instituto de telecomunicações

Relatório Final (Bolsa) *Final Report (Grant)*				
Referência da Bolsa (*Grant Reference*)	BII Nº2022/00110			
Tipo de Bolsa *(Grant Type)*	Bolsa de Iniciação à Investigação			
Entidade *(Entity)*	Instituto de Telecomunicações			
Polo/Delegação *(Entity's site)*	Aveiro			
Nome Completo do Bolseiro *(Grantee's Full Name)*	David José Araújo Ferreira			
Orientador *(Supervisor)*	António Navarro Rodrigues			
Projeto de Investigação *(Research Project)*	FireTec			
Referência do Projeto *(Project Reference)*	UIDB/50008/2020-UIDP/50008/2020			
Período da Bolsa *(Period of the Grant)*	**De** *(From)*	01/01/2023	**Até** *(Until)*	30/06/2023

Developed work

During my participation in the FireTec project, my main task focused on developing the necessary software to expand the communication capabilities of the existing infrastructure, by imbuing the server-side components with the capability of generating original CAP messages and the FireTec Switch with the capability of reading them.

Common Alerting Protocol

My work began with studying and understanding what Common Alerting Protocol (CAP) is, how and when it can be used, and how it is ideal for meeting this project's communication necessities.

The Common Alerting Protocol (CAP) is an open standard for exchanging emergency alert and public warning messages between different systems and platforms, developed by the OASIS organization. It provides a standardized format and structure for the transmission of alerts, ensuring interoperability and compatibility across various emergency management systems. CAP allows emergency management agencies, alerting authorities, and other organizations to create, share, and distribute alerts in a consistent and timely manner.

At its core, CAP defines a comprehensive data model and an XML-based message format for conveying emergency alerts and warnings. The protocol supports the transmission of various types of information, including event descriptions, locations, severity levels, recommended actions, contact information, and multimedia attachments such as images or audio files. CAP messages can be categorized by the type of event, such as natural disasters (e.g., hurricanes, earthquakes), public safety incidents (e.g., terrorist attacks, chemical spills), or other emergencies.

CAP messages follow a structured format, allowing systems to process and interpret them automatically. This enables alerting authorities to disseminate alerts through multiple channels simultaneously, including emergency alert systems, mobile applications, social media platforms, email, and more. CAP is designed to enhance the speed and effectiveness of emergency communication, ensuring that critical information reaches the right people and organizations promptly, regardless of the specific technology or platform they use.

Overall, CAP plays a crucial role in modern emergency management and public warning systems by facilitating the exchange of standardized, machine-readable alerts. Its adoption promotes interoperability, consistency, and efficiency in alerting processes, ultimately improving emergency preparedness and response capabilities at local, regional, national, and international levels.

CAP Messages creation

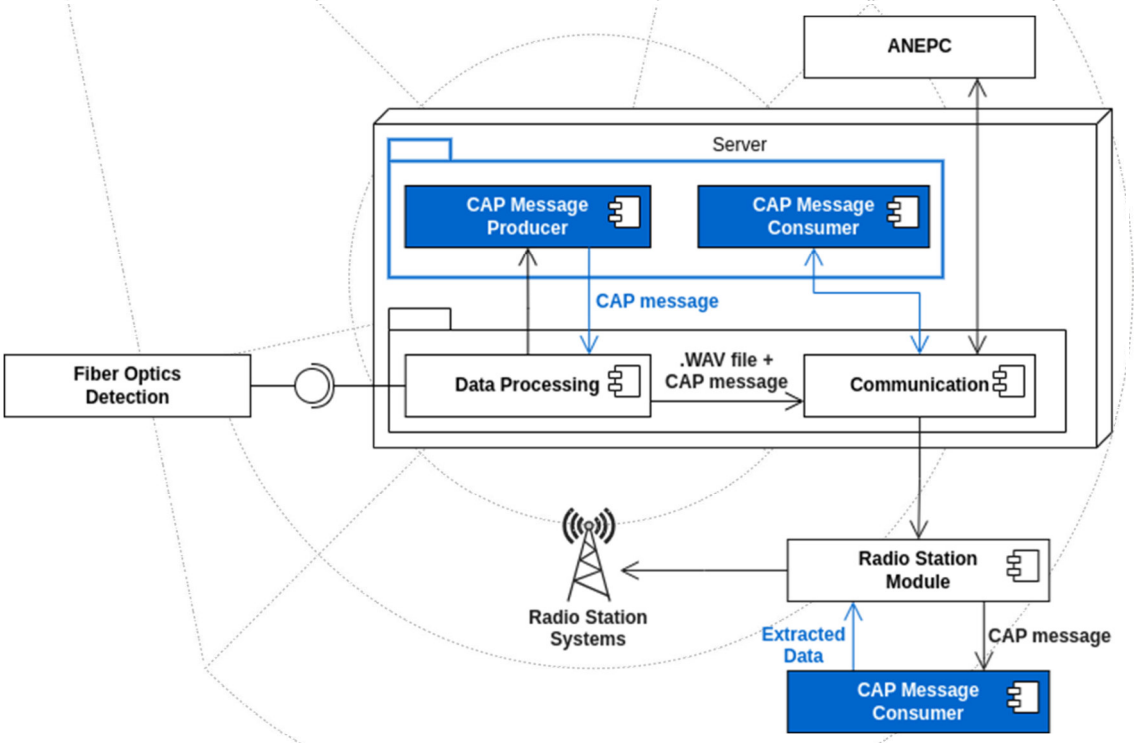

Figure 1 - FireTec overall architecture, with CAP-handling components highlighted in blue

As illustrated in Figure 1, it was necessary to create a component that is able to read, modify and create CAP messages, and interact with the server-side processes, as well as a component capable of reading CAP messages and running in the Arduino running on the radio station module (*FireTec Switch*).

To accomplish this, Python language was used and a new package, *capparser*, was created. Using the *capparser* package, users can access the content of CAP messages through the provided functions and methods. They gain the ability to extract crucial information such as event descriptions, locations, severity levels, recommended actions, contact details, and multimedia attachments. This comprehensive analysis and processing of CAP messages enable users to derive insights or take specific actions based on the received alerts.

Additionally, the *capparser* package supports exporting CAP messages to binary format or saving them to a file. This feature proves useful for archiving or transmitting CAP messages to other systems or users. Exporting to binary format ensures efficient transmission and storage while saving to a file enables easy sharing and integration with external tools or systems.

Regarding the content specific to the *FireTec* project, it was within this project's scope to transmit audio messages from the server to the *FireTec Switch,* and thus, audio content must be included in CAP messages. To include audio within a CAP message, an alternative approach was adopted in order to minimize the message size and avoid unnecessary padding associated with *base64* encoding, which is typically used for this type of necessity. When using *base64* encoding, padding is added to ensure that

the encoded data length is a multiple of 4, which can introduce additional characters and increase the message size.

Instead, the generated audio file was read by the server in binary mode to access the raw byte data directly. Each byte of the audio data was then converted into its corresponding hexadecimal representation as a string. This conversion allows the binary audio data to be represented in a compact textual format without padding.

By representing the audio data as hexadecimal strings, the CAP message remains concise, and the unnecessary padding introduced by *base64* encoding is avoided. Each pair of hexadecimal characters in the resulting string represents one byte of audio data.

To include this audio data in the CAP message, each byte is converted to its hexadecimal representation as a string: "*E02A671F…"*

The audio data can be reconstructed by parsing the hexadecimal string back into its original binary form when needed, ensuring accurate utilization without the need for additional padding.

In addition to the audio content, information relating to the audio transmission via radio is also transmitted within the CAP message. This includes the RSD commands PS, PI, and AF, these will be included in *parameter* fields.

Message parsing by a *FireTec Switch*

The server utilizing the *capparse* package establishes communication with a target system, an Arduino microcontroller. However, due to the Arduino's limited memory, it cannot store the entire CAP)message in a single String variable as it exceeds the buffer size. To address this constraint, an approach called "*in-reception parsing*" has been implemented, allowing the Arduino to parse and process the CAP message while it is being received.

"*In-reception parsing*" allows the Arduino to parse and process the CAP message while it is being received, enabling real-time analysis and handling of the incoming data. Instead of waiting for the entire message to arrive, the Arduino progressively parses and extracts relevant information from the XML structure as each portion of the message is received. This approach ensures that memory limitations are not exceeded and allows the Arduino to efficiently process the CAP message without waiting for its complete arrival.

During the "*in-reception parsing*" process, a crucial step involves detecting the segment of audio data within the CAP message. As each byte of the audio segment is identified, it is immediately converted into its corresponding hexadecimal representation and written directly to a file. This approach ensures that the audio content is saved in real time as it is being received.

The Arduino microcontroller effectively manages its limited memory resources by writing the converted bytes directly to a file during the reception process. This strategy prevents the need to store the entire audio segment in memory before writing it to a file, which could exceed the Arduino's memory capacity.

Overall appreciation of the developed work

Overall, the development work accomplished in creating the *capparse* package and implementing "*in-reception parsing*" on the Arduino microcontroller has proven to be a successful endeavor. The objectives set for the project have been achieved, demonstrating the effectiveness and functionality of the implemented solution.

The *capparse* package's comprehensive capabilities, including reading, modifying, and creating CAP messages, coupled with its ability to export data in various formats, have greatly improved the handling of emergency alerts and information exchange. By incorporating "*in-reception parsing*" on the Arduino microcontroller, memory limitations have been addressed, allowing for the real-time processing of CAP messages. The selective filtering and processing of specific fields, such as the *parameter* field and the audio hexadecimal string representation, have enabled efficient memory usage and timely actions.

Overall, the successful completion of the project signifies a notable accomplishment. The diligent efforts put into developing the *capparse* package and implementing "*in-reception parsing*" on the Arduino microcontroller have resulted in an effective and practical solution for managing CAP messages.

Provided code

For this project, the developed code was aggregated into a directory with the following content, organized in directories.

- *arduino-complete-code/*
  - Contains all the code that can be compiled and loaded into the Arduino in order to run the complete FireTec application.
- *arduino-socket-test/*
  - Contains code that can be compiled and loaded into the Arduino, only creating a socket connection and implementing the CAP parsing algorithm. This is a skeletal version of the complete code, used for the development and testing of the CAP communication. This would simulate the client, *FireTec Switch*, side of the communication operation.
- *audio-sender-mock/*
  - Contains a simple Python script that imports the *capparser* package and generates a new and simple CAP message. This was used in combination with the previously described code in order to develop and test the CAP communication. This would simulate the server side of the communication operation.
- *fire-tech-switch-ddns-look-up/*
  - Contains code that can be compiled and loaded into the Arduino, and implements DNS look-up.
- *references/*
  - Contains all the gathered documentation used for the initial study regarding the CAP protocol implementations, uses cases for CAP in large-scale operations, and protocol specificities.
- *scholarship-docs/*
  - Documentation created for this scholarship, such as the raw vectors for the illustrative diagram previously cited and the work plan initially proposed.
- *XMLs/*

    - Contains example CAP messages developed by the OASIS organization, as well as a provided CAP message example, created by PROCIV.
- *conformance-reference.xsd*
    - This conformance file, also made available by the OASIS organization, is used to verify the conformance of CAP messages against the required protocol specifications.

Data *(Date)*	
Assinatura do Bolseiro *(Grantee's Signature)*	
Assinatura do Orientador *(Supervisor's Signature)*