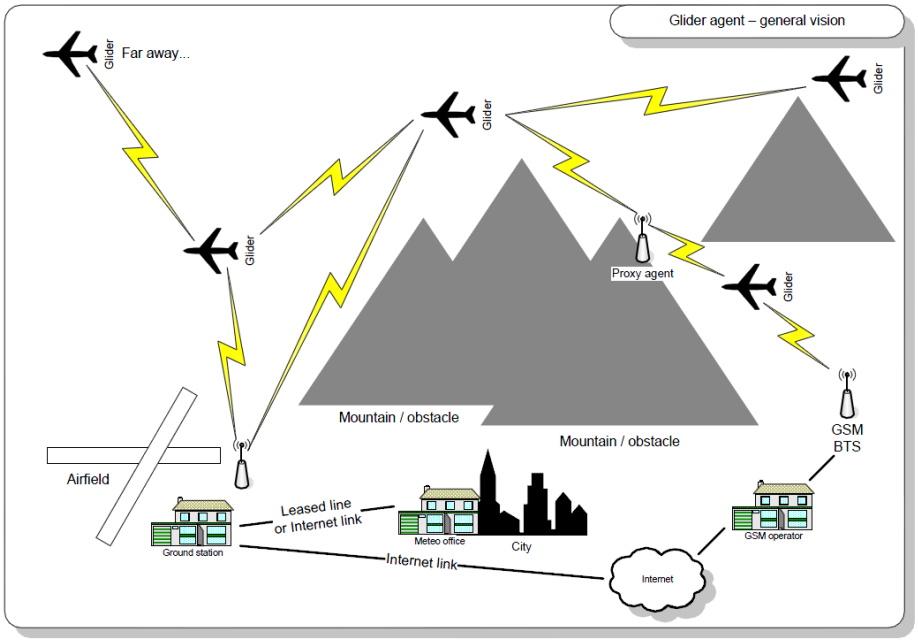

Figure 1. Overview of the GliderAgent system

in intervals that depend on the context (e.g. type of flight, current altitude, etc.) In selected contexts (e.g. perceived danger) results of monitoring should be autonomously communicated to the ground station.

Temperature and estimated life-time of battery One of the important factors influencing the ability of on-board electronics to function (and thus the GliderAgent system to run) is the state of the battery. The system should estimate how much energy is left in the battery, and thus its remaining life-time. It should also warn the pilot (and possibly the ground station) in the case when the battery is about to run out. For this purpose, measurements of the temperature of the battery (performed by appropriate sensors) can be used. The amount of available resources, and the type of flight, provide context for the energy management decisions (e.g. to shut down some functionalities, to retain the critical capabilities of the system, such as collision detection, tracking, etc.)

Flight state monitoring An important feature of the system is an ability to track gliders (from other gliders, as well as from the ground). To do this, the system has to keep track of each glider's flight state. This includes parameters such as position, speed vector, acceleration change (or, more specifically, total energy change), etc. This information should help avoid crashes, as well as to help supporting student training (by feeding the on-the-ground instructor with detailed information). Moreover, the current flight state should be clearly visible for the pilot to support decision making. Note that, here, the proposed functionality should extend and supplement capabilities of the existing glider-GPS-software, while using other data specified in this section.

Vertical stream indication Another vital aspect of glider flights, are vertical streams. Up-streams (lifts) help the glider to gain altitude, but down-streams (sinks) have the opposite effect and should be avoided. While the existing software, like the XCsoar, indicates such streams, the goal of the system would be to use glider-glider communication to share the information. Furthermore, since vertical streams often materialize in similar locations during similar weather conditions, access to the database of possible stream configurations would be useful.

Collision detection The GliderAgent should contain the collision detection functionality. It is expected to be based on exchanging proximity information between gliders. In the case of detected potential collision (i.e. two, or more, gliders are within a threshold distance from each other), the user and the ground station should be warned about the situation (collision warning should be displayed).

Maximal range The system should help the pilot to estimate the maximal range of a glider in the given conditions (with a reserve for the landing operation). This should be based on a range of parameters provided by a user, such as the glide ratio, height at which to start the landing operation, etc. It would help the user properly plan the flight, and properly react to changes in its conditions. This should also allow to issue a warning when the calculated maximum distance is shorter than the distance to the airfield.

Data logging Flight and pilot data should be logged and gathered for future analysis. Since the glider may have limited data-space available, depending on the context, selected information should be forwarded to be stored at the ground station. For instance, in the case of training flights, battery life is not as important as gathering "complete" data about the flight, whereas in the case of a long-distance achievement flight, only selected data should be stored/transmitted, since battery life is of key importance.

Team flying An auxiliary requirement for the system, is support of team flying. However, its completion, or lack thereof, does not directly influence the core functionalities described above. The user should have the possibility to create or to join a team of gliders. Creation should be based on specifying the name of the team. All users knowing this name should be allowed to join the team. Users from the same team should be marked on the system display.

The above requirements analysis is summarized in a use case diagram presented in Figure 2; for an extended discussion of its content, see [14].

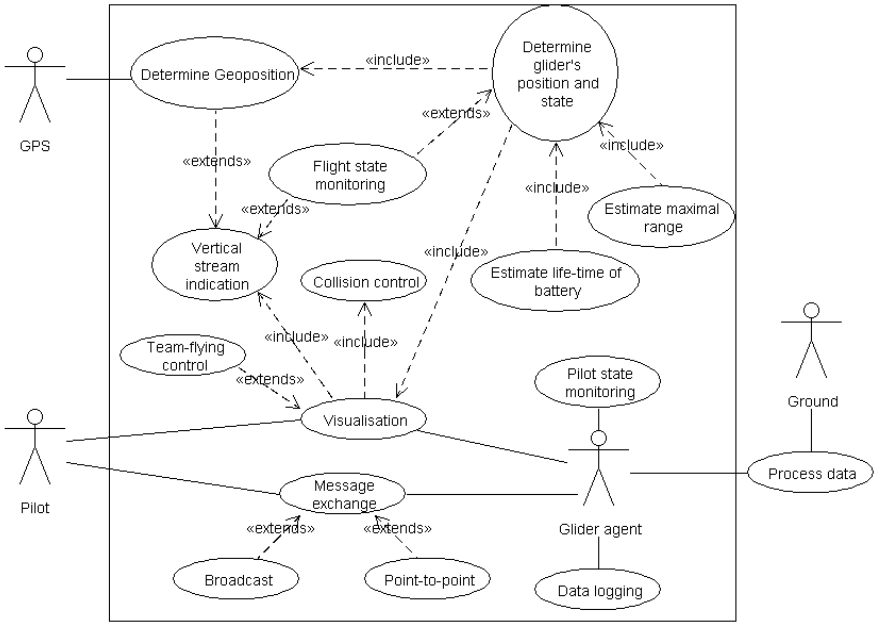

Figure 2. Glider agent system use case

1.2. Technical requirements analysis

Let us now describe the technical details of the GliderAgent system skeleton that we have decided to implement in the first stage of the project. This includes defining the software, the main classes of the system and their dependencies, the graphical user interfaces of various parts of the system, as well as implementation assumptions and limitations.

1.2.1. Utilized software

Let us start from specifying the software used in the project. The GliderAgent system consists of a number of agents that interact with each other to support functions described in the previous section. In addition, other non-agent software is used in the system. Here, we present the list of most important programs / environments used (we do not specify development tools, like the Eclipse, etc., as this information is of no substantial technical value).

- Software agents (except of agents working with sensors) have been implemented using Java Agent DEvelopment framework (JADE, v4.0.1); see, also Chapter by "JADE" G. Caire in this volume.
- MAPS (Mobile Agent Platform for Sun SPOT devices) is an agent oriented, Java-based framework for wireless sensor networks, which are based on the Sun SPOT technology; see, also Chapter "Wireless Sensor Networks and Software Agents" by G. Fortino and M. Essaaidi in this volume. Here, we assume that in our GliderAgent skeleton, MAPS agents and Sun SPOT sensors will represent "all possi-

ble sensing solutions." In other words, in a production environment they can be replaced by any technology with the needed functionalities.
Unfortunately, while both MAPS and JADE are Java-based, they use different communication methods. Specifically, while JADE sends messages according to the FIPA standards (using the ACL), MAPS creates its own messages based on events. Therefore, a gateway (JADE-MAPS/ Gateway) had to be implemented to facilitate message exchange between MAPS and JADE agents. However, this also means that in the case that a different sensing technology is to be used, all that will be needed is to develop another gateway between the sensor-originating messages and the JADE-based part of the system.

- Java Virtual Machine (Java SE 1.6 update 22) was used to run both JADE and MAPS agents. Note that, we made an assumption that there will be a possibility to run the Standard Edition of the Java Virtual Machine (JAVA SE) on the pilot-owned device. This assumption is a reasonable one, but is expected to be actually fulfilled in the next generation of smart devices (e.g. devices with the Android 3 operating system).
- Sun SPOT SDK [**?**] (Sun SPOT SDK 6.0, with the Solarium emulator included) is a set of libraries and tools provided by Sun Microsystems (currently Oracle), to develop applications for Sun SPOT devices. It contains an emulator called Solarium, which faciitates emulation of Sun SPOT devices for MAPS agents. The only limitation of this approach is that the use of Solarium disables mobility of MAPS agents, but this does not affect our project.
- The XCSoar [12] (version 6.0) is used and extended. Since this is an open-source software, it is possible to augment it with extra capabilities (e.g. to display various warnings). Note that, since the XCSoar is written in C++, while the JADE platform is Java-based, a bridge between the two had to be implemented to allow for data exchange.
- OpenStreetMaps [**?**] is a free map of the world. It consists primarily of road maps, contributed by users from around the world. It contains also some basic geographical data (e.g. forests, water, etc.), but doesn't contain more detailed information, such as the elevation. Swingx-ws's JXMapKit dynamically loads data from the OpenStreetMaps server. This means that it requires an Internet connection to function.
- Swingx-ws [**?**] (version 1.0) is a library of (open-source) add-on Swing features developed by the Swing Labs. It is primarily used to display a map including locations of gliders as seen by the ground station. The map is loaded from the OpenStreetMaps.

1.2.2. System structure

The system structure is depicted in the component diagram in Figure 3. It should be viewed together with the use case diagram (in Figure 2).

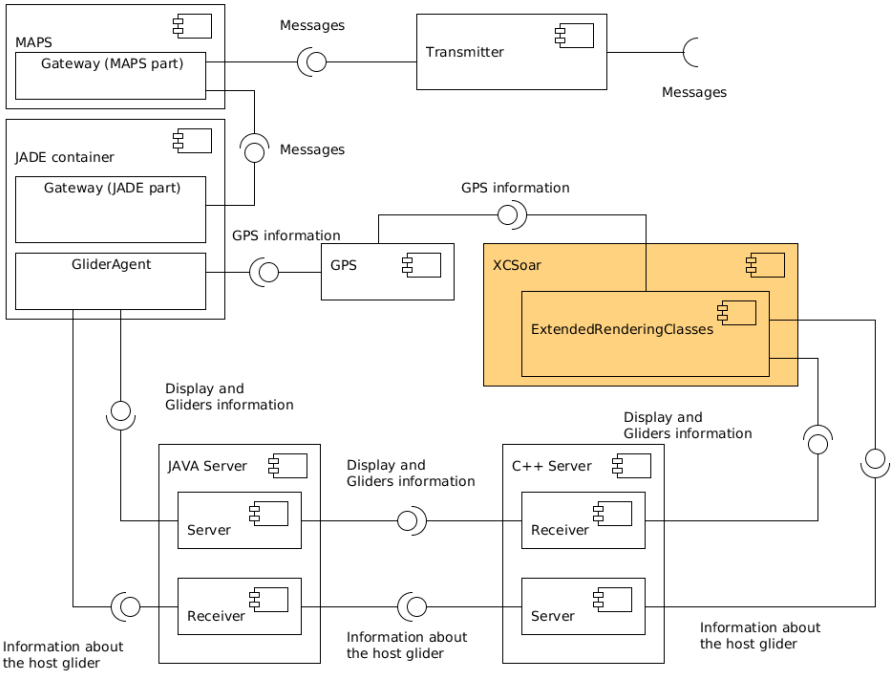

Figure 3. Glider agent component diagram

The component diagram shows components of a *Glider Agent* (*GA*), software agent residing on a single glider. The correctness of the decision making process of the *GA* depends on information obtained from a set of MAPS agents via a radio transmitter and the XCSoar, or directly from the GPS antenna. The radio transmitter connected to the *GA* allows it to exchange information with other JADE and MAPS agents. Due to this, the *GA* is able to use the radio transmitter (via a gateway) and obtain information from other gliders (e.g. position, messages, air stream data) and from its own set of sensors. As mentioned above, since JADE and MAPS use different communication protocols (FIPA and Events), the JADE/MAPS gateway *(J/M)G* is needed to translate messages between these two agent platforms.

Since the *GA* is written in JAVA, while the XCSoar is written in C++, a common communication interface is required. Therefore, a *JAVA server* and a *C++ server* are introduced. Both servers are based on a local socket connection. The XCSoar program gathers and displays messages generated in the decision making process of the *GA* and the data concerning nearby air traffic, which is obtained through communication within the system (i.e. messages containing positions of other gliders). The XCSoar uses our extended rendering classes to fulfill the display requests. In return, it sends information about the glider to the *GA* (GPS position, etc.) In case of any failure in the communication with the XCSoar, there is a special back-up connection from the *GA* to the GPS. It allows the *GA* to determine the error and, if possible, provide critical functionality such as collision detection, communication within the system, etc.

Due to the lack of space, we present only the core information regarding the system. Therefore, for the ground station and the proxy station component diagrams, refer to [**?**]

1.2.3. Graphical user interface

Let us now outline the key information abut the GUIs implemented in the GliderAgent system.

- **Glider agent** does not have its own GUI. However, the modified XCSoar display is used to display important information to the pilot. First, the standard XCsoar information is displayed, such as:

  * glider position and state
  * nearby area (using map)
  * vertical streams

  However, the display is extended to show the position of others gliders and messages concerning current emergencies and warnings, as well as other messages from the *GA*.
  Ground Station. The graphical user interface of the ground station is meant to allow the ground staff to monitor the current flight data concerning gliders in the air. For this purpose, the GUI provides the user with a list of Glider Agents visible to the given Ground Station Agent. The GUI provides a map of the area with gliders highlighted on it. The map is implemented via a 3rd party API (Swingx-ws). The ground station GUI delivers detailed information concerning a selected GliderAgent's data. This data includes:

  * Geographical coordinates (latitude, longitude)
  * Altitude
  * Velocity
  * Heading
  * Bank
  * Pitch
  * Battery voltage and temperature
  * Pilot temperature
  * Oxygen pressure
  * Warning messages

  Note that, in the GliderAgent system, we assume that communication between the pilot and the instructor is also conducted using medium other than the agent software (for example, the classical voice-over-radio transmission).
  Proxy Station. The proxy station serves only to relay messages between Agents, so it does not require any special user interface beyond the default JADE display of visible agents.

1.2.4. Initial GliderAgent skeleton development—assumptions and limitations

The GliderAgent system is meant for live usage (we plan to deploy a demonstrator in real-life settings of the Jeseník Aero Club (CZ), or the Nowy Targ Aero Club (PL); both Aero Clubs located close to Carpathian Mountains). But during the development, it was necessary to utilize a special emulator environment allowing us to check basic function-

alities. Here, we had to make a number of assumptions concerning the development of such emulator. It should be noted that some of the assumed technologies (listed next) are in early stages of development themselves, so some of our assumptions are based on observed trends related to the technological development of PDAs and transmission technologies. Let us also specify, which parts of the skeleton were selected to be developed at this stage of the project. Finally, let us list certain arbitrary criteria set for testing purposes.

Radio technology with a minimum 20-40 km range. It is assumed that long range radio technology with 20-40 km range is installed on every glider. It is not known if such technology is currently available. However we assume that we will be able to acquire such technology in the future. For example, the XBee-PRO XSC or the XTend RF technologies are very promising one (see [**?**]). This will allow to fully utilize functionality of the GliderAgent system.

PDA/Netbook. It is assumed that the PDA / smartphone device is represented by one personal computer connected to the Internet. The device should be capable of properly running all software used in the system.

Sensors/Solarium. Since it is unclear what sensing technology will be used in the actual deployment of the system, we have decided to emulate sensing using the Solarium emulator (provided with the Sun SPOT SDK), which allows us to run MAPS agents.

Communication between system elements. We assume that each element has to setup its unique, global ID such that it can be easily detected, and connected to other elements.

Average pilot health. Assumption concerning average pilot health state have been made on the basis of the literature (see [**?**]). These basic assumptions allow us to determine when there is something wrong with the pilot.

Exemplary glider SZD-50-3 Puchacz. Due to a variety glider models, it is assumed that characteristics of a Polish glider SZD-50-3 known as Puchacz will be used [**?**]. Note that this assumption does not impact results of our work, but it allows us to specify some realistic parameters and thus make tests of the system more reliable.

Maximal glider range assumptions. When estimating the maximal range of a glider in current situation we need to take various factors into consideration. The user of a *GA* will have to provide some of them, while the remaining ones will be calculated from the user input.

Assumed atmosphere and environment model. The International Standard Atmosphere (ISA) model will be used. It is characterized by two important features: (1) below *36 000* feet, the standard temperature lapse rate is 2°C (3.5°F) per *1 000* feet of altitude change; and (2) pressure does not decrease linearly with altitude, but for the first *10 000* feet, 1 in.Hg. for each *1 000* feet approximates the rate of pressure change.

2. System implementation

Let us now describe in some detail various issues concerning implementation of the GliderAgent system skeleton. Let us start from the details of the emulation environment.

2.1. Emulation details

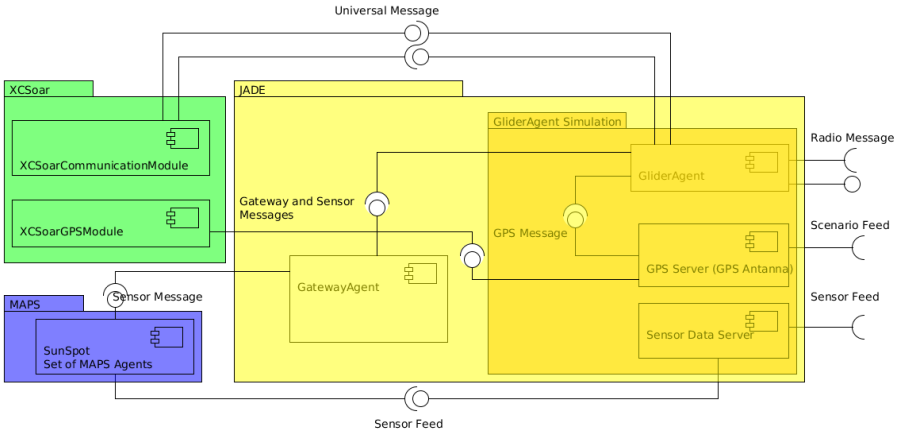

Figure 4. Simulation component diagram

In Figure 4, we can see the component diagram of the Glider Agent implemented for our simulation. The GliderAgent System Simulation, which is run on the JADE agent platform, is composed of the Glider Agent *GA*, the *GPS server* and *the sensor data server*. The *GA* depends on radio messages from the *the radio server* (which is a part of the GliderEmulator presented in chapter 2.1.2), information from the XCSoar, and the GPS feeds from the GPS antenna (in the simulation represented by the *GPS server*). The *GA* component contains the code of the JADE agent. Radio messages are sent and received via a socket connected to the *radio server*, opened by the GliderEmulator.

The *GPS server* and the *sensor data server* depend on the data received from the *scenario server* (another part of the GliderEmulator application). They provide a special interface to resend these feeds among its clients. The Glider Agent, is always the first client of these servers, and thus it always gets the data. In this particular case, the *GPS server* provides the GPS feeds to the *GA* (its owner) and to the XCSoar (a registered external client).

In the same manner, the *sensor data server* sends the data to all MAPS agents. However, for the simulation purposes, we do not handle the sensor data in the *GA*. MAPS agents receive the sensor data about the voltage of the battery, battery temperature, pilot temperature (sensor placed near legs of the pilot), blood saturation and oxygen pressure. Each MAPS agent is responsible for only one type of message (one MAPS agents per sensor approach, which may be modified in the future).

The gateway agent provides an interface to enable communication between JADE and MAPS platforms. The gateway agent depends on Requests sent by the *GA* and Events sent by the MAPS agents (for more details, see Section 2.1.2). The XCSoar components depend on the GPS feeds (provided by the *GPS server*) and information provided by connected agents in UniversalMessage(s). The XCSoar application is a “glider computer,” which visualizes other gliders and messages on the screen. In reply to the *GA*, the XC-

Soar sends detailed information obtained through its internal data processing (i.e. bank and pitch angle data).

2.1.1. Ground agent and proxy agent emulation details

The ground agent connects to the emulator via two sockets—one for the *scenario server*, and one for the *radio server*. The *radio socket* uses the host and port specified, while the *scenario socket* uses the same host, but with the *port + 1*. The *scenario server* receives and stores weather data as a replacement for an actual source of weather data. The *radio server* is used to simulate sending and receiving messages to and from other JADE agents within the simulated radio range.

The proxy agent works similarly, except that it only makes use of the *radio server* and, as intended, it only forwards massages to other agents within its radio range. To overcome problems concerning "infinite" forwarding messages between agents, we introduced in each message a maximal number of hops (initially equal to 2). The proxy agent briefly looks at the message's number of hops and decreases it by 1 before sending. If the proxy agent receives a message with a number of hop equal to 0, then the message is discarded. Obviously, this parameter may be changed if needed.

2.1.2. Environment emulation (GliderEmulator)

Both aforementioned components rely on a feed from the environment simulator—the GliderEmulator *GE*. The *GE* is a GUI application written in Java allowing users to directly prepare the route of the glider(s) and set the crucial points (scenario waypoints) which describe the state of glider in these points and in between.

The application is composed of a main class and two servers:

- *Radio server*—emulating the radio communication (range of communication).
- *Scenario server*—providing GPS routes, sensor data for glider and weather data for ground stations.

The *GE* uses OpenStreetMaps allowing user to have a fairly accurate view of the environment in which s/he plans to emulate the GliderAgent system operation. To calculate distances, the *GE* uses an implementations of Tadeusz Szpila's "Vincenty's formulas" [**?**], which are a two related iterative methods used to calculate distance between two points on spheroid, and finds the destination point knowing the starting point, distance and bearing. Also, the GE provides a set of GPS message constructors, simulating a real feed from Garmin-type GPS receivers.

2.2. JADE-MAPS Gateway

The JADE-MAPS gateway (or simply the gateway) is a software agent, which allows exchanging messages between JADE and MAPS agents. In Figure 5 we present its Use Case diagram.

The gateway contains the gateway agent which is a JADE agent. Without loss of generality, in the diagram we represent only a single other JADE agent (which represents all JADE agents that may need to communicate with the MAPS agents). This external agent performs actions, which affect the gateway's behavior. Precisely, it can register or deregister itself at the gateway, ask for a list of MAPS agents, and request sending of a message to a specified MAPS agent. It is assumed that only registered agents can send

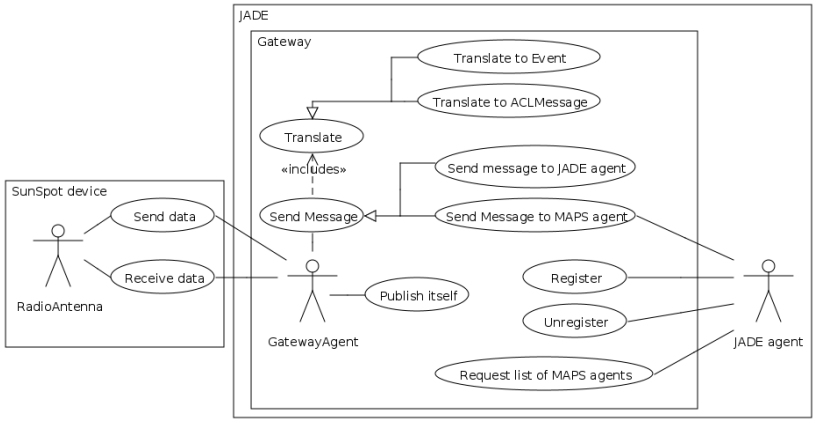

Figure 5. JADE-MAPS Gateway use case diagram

and receive messages via the gateway. Furthermore, the gateway can publish itself in the MAPS wireless sensor network, and in the JADE-based system (using the DF agent, which manages descriptions of services). As a result, the gateway becomes visible for both JADE and MAPS agents.

The main function of the gateway is the translation of messages. Here, we have to consider two translations: translation from messages used within JADE to events used within MAPS, and vice-versa.

The radio antenna is a part of the Sun SPOT system, which allows sending and receiving data via the wireless sensor network of Sun SPOT devices. The gateway uses this radio antenna to communicate with MAPS agents.

2.2.1. Implementation details

The *GatewayAgent* is the main class of the gateway project, which extends the *Agent* class from the JADE library. In Figure 6, we can see the relation between the *GatewayAgent* class and other classes responsible for agent management and communication within both JADE and MAPS platforms.

The gateway has to utilize both JADE and MAPS source code. Therefore the gateway is a JADE agent which contains also MAPS code, responsible for the radio communication within the MAPS. Specifically, the gateway uses directly the *MobileExecutionEngine* class from the MAPS. Since the mobile execution engines are responsible for routing messages, running and maintaining MAPS agents on the Sun SPOT devices, writing the gateway in such a way helps to achieve a better integration of both agent platforms. The gateway becomes visible for the MAPS side as the mobile execution engine, while remaining visible within the JADE platform (as a JADE agent). In this way, JADE agents can be seen as the MAPS agents to “real” MAPS agents.

Since we develop the gateway agent as the MAPS execution engine, the important parameter describing the gateway is its IEEE address. The IEEE address is directly taken from the Sun SPOT SDK (the Sun SPOT SDK provides support for host applications that have access to a connected Sun SPOT Base Station consisting of a processor and a

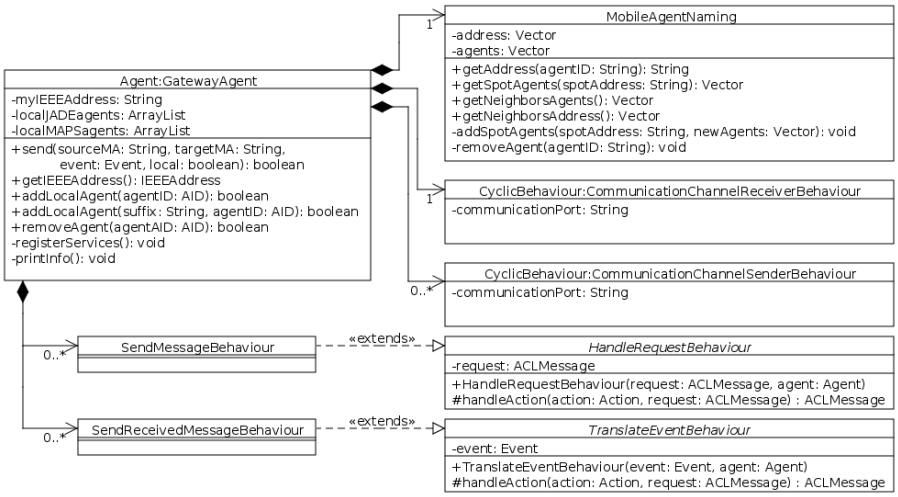

Figure 6. JADE-MAPS gateway class diagram

radio antenna). This address is unique and it is precisely this address that allows other execution engines on the Sun SPOT devices to communicate with our gateway agent.

The *GatewayAgent* class also contains two lists, the local JADE agents *ArrayList* and the local MAPS agents *ArrayList*. There is a bijection between JADE agent ID's stored in the *localJADEagents* and their MAPS agent ID's stored in the *localMAPSagents*. During the registration of a JADE agent at the gateway, a new MAPS ID is created, published among other execution engines and assigned to this agent. MAPS ID's of JADE agents are used in addressing messages within the wireless sensor network. However, an ID is not usable by its owner, only the gateway agent can refer to it during the translation process. For user-friendliness, there is a possibility to specify a suffix, which may be used in the process of generation of MAPS IDs. If a suffix is not specified, then it becomes a randomly generated number.

Since the gateway is also a JADE agent, it has to obey the one thread per agent policy. To satisfy this requirement, to run MAPS within a JADE agent, most of MAPS classes were rewritten as agent behaviours, which are run one by one within the agent and perform quick, simple tasks. Thanks to this, the gateway is written as a single-threaded agent with only a few unavoidable exceptions (for example, receiving data from the radio connection). Here, the JADE threaded behaviour was utilized. A threaded behaviour runs as a normal JADE agent behaviour, but in a separate thread and does not block an agent by waiting, or performing complex operations.

As presented in the class diagram (Figure 6), the gateway agent owns an instance of the *MobileAgentNaming* class. The *MobileAgentNaming* was taken from the MAPS' source code, and only slightly modified. The main purpose of this class is to store information about remote execution engines and their agents. This class allows to acquire the required address(es) based, for example, on the name of a MAPS agent, and save current data based on radio feed from another execution engine. It is worth noting that

MAPS agents do not have their own IEEE address. The IEEE address of a MAPS agent is always the IEEE address of the execution engine in which this agent is present.

Two other important classes are: *CommunicationChannelReceiverBehaviour*, and *CommunicationChannelSenderBehaviour*. These two behaviours are responsible for accessing the radio communication channel provided by the radio antenna. In Figure 6, we can see that the gateway agent has only one receiver behaviour, and may have zero or many sender behaviours. The *CommunicationChannelSenderBehaviour* is run within a separate thread as a threaded behaviour. Otherwise, the gateway wouldn't be able to check availability of data in the communication channel and would block the execution of other behaviours, which could be performed during this time. Data received by the communication channel receiver behaviour is translated into an event and forwarded to the execution engine (in this case the gateway), which routes it to the proper local agent, or to another execution engine. If an event is addressed to a local MAPS agent then the translation mechanism is initialized and the message is translated to an *ACLMessage*. This *ACLMessage* is sent by the JADE to the proper, registered agent. The *CommunicationChannelSenderBehaviour* extends the *OneShotBehaviour* class. This behaviour is registered for a different event, each time the gateway would like to send some data. There can be many such behaviours registered concurrently. Each of them is run one by one (due to the one thread per agent requirement).

The *SendMessageBehaviour* and the *SendReceivedMessageBehaviour* are one-shot behaviours, which are part of the translation mechanism within the gateway. Similarly to the communication channel sender behaviour, there can be zero or many such behaviours, registered for different messages. With respect to the purpose of the message, a proper class extends one of the following classes: *HandleRequestBehaviour* and *TranslateEventBehaviour*. It is worth to remember that a JADE agent requests are send as *ACLMessages*, hence a translation from an *ACLMessage* to an *Event* is a part of the *HandleRequestBehaviour*. Proceeding in the opposite direction, translation from an *Event* to an *ACLMessage* has to be handled automatically when a message contains a local MAPS agent ID (connected to a JADE agent); since MAPS agents cannot specify an addressee of the request directly.

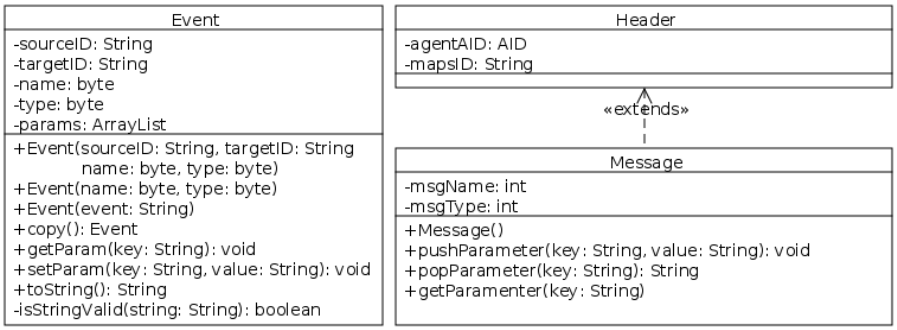

Figure 7. Event and Message class diagram

In Figure 7, we can see the Class diagram of the *Event* and *Message* classes. These two classes represent the basic communication formats used in MAPS and JADE re-

spectively. Despite the differences in the structures of these classes, they are based on a very similar idea of sending data. Each of them has fields responsible for describing the sender, receiver, message name and type. The message name and type are used only within the MAPS sensor network and describe the topic of the message (for example, current temperature, exceeded threshold, etc.), and the time (now, first occurrence, permanent). By default, in JADE, these fields are set to the topic called MSG (message) and the type NOW. The content of a message is stored as a sequence of keys and values, which may be represented as follows: key1->value1->key2->value2-> .... The receiver of a message may get a value by providing a specified key, or by adding a new one.

Due to the simplicity of the radio communication channel, the *Event* is translated into a *String* and sent over the air. The main purpose of the gateway is to translate simple *Strings* (*Events*) to more complex structures (*Messages*) (and vice-versa). Thanks to the similarities described above, the basic translation is performed almost immediately by copying data. The gateway takes care of filling all fields correctly and changing the *sourceID* or the *targetID* into a proper one.

3. Test scenarios and experimental validation

The *GA* is expected to support the pilot in different flight situations. The main functionality of the agent is to track the flight state, which includes monitoring air traffic in the nearby area, monitoring critical elements within the glider (i.e. the state of the battery), as well as the state of the pilot (e.g. the oxygen level in the blood). To illustrate some of the already implemented functionalities, and to test basic features of the system, we have selected two scenarios.

3.1. Proximity and battery warning generation

The first scenario illustrates the ability of the *GA* to notify the pilot about potential collision (proximity warning), and about a drop of battery power (critical battery level warning).

This scenario requires one GliderEmulator, two glider agents (agents A and B), and one ground agent. At the beginning, glider A is on the ground, while glider B is in the air. When the scenario starts, both gliders begin to change their positions. Because it is assumed that the radio range of a glider is equal to approximately 40 km, they directly communicate with each other and with the ground agent (representing the ground station) by sending messages containing their current positions. Based on this information, each *GA* and the ground agent display the nearby air traffic.

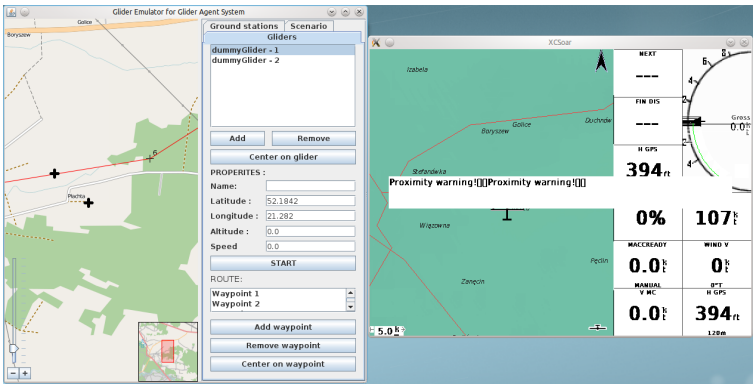

Figure 8. GliderEmulator GUI and XCSoar display of glider A - Proximity warning

At some point in time, gliders A and B start to be too close to each other. This dangerous situation is presented in figure 8, which contains a window of the the GliderEmulator and a display of the XCSoar program used by one of the glider agents (agent A). Both gliders, based on obtained information about position of nearby gliders, warn their pilots by displaying a "Proximity warning." In the scenario, gliders avoid collision and the warnings disappears as soon as they reach a safe distance.

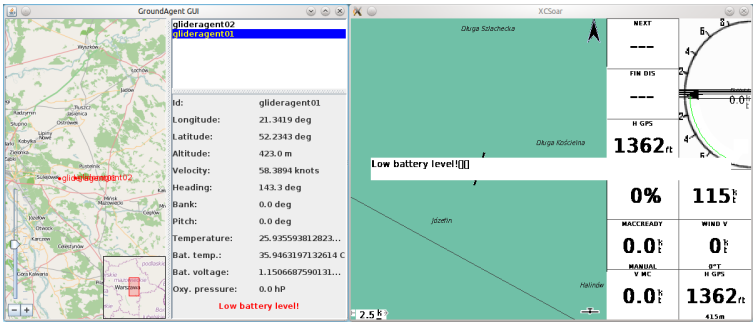

Figure 9. Ground agent GUI and XCSoar display of glider A - Low battery level warning

After a while, the glider agent A receives (from the MAP agent assigned to sense the battery power) information concerning low voltage level in the battery. In the next few seconds, the voltage level reaches a critical state and the remaining battery power is equal to 0%. The *GA* warns the pilot of the glider by displaying the "Critical battery level" message. Figure 9 illustrates this situation. On the left hand side, we can see the display of the XCSoar program with the notification concerning the state of the battery. The window on the right hand side shows the GroundStation GUI which is also made aware (via a message sent an autonomously by the *GA* of the glider A) of the incident on the glider A, caused by voltage level equal to 0%. Since the assumed reason for the sudden drop in battery level was cabling problem, it is assumed that pilot of glider A managed to reconnect the battery and made the warning disappeared. Completing the scenario, glider A lands back on its airfield.

3.2. *Oxygen level monitoring and MAPS-JADE communication*

The second scenario concerns ability of the system to monitor the state of the pilot and alert her/him about dangerous events which may influence the safety of the flight. Here, we assume that the *GA* monitors the selected biological parameters of the pilot, by communicating with a set of MAPS agents. The following scenario tests the ability of the *GA* to notify the pilot about possible danger caused by low oxygen level, and communication between MAPS-JADE agents.

In this scenario, we consider two types of warnings: (1) low oxygen level generated at 9842.52 ft (3000 m above the sea level), and (2) critical oxygen level generated at 13123.36 ft (4000 m above the sea level).

At the beginning of the scenario, the glider stays on the ground at the altitude of 0 m (above ground level). The position of the glider and its altitude start to change when the scenario is executed. It is assumed that the glider is conducting lee-wave flying, and its altitude is increasing fast.

Due to limited amount of resources (in this case the battery), MAPS agents send their information to the *GA* within specified time intervals. In case of the oxygen level, this time interval is initially set to 30 seconds. However, MAPS agents communicate among themselves, and with the *GA*, and based on the received information they modify the interval in order to adapt to the situation. Due to this, at the altitude 2000 m, the oxygen MAPS agent modifies the measuring interval from 30 s to 10 s. Similar changes can be observed after every next 500 m above this level, until the interval is set to 1 s at the altitude 4000 m (considered the critical level).

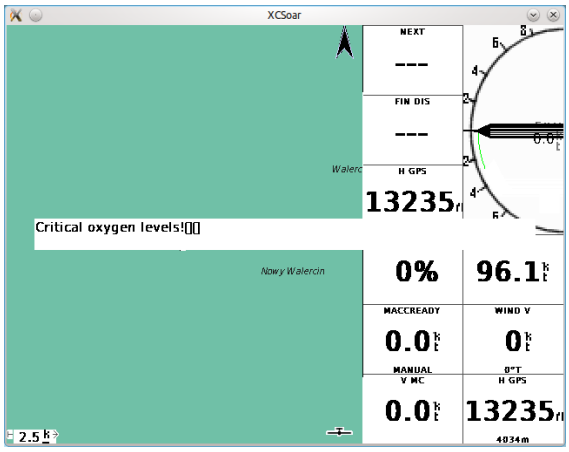

Figure 10. XCSoar display showing "Critical oxygen level" warning

In figure 10, we can see the output window from the running sensors and MAPS agents, and the display of the XCSoar program. The oxygen MAPS agent acts as expected. It modifies its measuring interval from 30 to 10 seconds as soon as it gets an information from the *GA* that the altitude of the glider exceeds 2000 m (6561.68 ft).

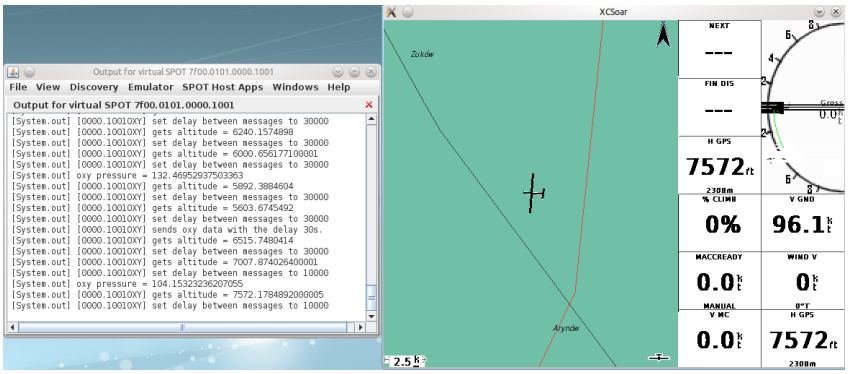

Figure 11. Output window of virtual Sun SPOT and XCSoar display of glider

Figure 11 presents the situation when the glider reaches the altitude of 4000 m. We can see the display of the XCSoar program which contains the altitude greater than 13123.36 ft, and the warning "Critical oxygen level."

Therefore, on the basis of both scenarios, we can say that MAPS and JADE agents efficiently communicate between each other. Furthermore, the *GA*, based on its position and information received from sensors, is able to warn the pilot of a glider about a dangerous situation by displaying a proper message on the display of the XCSoar program.

4. Concluding remarks

In this chapter we have described issues concerning development of the initial skeleton of an agent system used in decision support of glider pilots. First, we have provided and overview of glider flying and summarized reasons for developing a glider pilot support system. Next, we have presented the requirements analysis for the GliderAgent system and discussed main issues concerning its implementation. Particular attention was paid to the development of the JADE-MAPS gateway, which allows two agent systems, based on somewhat different design philosophy to interact with each other. Finally, two test scenarios were used to illustrate the implemented functionalities. As we move along, the next step of the project will be to add decision making capabilities to the GliderAgent, by infusing it with flight-context recognition, to be used in on-board resource management.

References

[1] Fédération Aéronautique Internationale (FAI).
[2] Altitude sickness. Wikipedia.
[3] Chile grand prix success. Rock Seven Mobile Services Ltd.
[4] FLARM. Wikipedia.
[5] GNSS recording devices. Fédération Aéronautique Internationale (FAI).
[6] LX navigation. C. Aero.
[7] OpenTracker+. A. D. Systems.
[8] TinyTrak3. Byonics, LLC.
[9] Traffic Collision Avoidance System (TCAS). Wikipedia.

[10] vPos. vPos as, Norway.
[11] WinPilot. Sierra SkyWare, Inc.
[12] XCSoar. XCSoar team.
[13] Yellowbrick specifications. Rock Seven Mobile Services Ltd.
[14] A. Gab, P. Andreou, M. Ganzha, and M. Paprzycki. GliderAgent—a proposal for an agent-based glider pilot support system. *Methods and Models in Automation and Robotics (MMAR), 2010 15th International Conference on*, pages 55 – 60, 2010.