CAMEO Software Architecture and APIs: Technical Specification

Valerio Arnaboldi, Marco Conti, Franca Delmastro

IIT Institute, National Research Council of Italy
via G. Moruzzi, 1 - 56124 Pisa, Italy
email: firstname.lastname@iit.cnr.it

Abstract

Mobile systems are characterized by several dynamic components like users' mobility, devices' interoperability and interactions among users and their devices. In this scenario context-awareness and the emerging concept of social-awareness become a fundamental requirement to develop optimized systems and applications. CAMEO is a light-weight context-aware middleware platform for mobile devices designed to support the development of real-time Mobile Social Networks (MSN) applications. MSN extend the paradigm of Online Social Networks with additional interaction opportunities generated by the users' mobility and opportunistic wireless communications among users which share interests, habits and needs. Specifically, CAMEO is designed to collect and reason upon multidimensional context information, derived by the local device, the local user and their physical interactions with other devices and users. It provides a common API to MSN applications through which they can exploit context- and social-aware functionalities to optimize their features. CAMEO has been implemented on Android platform together with a real example of MSN application. Validation and performance evaluation have been conducted through an experimental testbed. This report describes the technical specifications of CAMEO's software architecture and related APIs and must be considered as additional material to the work presented in [1].

Keywords: mobile social networks, middleware, context-aware, social-aware, opportunistic networks

1. CAMEO Software Architecture: the high-level description

CAMEO is a context- and social-aware middleware platform designed to efficiently support the development of Mobile Social Network applications, belonging to heterogeneous domains (e.g., health and well-being, work life, social support). It is designed as a light-weight and modular software architecture consisting of a single software package containing two subpackages (as shown in Figure 1):

- **Local Resource Management Framework (LRM-Fw)**, aimed at implementing features strictly related to the interaction with the local resources of the device, both hardware (e.g., embedded sensors) and software (e.g., communication primitives and programming libraries). It is also in charge of managing the interactions between the node and the remote sources (e.g., single external sensors, sensors networks, centralized repositories)
- **Context-Aware Framework (CA-Fw)**, aimed at storing and processing all the collected context information.

In addition, CAMEO provides an API towards MSN applications and it directly interacts with an external module for the user's profile definition called User Profile Module.

1.1. Local Resource Management Framework

LRM-Fw is composed of three software modules:

Network Manager. In order to deploy real-time MSN, CAMEO allows mobile devices to exploit all the opportunities to communicate and exchange data through opportunistic communications. To this aim, the Network Manager interacts with all the available wireless communication interfaces (e.g., WiFi ad-hoc, WiFi infrastructure mode, Bluetooth) selecting the best communication medium under specific conditions. It is also in charge of notifying other interested CAMEO components (specifically, the Transmission Manager) about the status of the connectivity between the local mobile device and her neighbors (e.g., WiFi link status and quality information, Bluetooth active/not active status).

Transmission Manager. After the Network Manager has selected the wireless interface for the transmission of a specific message (either middleware or application messages), the Transmission Manager is in charge of establishing the communication channel between a source and a destination

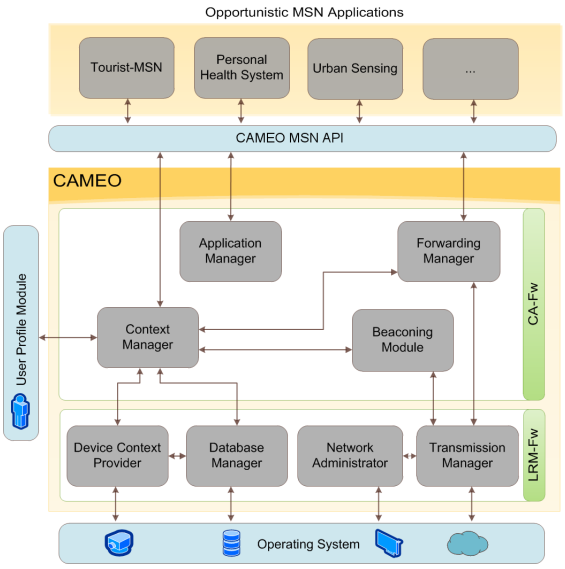


Figure 1: CAMEO software architecture.

node through the use of standard communication primitives (e.g., socket, TCP/UDP protocols and related parameters). It also receives notification messages from the Network Manager in case of link errors or disconnection events towards the message destination node.

Database Manager. It is responsible for the interaction of LRM-Fw with a SQL database that implements the CML model of CAMEO well-being context described in [1].

Device Context Provider. It is in charge of collecting context data derived from internal components of the mobile phone (e.g., embedded sensors, storage capacity level, battery level, resources consumption). Data specification and related parameters (e.g., sampling frequency for GPS or accelerometer, CPU occupancy threshold) are provided by the interaction of this module with the CA-Fw, following the directions provided by upper-layer applications or internal modules. In addition, Device Context Provider is able to manage data collected from heterogeneous sources (either internal or external to the mobile device) as specific support to participatory and opportunistic sensing services. It is worth noting that both embedded and external sensors are generally characterized by proprietary specifications and data formats. Thus, in order to guarantee the interoperability of CAMEO with those sources, we defined a new light-weight standard

for efficiently identifying and encoding heterogeneous sensing information on mobile devices, called *Sensor Mobile Enablement* (SME). SME is implemented in CAMEO as a software library managed by Device Context Provider. SME is compliant with OGC Sensor Web Enablement (SWE) standards [2] designed for web services dedicated to collect sensing data. In this way CAMEO is also able to establish bidirectional communications with sensor web services and to forward this information to the opportunistic network. We recently presented SME and its integration in CAMEO in [3], demonstrating the efficiency of SME for sensing data management.

1.2. Context-Aware Framework

CA-Fw represents the core of CAMEO, being responsible for the collection, management and processing of all the context information (local, external and social) and the development of internal context- and social-aware services (e.g., forwarding protocols, resource sharing services). It is composed of the following software modules:

Beaconing Module. It implements the periodical context exchange among 1-hop neighbors. This procedure allows CAMEO to discover new neighbors inside the current physical community and to build up and maintain the social context of the local node. Only a subset of the local context components are disseminated through the network. Specifically, the device context is not exchanged with other neighbors due to its real-time nature. In fact, it represents local measurements of internal resources with limited temporal validity, and it is generally locally processed to evaluate the feasibility of specific actions (e.g., the local node receives a download request from a neighbor and it checks its local resources, like battery lifetime, before accepting and managing it). As far as the sensing context is concerned, only a summary of the available information on the local node is included in the beaconing message so that interested nodes and applications can directly request specific information. In order to avoid the periodical transmission of large quantity of data, the Beaconing Module implements an optimized data exchange procedure.

Forwarding Manager. It is responsible for the implementation of end-to-end communications. Specifically, it is designed to implement optimized forwarding protocols for opportunistic networks to successfully deliver a message to a multi-hop destination in case of intermittent connectivity (e.g., HiBOp forwarding protocol [4]).

Application Manager. It is in charge of establishing a communication channel between each MSN application and CAMEO through MSN API (see

Sections 1.3 and **??**).

Context Manager. The main functions of Context Manager can be summarized in the following points: i) management of the well-being local, external and historical contexts; ii) interaction with the Database Manager to retrieve/store context data from/to the database; iii) implementation of algorithms and procedures for context reasoning, identification of specific situations and related middleware adaptations as detailed in Section **??**.

In order to provide efficient and reliable access methods to context information at run time, Context Manager implements four separate data structures that partially reflect the database content:

well-being Local Context (wbLC): it contains the context information related to local context components divided in the following structures: *User Profile* derived from Context Manager interactions with the User Profile Module[1] (see Figure 1); *Application/Service Context* specified by each application developed on top of CAMEO and by single internal services; *Device Context* derived from the interaction of Context Manager with DeviceContextProvider. The latter contains all the information related to the device's status, such as sensor's data and the available resources.

External Context (EC): it contains references to the external context types available in the database (e.g., pollution data, noise data, weather data). In this way, EC maintains a list of the available sensing services on the local node and a set of pointers to the respective data actually stored in the local database. Since the external context can be represented by a huge quantity of data, it is not efficient to manage it at run time through simple data structures.

current community Social Context (ccSC): it contains the list of current 1-hop neighbors of the local node and their context information disseminated through periodical beaconing messages. The content of ccSC represents thus a snapshot of the physical community of the local node in a specific instant.

historical Social Context (hSC): it contains the list of communities previously visited by the local node associated with a timestamp as the temporal information of the last visit and a counter to maintain the number of

[1]We decided to design this module externally to CAMEO since it can be implemented as a stand-alone application dedicated to the collection of user's personal information independently of the running applications and services. However, the information provided by User Profile Module are integrated in wbLC with information provided by other services and applications that are mainly related to the user's interests, habits and so on.

visits in a predefined period of time. In addition, for each visited community hSC maintains the list of encountered nodes and their context. This information is essential to implement social-oriented policies for the evaluation of context-based utility functions (see [1] for details). Information in hSC are periodically updated, maintaining at run time a subset of all the historical context based on predefined configuration parameters (e.g. time threshold, frequency). All the other information are stored in the database.

1.3. CAMEO APIs towards MSN applications

CAMEO APIs provides a full access to CAMEO context- and social-aware functionalities for the development of MSN applications. Since the communication between CAMEO and MSN applications is bidirectional, we define two distinct APIs for applications' requests and CAMEO notifications (e.g., messages, events, errors). In the following we provide a high-level description of the possible interactions between MSN applications and CAMEO. The complete specification of CAMEO APIs can be found in Section 2. A practical example of MSN application and its implementation is then presented in Section 1.5.

- **Registration**. Each application must register to CAMEO in order to access its internal functionalities. During the registration a unique identifier is assigned to the application and a callback interface is established. In the case an application is interested in an internal or remote service provided by CAMEO (e.g., embedded sensing features or remote connections to external sensing services), it must specify the type of service it is interested in during the registration procedure, so that CAMEO will be able to satisfy its request.

- **Application Context specification**. Each application specifies the set of context information relevant for its execution in order to be evaluated by CAMEO Context Manager. This information characterizes the running application on the local node and it is disseminated over the network through the beaconing procedure as part of the local context.

- **Utility function evaluation**. Each MSN application can define an algorithm for the evaluation of the utility of a specific content. The utility function is expressed in the form of a set of criteria to be applied to a given content through logic operations. The application can ask CAMEO to evaluate the utility function by passing the utility function,

the id of the content to be evaluated (stored by CAMEO) and the social policy.

- **Message sending/receiving**. MSN applications can send/receive messages through peer-to-peer communications and they are notified in case of failure during a message sending[2]. Through these messages applications can send/receive messages using their own communication protocols, as well as messages to request the exchange of context data (both local and remote) and services.

CAMEO notifications towards MSN applications are implemented using the callback interfaces created during the registration procedure. To manage different concurrent applications, CAMEO maintains the list of registered and currently active callback interfaces assigning a logical communication port to each of them. A special communication port is used by Context Manager for the context exchange over the network and its interactions with the other CAMEO internal modules. CAMEO notifications are related to the following events:

- **New application content discovery**. Every time CAMEO finds a new application content from a remote node it informs the interested application.
- **New service discovery**. Every time CAMEO finds a new service available on a remote node (including sensor web services or external sensing devices) it informs the interested applications. In case of a sensing service, CAMEO informs the interested applications by sending them the description and the capabilities of the involved sensors.
- **New neighbor discovery**. CAMEO informs the applications when a neighbor enters/exits the 1-hop area.
- **New community detection**. CAMEO informs the applications when the community detection algorithm results in a physical community change.

[2]In case of exchange of large application contents, system's primitives for the standard message exchange can present overload problems due to the predefined memory size assigned to each Android process (32MB). To overcome this issue, CAMEO implements a file segmentation procedure splitting the requested content into fixed length data chunks (512Kb each). The correct reception of each chunk is acknowledged, so that the sender node can manage automatic retransmissions of not acknowledged chunks.

1.4. Android implementation

To validate CAMEO functionalities and evaluate its performances, we implemented it on Android 4.2.1 platform. We have chosen Android due to its constantly rising popularity and because it naturally supports Java-based distributed and concurrent applications in addition to an easy access to system information like those related to embedded devices (such as GPS, sensors, camera). To better understand our implementation choices, we briefly introduce the definition of basic Android software components provided to developers. For additional technical details we refer the reader to [5].

The Application Framework is the main Android component provided to the external developer. On top of this framework, developers can design their own applications with the same full access to APIs used by the core applications. An Android application is composed of four components: (i) *Activities*, representing the Graphical User Interface of the related application; (iii) *Services*, which allow the background execution of independent tasks; (iii) *Broadcast Receivers*, which listen for broadcast event communications among different applications and the other Android modules; (iv) *Content Providers*, which make a specific set of application's data available to other applications. The activation of the three first components and their following interactions are implemented through the *intent* mechanism: asynchronous messages that are exchanged between the Application Framework components, containing the definition of the action to be performed.

Since CAMEO is a middleware platform supporting multiple concurrent applications, we decided to implement it as an Android Service. Thus, MSN applications, designed and developed to interact with CAMEO, are implemented as Android applications. A single instance of CAMEO, running on a separate process, is shared among all the applications. To support the communication between CAMEO and MSN applications, we exploit an Android technique based on the interprocess communication paradigm (IPC). The interfaces defined for IPC are based on Android Interface Definition Language (AIDL) similar to other popular languages based on CORBA or COM specifications (e.g., Java CORBA IDL [6] and C++ CORBA IDL [7]). The data that can be transferred through AIDL interfaces is limited to Parcelable objects, which are designed as a high-performance IPC transport objects. This mechanism allows fast data exchange to the detriment of limited design flexibility due to the lack of standard functions for the marshalling of Parcelable objects.

Regarding CAMEO support for direct communication between devices, the current implementation uses WiFi ad-hoc mode, configurable only

through customized firmwares requiring root access to the device. To solve this issue, we are working to support communication based on WiFi Direct standard [8], natively supported by the latest versions of Android.

1.5. Tourist-MSN Application

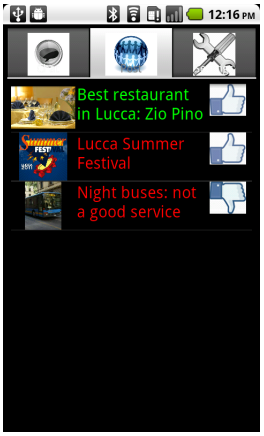
To better understand how to develop a real MSN application on top of CAMEO, we present the Tourist-MSN [9]. It is aimed at improving people experience during tourist visits by allowing individuals to create, collect and share useful information, related to geo-located points of interest (POIs). Contents are exchanged through opportunistic communications among users' mobile devices. Tourist-MSN provide users with two main functionalities:

- the generation and sharing of multimedia contents, denoted as *post* and characterized by a title, a textual content (to comment or express impressions related to the POI), and optional information like audio files, images or videos. Posts are divided into categories (e.g., event, cultural visit, transportation, restaurants) in which users can express their interests.

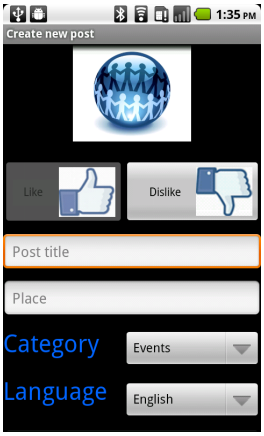
- real-time textual communications through an *opportunistic text chat* inside a limited group of users in proximity. Each chat is identified by a title and a category.

Tourist-MSN specifies (through MSN API) the following information as application context to be disseminated over the network by CAMEO beaconing procedure: (i) title and category of each post and chat generated by the local user; (ii) user's interests in specific categories of posts and chats. In this way, each node becomes aware of other nodes running Tourist-MSN in its current physical community and the list of available contents. Even though each node maintains a historical profile of neighbors and contents encountered in different physical communities, the management of a real-time chat is limited to the current physical community due to intermittent connectivity conditions characterizing opportunistic networks. However, since posts' distribution results in an asynchronous content exchange, Tourist-MSN provides CAMEO with the utility function algorithm designed to implement the context- and social-aware dissemination of posts among different physical communities. Moreover, since users can increase the content of a post by adding their own comments, CAMEO is also able to manage and distribute the content updates to the interested nodes.

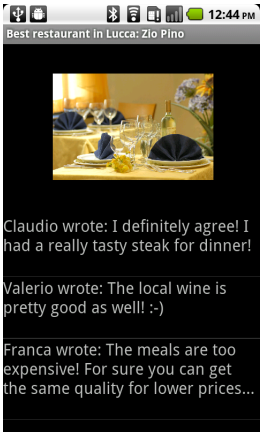
Every time a new post or a new chat matching the interests expressed by the local user becomes available in the neighborhood, CAMEO notifies



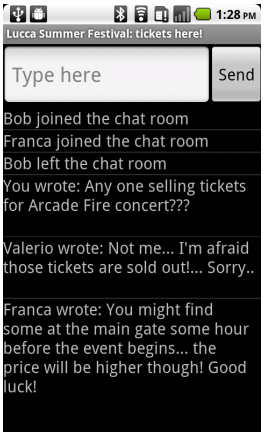
(a) List of available posts. In green: posts already downloaded. In red: posts available for download



(b) Interface for the creation of new posts



(c) Post comments



(d) Real-time opportunistic chat

Figure 2: Screenshots of Tourist-MSN Graphical User Interface

Tourist-MSN with an event message. The application then notifies the user through the GUI about the availability of the new content and the user can decide whether to download the post (or join the chat room) or not. In case the local node experiences a community change and there is one or more new available contents, CAMEO evaluates the Tourist-MSN utility function on each content with respect to the interests of users previously encountered in a different physical community by implementing Uniform Social (US) or Most Frequently Visited (MFV) social-oriented policies described in [1]. CAMEO provides then the application with a ranked list of the available contents and checks the feasibility of the related download procedures with respect to the local node's physical requirements (e.g., memory availability, permanence time in the current community). In case a new content with a higher utility becomes available but additional resources are required, CAMEO will discard the content with minor utility. The main purpose of this mechanism is to maximize the utility of contents to be disseminated in the network following specific social-oriented policies. Figure 2 shows some screenshots of Tourist-MSN application running on Google Nexus One.

In the following we present some code snippets related to the development of main Tourist-MSN operations. As a first step, Tourist-MSN starts using CAMEO establishing a connection between its Service component and CAMEO platform.

```
@Override
public int onStartCommand(Intent i, int startId, int flags) {
 Log.i(tag, "Connecting to CAMEO...");
 //Connect to CAMEO and start it if not already running
 bindService(new Intent("cnr.CAMEO.PLATFORM"),
   CAMEOConnection, Context.BIND_AUTO_CREATE);
 return START_STICKY;
}
```

Then, Tourist-MSN has to make a specific registration request to CAMEO in order to have full access to CAMEO functionalities (e.g., to send and receive messages over the network, to exploit content dissemination protocols). In the following code we report the basic instructions used by Tourist-MSN to require a registration, providing a logical port and a callback interface used by CAMEO for event notifications and replies. Note that the variable called `CAMEO` represents the interface used by Tourist-MSN to interact with CAMEO. Once connected, Tourist-MSN can use local messages with predefined values to access CAMEO functionalities (in the example below Tourist-MSN asks CAMEO for the local user's context).

```
@Override
public void onServiceConnected(ComponentName name,
  IBinder service) {
 CAMEO = PlatformInterface.Stub.asInterface(service);
 if (!registered) {
  try {
   resp = CAMEO.registerClient(PORT, callback);
   ...
   if (resp.getType() == LocalMessage.SUCCESS) {
    registered = true;
    key = (Long) resp.getContent();
    try {
     resp=CAMEO.sendLocalRequest(new LocalMessage(
     LocalMessage.GET_USR_CONTEXT, key));
     ...
```

As far as CAMEO notifications are concerned, Tourist-MSN must define the events for which it wants to be notified and the subsequent operations by using the `callback` interface defined during the registration procedure. In the example below, Tourist-MSN is notified by CAMEO when a new message is received, thus it must implement `onReceiveMessage` function specifying the operations to be executed for each type of message received.

```
private final CallbackInterface.Stub callback =
  new CallbackInterface.Stub() {
 @Override
 public void onReceiveMessage(LocalMessage packet,
   byte[] source) throws RemoteException {
  TouristMessage msg = (TouristMessage) packet.getPayload();
  switch (msg.getType()) {
   case TouristMessage.CHAT_MESSAGE:
   String chatMSG=(String)msg.getContent();
   ...
```

As a last example, we show in the following how Tourist-MSN specifies its utility function to CAMEO. The application can request CAMEO to evaluate the utility function after a specific event (e.g., new available content, community change). In general, the utility function is expressed as a list of criteria to be applied to a given content and its properties through a logical operation. Each criterion is independently evaluated, then the results of all

the criteria are combined together as a weighted sum (according to the given weights and the specific social policy), to obtain the overall utility of the content. As an example, in the following piece of code, Tourist-MSN specifies the utility function as a match between the property "category" of a post and the interest of the user, defined by the preference value "museum", and it requests CAMEO to evaluate it every time a new content is available in the neighborhood. If the result of the logical operation is true, the criterion assumes the value specified by the last parameter (weight) passed to the function addEvaluationCriterion (1 in this case).

```
private final CallbackInterface.Stub callback =
  new CallbackInterface.Stub() {
 @Override
 ...
 //property = 'category', preference = 'museum'
 //opType = 'match', weight = 1
 utilityFunction = new ContentEvaluator()
  .addEvaluationCriterion(property, preference,
  opType, weight);
 socialPolicy = 1 //MFV policy
 public void onReceiveMessage(LocalMessage packet,
   byte[] source) throws RemoteException {
  TouristMessage msg = (TouristMessage) packet
   .getPayload();
  switch (msg.getType()) {
   case TouristMessage.NEW_POST:
   CAMEO.evaluateUtility(utilityFunction,
    msg.getId(), socialPolicy)
   ...
```

2. CAMEO Technical Specification

In this Section we give a detailed description of CAMEO software architecture, starting from software packages up to class definition. All the classes have been written in Java programming language for the Android OS.

CAMEO is composed of the following software packages:

- cnr.CAMEO
- cnr.CAMEO.CAFw

- cnr.CAMEO.CAFw.ContextManaging
- cnr.CAMEO.LRMFw
- cnr.Common

2.1. cnr.CAMEO

This package contains basic classes of CAMEO, such as the class of the main GUI launched when CAMEO starts running, some custom exceptions classes and the external module dedicated to the user context management (User Profile Module). The package is composed of the following classes:

- `MainActivity`: extends `Activity` (Android). This class represents the main GUI of CAMEO and it is the first class launched by CAMEO. The purpose of the `MainActivity` class is to provide a simple graphical interface by which the user can control the execution of CAMEO service (i.e., start and stop its execution). Moreover, the GUI contains a button by which the user can launch the `UserProfileModule Activity`.

- `RemoteListEntry`: This class defines an entry of the Android `CallbackRemoteList`, the structure where the applications callback interfaces are maintained. CAMEO uses these callback interfaces to communicate with the registered applications. When an application registers to CAMEO, a `RemoteListEntry` is added to the `RemoteCallbackList`. The entry is removed in case of application log out or crash.

- `ServiceManager`: extends `Service` (Android). It is the main component of CAMEO as the Android `Service` that, running in background, provides all context-aware and opportunistic features to the registered applications. The `ServiceManager` is launched by the `MainActivity` and is responsible for the creation of all the components of CAMEO CA-Fw (i.e., the `ContextManager`, the `Forwarding` module, and the `Beaconing` module). The functionalities of the `ApplicationManager` (see Section 1.2) are implemented inside the `ServiceManager`, indeed the `ServiceManager` implements an AIDL interface called `PlatformInterface`, which provides access to CAMEO MSN APIs to the applications. The `ServiceManager` takes also care of the following tasks:

  1. to register each application that wants to interact with CAMEO, setting a unique identifier (i.e., a port number);

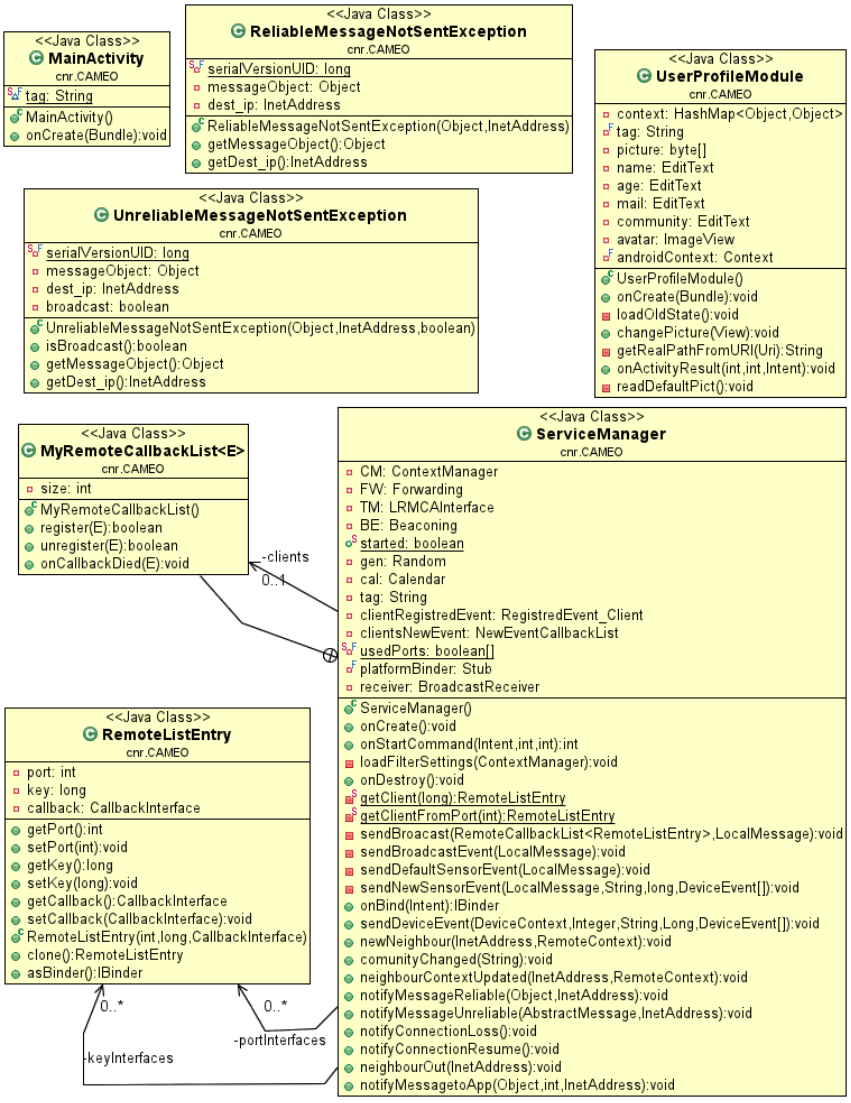


Figure 3: cnr.CAMEO package - UML class diagram

2. to receive and process requests coming from the registered applications (see Section 3 for further details);
3. to notify the applications when an event occurs or a message is received (using the identifier to select the right application). A detailed description of this notifications can be found in Section 4.

- `UserProfileModule`: extends `Activity` (Android). This class defines the module in charge of acquiring and managing the user profile (e.g., preferences, personal profile, ...). The `UserProfileModule` is a GUI component that allows the user to insert and successively modify his personal information into a set of pre-defined fields (e.g., name, gender, age, home community, preferences).

- `ReliableMessageNotSentException`: extends `Exception`. This class defines a custom exception used to handle the errors generated when CAMEO tries to send a message using the reliable transmission protocol (TCP), but the `TransmissionManager` (package cnr.CAMEO.LRMFw) is not active or it is in idle state.

- `UnreliableMessageNotSentException`: extends `Exception`. This class defines a custom exception used to handle errors generated when CAMEO tries to send a message using unreliable transmission (UDP), but the `TransmissionManager` (package cnr.CAMEO.LRMFw) has not been started yet or it is in idle state.

2.2. cnr.CAMEO.CAFw

This package contains all the classes related to the Context-Aware Framework (CA-Fw, see Section 1.2 for further details on the services provided by this framework). The package is composed of the following classes:

- `Beaconing`. This class represents the `BeaconingModule` in charge of generating periodic messages, also known as beacons (defined by the `BeaconMessage` class), used by CAMEO to advertise the presence of a node and its context towards the 1-hop neighbors in the network. `Beaconing` uses a Java `Timer` to generate the beacons periodically and passes them to the `TransmissionManager`, which is responsible for sending the messages over the network.

- `BeaconMessage`: extends `AbstractMessage`. This class defines the structure of a beacon message. It contains a `BeaconContent` and a timestamp that reflects the beacon creation time. The timestamp is

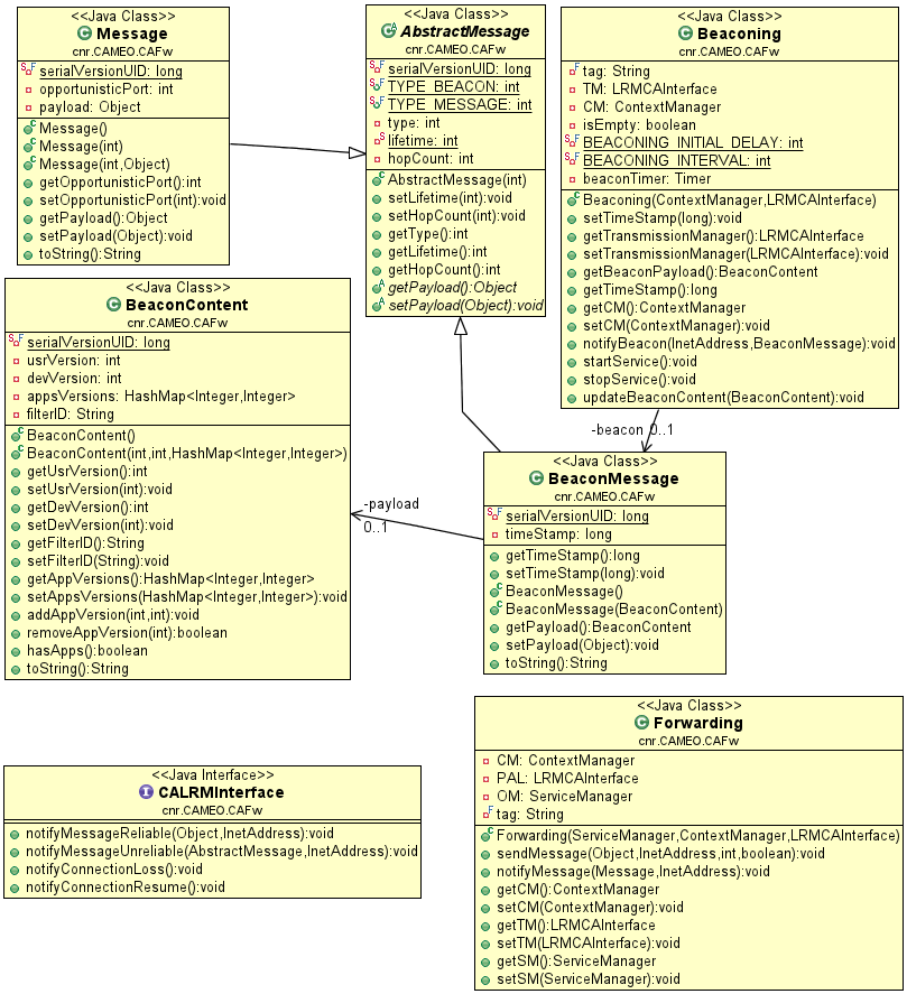


Figure 4: cnr.CAMEO.CAFw package - UML class diagram

used in conjunction with the contexts' version numbers included in the `BeaconContent`.

- `BeaconContent`: implements `Serializable`. This class defines the content of a beacon message. This content is formed of a set of version numbers related to user's and applications' context running on CAMEO and they are defined to optimize the dissemination of context's updates on the network. If CAMEO crashes or is restarted, the version numbers are reset to zero. In this case, the timeout contained in the `BeaconMessage` is thus used to identify contexts with the same version number, but with possibly different content. `BeaconContent` also contains the IP address of the sender, which is used as the unique node's identifier.

- `CALRMInterface`. This interface is used for the communication between the Local Resource Management Framework (LRMFw) and the Context-Aware Framework (CAFw). The modules of the LRMFw can send their notifications towardsthe CAFw using this interface, without knowing which module will elaborate the requests, ensuring the modularity of the system. It is implemented by `ServiceManager`.

- `Forwarding`. This class is designed to implement an opportunistic routing algorithm. Currently the class is empty and CAMEO uses only 1-hop communication. In future work we plan to implement a context-based forwarding protocol as HiBOp [4].

- `AbstractMessage`: implements `Serializable`. This is an abstract superclass defining a generic message, containing the properties and methods in common between the `BeaconMessage` class and the `Message` class.

- `Message`: extends `AbstractMessage`. This class defines a message to be exchanged over the network between applications through CAMEO. The message has a payload field, inherited from the superclass `AbstractMessage`. This generic payload represents the content of the message. To deliver the message to the right application at the destination node, CAMEO assigns a port number to each registered application and each `Message` is marked with the respective port number.

2.3. cnr.CAMEO.CAFW.ContextManaging

This is a sub-package of `cnr.CAMEO.CAFW` containing all the classes related to the management of context information. The content of the package

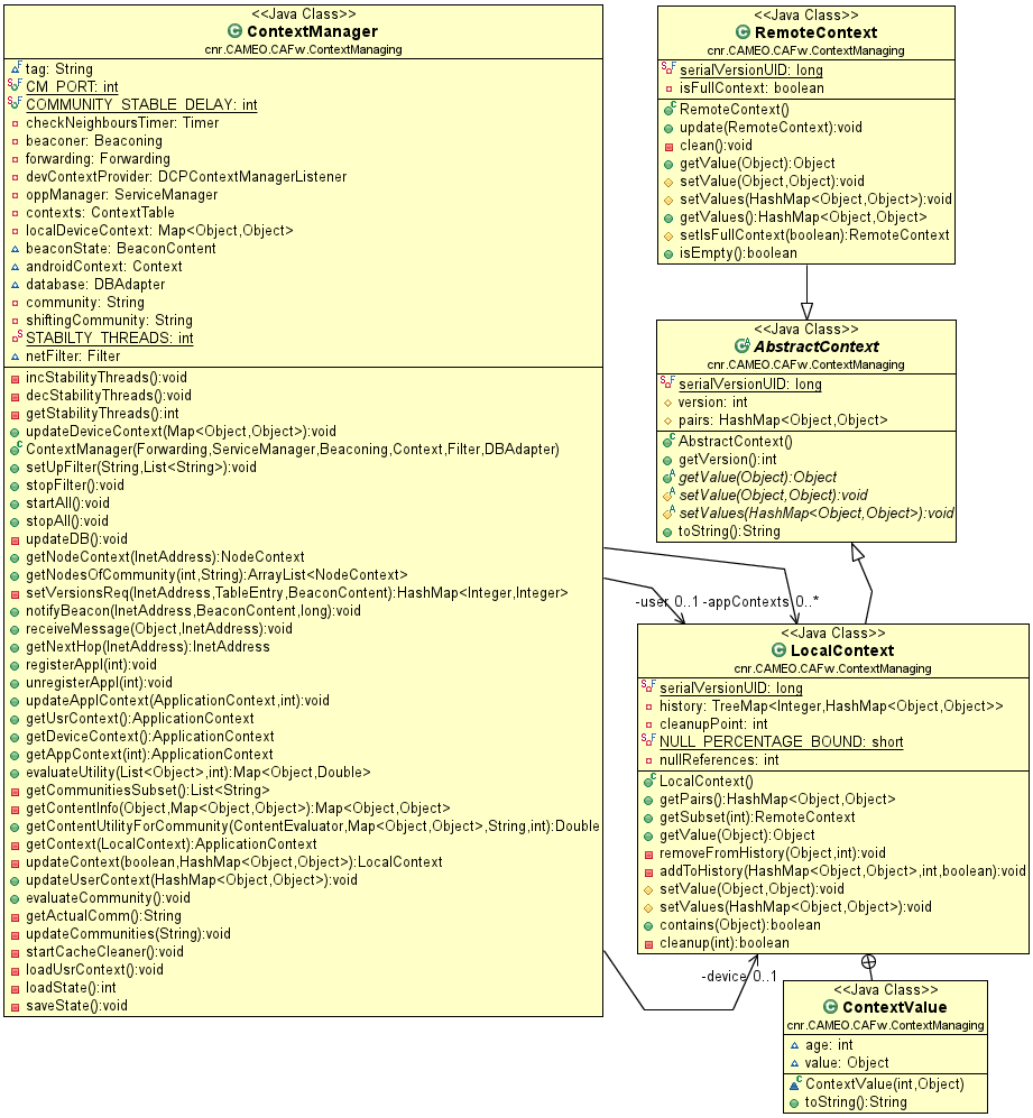


Figure 5: cnr.CAMEO.CAFw.ContextManaging package - UML class diagram (1)

Figure 6: cnr.CAMEO.CAFw.ContextManaging package - UML class diagram (2)

is the following:

- `AbstractContext`: implements Serializable. This is a superclass representing a generic context. The `AbstractContext` contains a version number and an `HashMap` representing the key-value pairs related to a context.

- `DeviceContextUpdater`. This interface allows the ContextManager to update the status of the local device context following the input of Local Resource Management Framework .

- `ContextManager`: implements `DeviceContextUpdater`. This is the main class for the management of context data. The `ContextManager` is responsible for the maintenance of context information related to the user, the device and the applications involving the local node and the remote nodes encountered in the network. It is in charge of exchanging `ContextMessages` between nodes in the network, performing the synchronization of context data between peers. The `ContextManager` manages the `Context`, adding, updating or removing context information related to remote nodes when needed. Moreover, it maintains the `LocalContext`, containing the context data of the local node and the `External Context`, containing context data coming from external sources. Through the `DeviceContextUpdater`, the `ContextManager` exposes the methods used by the applications to modify the local device context. The `ContextManager` checks the beacons received from the `Beaconing` module to find if a neighbor has entered or quit the 1-hop area. When a new neighbor is detected, the `ContextManager` adds an entry to the `Context`. Then, it periodically checks the ccSC (see Section 1.2 for further details) to determine if a neighbor has left the 1-hop area. The `ContextManager` also evaluates, after an explicit request received from an application, the utility of one or more application content for one or more nodes of a community or a set of communities by using the utility function provided by the application and by adopting the social-oriented polices required by the application. The `ContextManager` is also responsible for the discovery of the current community of the local node. As far as the external context is concerned, the `ContextManager` receives from the applications the type of external context they are interested in at the registration time. Then, it informs the underlying framework (LRMF - see Section 2.4) by passing the external context types required by the applications. The `ContextManager` is notified by the underlying framework when new

external sensing services are available and it informs the interested applications. The applications can decide to directly contact the sensing services to receive the related data, by sending special messages to CAMEO through the API, as specified in 3. These messages are processed by the `ContextManager`, which checks the `ExternalContext` data structure before passing the request to the underlying framework and, in case the requested information is already in the local memory, it passes it to the application.

- `ContextMessage`: implements `Serializable`. This class defines a special type of message used to exchange context information between nodes in the network. The `Beaconing` module advertises available context data inside the 1-hop neighborhood. If a node finds new available context information from a new neighbor, or a new version of a context from a known neighbor, it sends a request to the remote node through a `ContextMessage`, eventually indicating the version of the context it already has to perform an incremental synchronization. The node which receives a context request sends its whole context (or part of it with respect to the version number received from the other node) to the requester, using a `ContextMessage`. The `ContextMessage` is used for user, application, and external context components.

- `Context`. This class represents the data structure holding the context information related to remote nodes. Specifically, this class implements the data structures defined as ccSC, hSC and EC (see Section 1.2). The ccSC data structure is implemented as a Java `Map` object, where the IP address of the remote nodes is used as an index and each entry contains the context data of the remote node. When a new neighbor is detected, the `ContextManager` notifies the `Context` object, which adds an entry to the ccSC. If the beacons of a neighbor are not received for a certain period of time, the `ContextManager` asks the `Context` object to remove the respective entry from the ccSC. In this case the `Context` object moves the information related to the node that has left the 1-hop area from the ccSC to the hSC. The ccSC and the hSC are also accessed by the `ContextManager` during the evaluation of the utility function for a given application content. The `Context` object maps the data in the persistent storage, maintained by the LRMFw.

- `LocalContext`: extends `AbstractContext`. The `LocalContext` represents the context information related to the local node, including

the application and the user context. It is the implementation of the well-being Local Context (wbLC). This data structure is implemented as a set of Java `HashTable`, each of which represents a context (i.e., user, device and application) and contains its key-value pairs. The information of the `LocalContext` is sent to the other nodes in the network using `BeaconMessage` and `ContextMessage` objects. When the `ContextManager` receives a `ContextMessage` with a request for a certain context with a specified version number, the `ContextManager` checks if the version number and the timestamp provided by the remote node are valid, then it returns the subset of information required by the remote node to synchronize its context data with the local data. Each change related to the data of a certain local context is tracked within a Java `TreeMap` object called `history`, so that each entry of the `history`, indexed by a version number, points towardsthe key-value pairs modified during the update. Each application can provide multiple updates at the same time to avoid frequent context updates which can lead to network congestion due to the potentially large amount of data to be exchanged for the synchronization procedure. Subsequent updates of the same context key-value pairs are collapsed into the last version number of the `history` data structure to reduce the `TreeMap` size. If the number of updates of a context data exceeds a pre-defined threshold, all the versions are collapsed into the last version number of the `history`.

- `ExternalContext`: extends `AbstractContext`. This class defines the structure of the External Context data structure (EC). It contains information regarding data collected from external components, such as wearable sensors or fixed sensing stations. It is exchanged with remote nodes similarly to the `LocalContext`. The `ExternalContext` object maps the data in the database related to the external context, maintained by the LRMFw.

- `NodeContext`: implements `Serializable`. The `NodeContext` class defines a wrapper for all the context information related to a remote node and represents a generic entry for the hSC of the `Context` object. The `NodeContext` object contains the user context and the list of application contexts related to a remote node.

- `RemoteContext`: extends `AbstractContext`. This class defines a generic context related to a remote node. A `Remote context` contains the key-value pairs that defines the context.

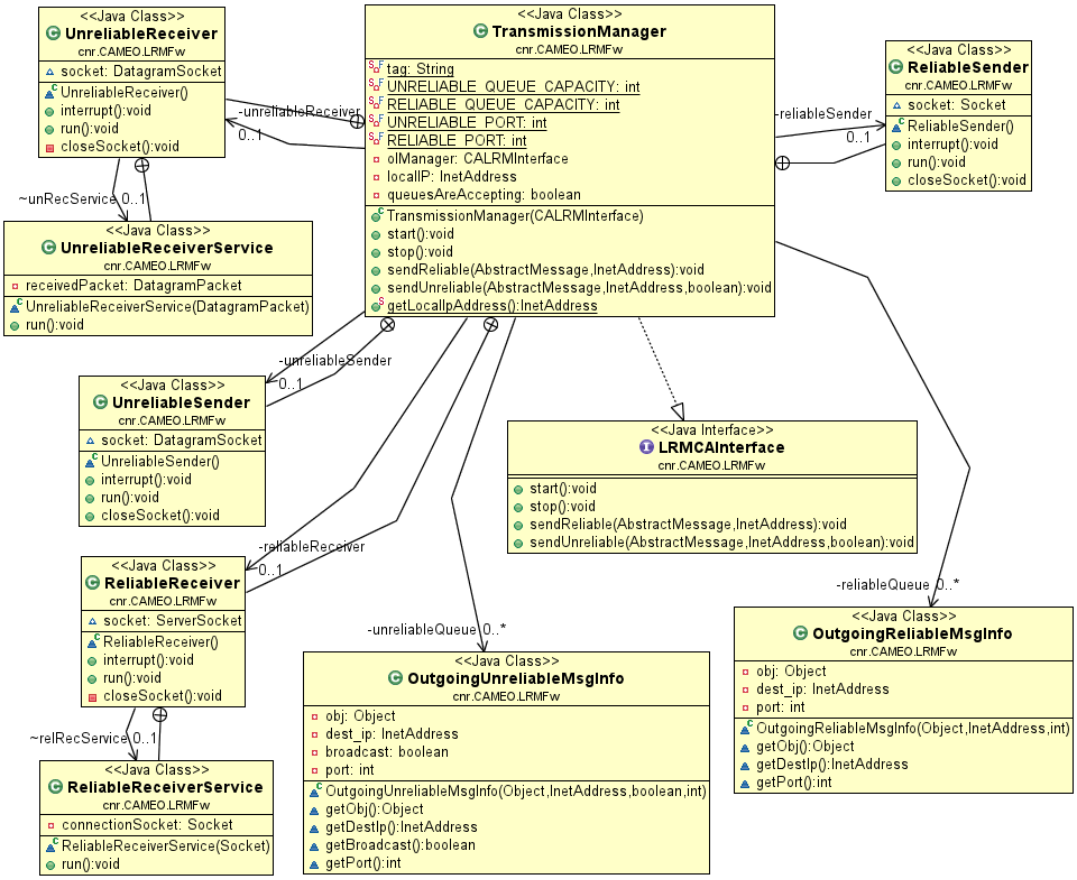


Figure 7: cnr.CAMEO.LRMFw package - UML class diagram (1)

- `CcscEntry`: implements `Serializable`. This class represents an entry of the ccSC data structure, indexed inside the `Context` object with the IP address of the respective remote node. Each `CcscEntry` contains a `NodeContext` object and a timestamp, used to discover a `neighbor_out` event (in case the beacons of a certain remote node are not received for a pre-defined period of time).

2.4. cnr.CAMEO.LRMFw

This package contains all the classes concerning the Local Resource Management Framework of CAMEO. This framework is in charge of performing all the lower level tasks, such as the exchange of messages over the network, the management of the wireless network interface, the management of the

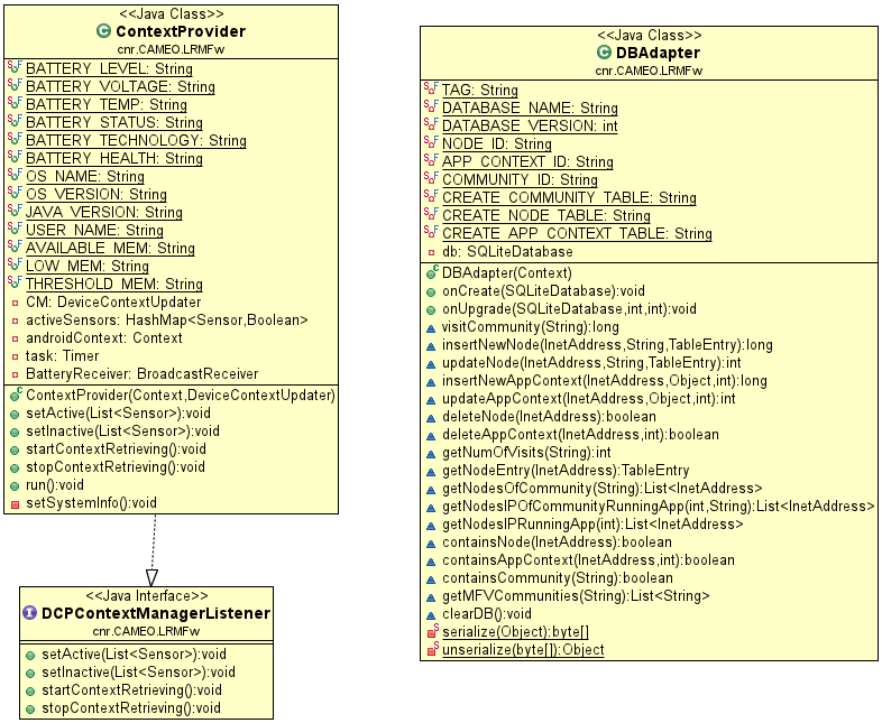


Figure 8: cnr.CAMEO.LRMFw package - UML class diagram (2)

SQLite database and the access to the information related to the hardware sensors of the mobile device. The package is composed of the following classes:

- `ContextProvider`: extends `Thread`, implements `DCPContextManagerListener`. The `ContextProvider` is in charge of acquiring information regarding the context of the device, including sensors data, battery status, memory usage, etc. The `ContextProvider` periodically checks the status of different hardware components with different sample rates. These parameters are defined by the `ContextManager` class of the CA-Fw based also on applications' requirements. In addition, the `ContextProvider` autonomously look for external sensor data from fixed stations or sensors connected to the device, according to the types of sensor specified by the running applications. It manages the communications between the device and the external sources, by implementing different communication standards, including Sensor Mobile Enablement [3], as described in Section 1. When it discovers new available external sensor data, it notifies the upper layer, which eventually informs the interested applications. If the upper layer needs to download external context data from remote sources, it informs the `ContextProvider`, which starts the download and passes the obtained data to the CAFw.

- `DBAdapter`: extends `SQLiteOpenHelper` (Android). This class contains all the methods used to access and manage the SQLite database defining the model of well-being context.

- `DCPContextManagerListener`. This interface allows the `ContextManager` to retrieve information concerning the device, gathered through the `ContextProvider`.

- `LRMCAInterface`. This interface defines the methods to be used by the CAFw to communicate with the LRMFw.

- `NetworkManager`. It interacts with Android to manage the wireless network interface. It can request Android to activate/deactivate the network interface and it listens for status changes of the interface.

- `OutgoingReliableMsgInfo`. This class represents a generic message stored inside the outgoing queue of the `TransmissionManager`. Each of these messages has a destination address, a port (used by CAMEO to identify the destination application) and a payload containing the

message itself. This class is related to a message to be sent using the reliable communication protocol (TCP).

- `OutgoingUnreliableMsgInfo`. This class represents a generic message stored inside the outgoing queue of the `TransmissionManager`. Compared with the `OutgoingReliableMsgInfo`, this class concerns unreliable communications (UDP).

- `TransmissionManager`: implements `LRMCAInterface`. The `TransmissionManager` is responsible for the data transmission over the network. It provides two different types of communication: reliable and unreliable. The former uses Java `ServerSocket` and `Socket` objects, while the latter uses `DatagramSocket` objects for data exchange. The `TransmissionManager` maintains two messages queues (for reliable and unreliable communication respectively).

2.5. cnr.CAMEO.LRMFw.Sensors

This package contains all the classes aimed at managing the collection of data coming from the different hardware sensors (embedded in the mobile device). The names of the classes are self-explanatory. The content of the package is the following:

- `AccelerometerManager`
- `GyroscopeManager`
- `LightManager`
- `MagneticFieldManager`
- `OrientationManager`
- `ProximityManager`
- `PressureManager`
- `TemperatureManager`

2.6. cnr.CAMEO.common

This package contains the classes shared between CAMEO and MSN applications, which must be imported by applications as external libraries to allow the application's access to CAMEO functionalities.

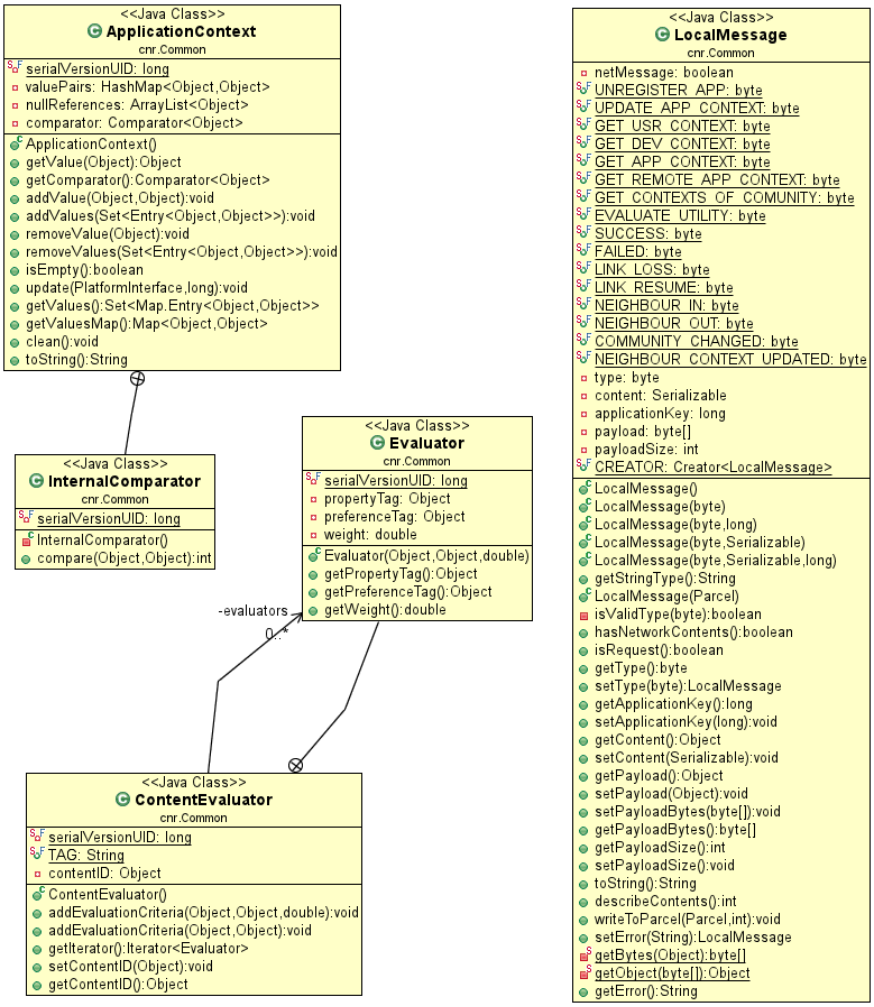


Figure 9: cnr.CAMEO.Common package - UML class diagram

- `ApplicationContext`: implements `Serializable`. This class represents the context related to a specific application as key-value pairs directly specified by the application. Each application can store or retrieve its own `ApplicationContext` object using CAMEO MSN API.

- `ContentEvaluator`: implements `Serializable`. This class is aimed at giving a simple way to define a set of criteria to be used by the `ContextManager` during the evaluation of the utility function. Each application can create its own `ContentEvaluator`, defining a list of fields (called evaluators) which represents the tags of the application contents to be matched with the respective preferences of the user. The application can weight the different evaluators with a value between 0 and 1. The utility function is computed as the weighted sum of the binary results of the matches of the fields defined by the evaluators. For example, a hypothetical application exchanging music content over the network defines two evaluators, named "genre" and "format" respectively, the former mapping the musical genre of an audio file with the preference of the user regarding musical genres and the latter identifying a link between the file format of the application content (e.g., mp3, wav, ...) and the preferred format of the user. The `ContextManager` matches the values mapped by the evaluators, returning 1 if the value related to the content and the preference of the user match, and 0 otherwise. The utility function calculated by the `ContextManager` for this application would be the sum of the values returned by the match of the two evaluators multiplied by their respective evaluator weights, then multiplied by the weight given to the community to which the involved nodes belong to, defined by the chosen social-oriented policy (see [1] for the detailed algorithm).

- `LocalMessage`: implements `Parcelable` (Android). This class defines the format of the messages exchanged between the applications and CAMEO. A `LocalMessage` can be an application request sent by the applications towards CAMEO or a CAMEO event sent towards the applications. A `LocalMessage` contains a `TYPE` field, which determines the type of request or notification. It contains also additional information (e.g., an application message to be sent over the network or additional parameters required by some requests) placed in two different optional fields. The first field is called `content` and it is used for local communication, while the second field is called `payload` and it is used for remote communication. The different types of `LocalMessage`

are explained in more details in Section 3.

In addition, CAMEO provides two AIDL interfaces to manage the communication between the middleware platform and each MSN application. Specifically:

- `MSNInterface` (AIDL). This interface contains the methods offered by CAMEO to the applications. The application directly uses MSNInterface object to call CAMEO methods.
- `CallbackInterface` (AIDL). This interface defines the methods called by CAMEO to send various notifications to the applications. The application must implement this interface and pass the instantiated object to CAMEO, which uses it to send each notification to the application.

  These interfaces are detailed in the following sections.

3. MSN Interface

MNS interface provides the following functionalities to MSN applications.

3.1. Register with CAMEO

- **Name of the service primitive:** `RegisterClient`
- **Description:** This method allows an application to register with CAMEO. CAMEO approves the registration if the port specified by the application is not yet in use, otherwise it denies the registration. The application can provide the types of external sensing services it is interested in. If CAMEO already has some information about the specified sensing service types in the local memory, it generates a series of *NEW_REMOTE_SERVICE_AVAILABLE* events (see 4 for further details). CAMEO adds the interface provided by the application during the registration to the callback interface list and uses it to communicate back to the application. A registered application can access the services provided by CAMEO, using all the other methods described in this Section. If an application is not registered, it can only use the `RegisterClient` primitive. A call to any other method will cause CAMEO to reply with an error message.
- **Type:** Confirmed, Synchronous, Local

- **Semantics:** The primitive shall provide the following parameters:
  1. *Port* (input)
     - Type: integer
     - Accepted values: a valid integer
     - Description: The port number used as unique `ID` to redirect the notifications and the messages received by CAMEO over the network to the right application.
  2. *External Context Type* (input)
     - Type: string
     - Accepted values: a valid external context type
     - Description: The external context type in which the application is interested. It is used by CAMEO to decide which kind of external context to forward to the application in case the node encounters external sources of sensor data.
  3. *Callback* (input)
     - Type: `CallbackInterface`
     - Accepted values: a valid `CallbackInterface` object
     - Description: This is the interface that the application must provide to CAMEO in order to register. CAMEO uses these interfaces to communicate back with the applications.
  4. *LocalMessage* (output)
     - Description: This is the returned value containing the result of the registration procedure.
     - Returned value: `FAILED` if the registration procedure has failed or a list of ids related to the external context data locally available otherwise.

- **When generated:** This primitive is generated when an application wants to register with CAMEO.

- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` (which implements the `PlatformInterface` interface) causes CAMEO to start the registration process. The `ServiceManager` checks if the port provided by the application is already in use. If this is the case it denies the registration, otherwise it accepts it.

3.2. Send a Generic Local Request

- **Name of the service primitive:** `SendLocalRequest`
- **Description:** This is a generic primitive provided by CAMEO to allow each application to request various types of services.
- **Type:** Confirmed, Local
- **Semantics:** The primitive shall provide the following parameters:
  1. *request* (input)
     - Type: `LocalMessage`
     - Accepted values: a valid `LocalMessage` object
     - Description: This parameter contains the specification of the request that the application wants to send to CAMEO. The request type is specified inside the field `TYPE` of the `LocalMessage` object. Additional content required by some requests can be placed inside the field `Content` of the `LocalMessage` object.
  2. *LocalMessage* (output)
     - Description: This is the returned value containing the result of the request.
     - Returned value: `FAILED` if the request has failed or `SUCCESS` otherwise.
- **When generated:** This primitive is generated when an application wants to request a service to CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` (which implements the `PlatformInterface` interface) causes CAMEO to evaluate and perform the actions required by the application, depending on the type of request the application has made. The different types of requests that the application can send to CAMEO are listed in the following.

3.2.1. Unregister with CAMEO

- **TYPE of LocalMessage:** `UNREGISTER_APP`
- **Description:** This type of local request is used by the applications that want to cancel their registration with CAMEO.
- **Additional parameters:** none

- **When generated:** This local request is generated when an application wants to unregister with CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to unregister the application and returns a `LocalMessage` object with the `TYPE` field set to `SUCCESS`.

3.2.2. Update Application Context

- **TYPE of LocalMessage:** `UPDATE_APP_CONTEXT`
- **Description:** This type of local request is used by the applications that want to update their application context (i.e., add, modify or delete the key-value pairs of the context data).
- **Additional parameters:** The application context that the application wants to store instead of the original one, saved into an `ApplicationContext` object.
- **When generated:** This local request is generated when an application wants to update its application context stored inside CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to update the context information related to the application that called the primitive. CAMEO distributes the new application context over the network, adding a new version number and performing the operations described in Section 2.3 regarding the `LocalContext` class. The request always returns a `LocalMessage` with the field `TYPE` set to `SUCCESS`.

3.2.3. Get User Context

- **TYPE of LocalMessage:** `GET_USR_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve the user context information.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read the user context information.
- **Effects of receipt:** If the user context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the user context does not exist, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.2.4. Get Device Context

- **TYPE of LocalMessage:** `GET_DEV_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve the device context information.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read the device context information.
- **Effects of receipt:** If the device context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the device context does not exists the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.2.5. Get Application Context

- **TYPE of LocalMessage:** `GET_APP_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve their own context data.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read their own context data.
- **Effects of receipt:** If the application context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the application context does not exists the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.2.6. Get Remote Application Context

- **TYPE of LocalMessage:** `GET_REMOTE_APP_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve the application context information of a remote node.
- **Additional parameters:** The IP address of the remote node of which the application wants to retrieve the application context.

- **When generated:** This local request is generated when an application wants to read an application context of a remote node.

- **Effects of receipt:** If the application context has been set for the given IP address, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field to `SUCCESS`. If the application context does not exists for the specified IP address, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.2.7. Get Remote Contexts For a Given Community

- **TYPE of LocalMessage:** `GET_CONTEXTS_OF_COMMUNITY`

- **Description:** This type of local request is used by the applications that want to retrieve the context information of all the nodes belonging to a selected community.

- **Additional parameters:** A string representing the unique identifier of the community for which the application wants to retrieve the context information.

- **When generated:** This local request is generated when an application wants to read the context data related to all the nodes of a given community. This situation usually occurs when the application wants to evaluate the utility function for a given application content with respect to the interests of the nodes of a specific community.

- **Effects of receipt:** If the given community identifier is valid and the set of contexts related to that community inside CAMEO is not empty, the `ServiceManager` returns the related context information to the application in the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If there is no information inside CAMEO regarding the given community, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.2.8. Get Sensing Service Data

- **TYPE of LocalMessage:** `GET_SENSING_SERVICE_DATA`

- **Description:** This type of local request is used by the applications that want to retrieve sensing data from a sensing service on a remote node.

- **Additional parameters:** An ID representing the unique identifier of the sensing service.
- **When generated:** This local request is generated when an application wants to obtain sensing data from a sensing service on a remote source. The application receives a notification from CAMEO when a new sensing service is available from a remote source (according to the sensing services types specified by the application). hence, the application can use this request to contact the external source and to obtain the external sensing data.
- **Effects of receipt:** The requested data is downloaded by CAMEO (or retrieved from persistent memory in case they had been already downloaded) and are passed to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field to `SUCCESS`. If the requested context data do not exists (or the remote service is not available anymore), the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`

3.2.9. Evaluate The Utility For One or More Application Contents

- **TYPE of LocalMessage:** `EVALUATE_UTILITY`
- **Description:** This type of local request is used by the applications that want to obtain an evaluation of the utility function for one or more application contents with respect to the interests of the nodes of a previously encountered community.
- **Additional parameters:** A list of ids related to the application contents for which the application wants to evaluate the utility, a utility function expressed by a list of `ContentEvaluator` and an integer specifying the preferred social-oriented policy.
- **When generated:** This local request is generated when an application wants to obtain an evaluation of the utility function for one or more application content.
- **Effects of receipt:** The `ServiceManager` asks the `ContextManager` to calculate the result of the utility function for the given content ids. The content properties are retrieved from the remote contexts stored inside CAMEO. The `ContentEvaluator` is used to match the properties of the given contents with the preferences of the user, found inside the local user context. To better understand how the `ContextManager`

calculates the utility of the given contents see Section 2.6. If the contents ids passed by the requester application are valid and are known to CAMEO, the `ServiceManager` replies to the application with a list containing the results of the utility function evaluations indexed by the content id. The result is placed in the field `CONTENT` of a `LocalMessage` object with the `TYPE` field to `SUCCESS`. If the content ids are not valid or unknown to CAMEO, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

3.3. Send a Message Over The Network

- **Name of the service primitive:** `SendMessage`
- **Description:** This method allows an application to send a customized message over the network to another node running an application using the same communication port as the sender.
- **Type:** Confirmed, Asynchronous, Remote
- **Semantics:** The primitive shall provide the following parameters:
  1. *packet* (input)
     - Type: `LocalMessage`
     - Accepted values: a valid LocalMessage with the boolean field `NET_MESSAGE` set to `TRUE` and the field `PAYLOAD` set with the content which the application wants to send over the network.
     - Description: This is the content of the message to be sent over the network.
  2. *dest* (input)
     - Type: IP address
     - Accepted values: any valid IP addresses
     - Description: This is the IP address of the destination.
  3. *broadcast* (input)
     - Type: boolean
     - Accepted values: { true,false }
     - Description: Whether or not the packet must be sent to all the nodes in the 1-hop area (like a beacon)
  4. *result* (output)
     - Type: boolean
     - Accepted values: { true,false }

  - Description: True if the message is successfully processed by the `TransmissionManager`. False in case of errors.

- **When generated:** This primitive is generated when an application wants to send a message to one or more nodes in the network. It is typically used to exchange application contexts.

- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to pass the message to the `ForwardingManager`, which calculates the best next hop for the delivery of the message and then passes the message to the `TransmissionManager`, which sends the message over the network towardsthe destination. If the `TransmissionManager` has not been started yet or is in idle state, a `ReliableMessageNotSentException` (or a `UnreliableMessageNotSentException` in case of a broadcast message) is thrown.

4. Callback Interface

Each application must extend `CallbackInterface`, implementing the methods called by CAMEO to notify events. During the registration procedure each application passes its custom `CallbackInterface` to CAMEO. CAMEO maintains a list of `CallbackInterface` in a dedicated data structure. An entry of this list is removed when the respective application logs out or crashes. The list is indexed by the application identifier. CAMEO sends two different types of notifications: (i) broadcast notifications (i.e., events to be sent to all the registered applications); (ii) direct notifications (i.e., events, errors or messages destined to a specific application). When CAMEO sends a notification, it passes to the applications a `LocalMessage` object, containing the details of the occurred event. In the following we provide a detailed description of the methods defined by the `CallbackInterface`.

4.1. Receive a Generic Asynchronous Event

- **Name of the method:** `onNewEvent`

- **Description:** This is a generic method used by CAMEO to notify different types of events.

- **When generated:** This method is invoked by CAMEO to notify the applications about an event occurred.

- **Parameters returned by CAMEO:** a *LocalMessage* containing the details of the notification. The different types of notifications are identified by the field `TYPE` of the returned *LocalMessage*. The types of event defined by CAMEO are the following:
  - *LINK_LOSS*: Generated by the `NetworkManager` when the status of the network interface changes from `CONNECTED` to `DISCONNECTED`. No extra parameters.
  - *LINK_RESUME*: Generated by the `NetworkManager` when the status of the network interface changes from `DISCONNECTED` to `CONNECTED`. No extra parameters.
  - *NEIGHBOR_IN*: Generated by the `ContextManager` when a new neighbor joins the 1-hop area. The IP address of the new neighbor is passed inside the `payload`.
  - *NEIGHBOR_OUT*: Generated by the `ContextManager` when a known neighbor leaves the 1-hop area. The IP address of the neighbor is passed inside the `payload`.
  - *COMMUNITY_CHANGED*: Generated by the `ContextManager` when the actual community changes. The unique identifier of the new community is passed inside the `payload`.
  - *NEW_CONTENT_AVAILABLE*: Generated by the `ContextManager` when a new content is available from a remote node.
  - *NEW_REMOTE_SERVICE_AVAILABLE*: Generated by the `ContextManager` when a new service is available from a remote node. The service can also be a sensing service from an external source matching the preferences of the application.
  - *NEIGHBOR_CONTEXT_UPDATED*: Generated by the `ContextManager` when new information regarding a neighbor context is available. This information can refer to both user and application contexts. Thanks to this notification the applications can discover new contents available for downloading from remote nodes. The IP address of the neighbor and the version numbers related to its contexts, received from the `Beaconing` module, are passed to the application inside the `payload` object.

4.2. Receive a Message From a Remote Node

- **Name of the method:** `onReceivedMessage`

- **Description:** This notification informs the applications about the presence of an incoming message from a remote node.
- **When generated:** This method is invoked by CAMEO when it receives a message for a specific application from a remote node.
- **Parameters returned by CAMEO:** a *LocalMessage* containing the content of the message and the IP address of the sender.

5. Conclusions

In this report we presented CAMEO technical specifications, highlighting requirements and procedures to implement efficient MSN applications exploiting context- and social-awareness features. CAMEO is able to collect, manage and reason upon multidimensional context information, derived both from physical and virtual worlds, characterizing the user's profile, her social behavior, the available services and resources and the surrounding environmental conditions. Several application domains can benefit from the analysis and correlation of this context information, contributing to the general well-being condition of users and their society. Results presented in [1] showed the efficiency of CAMEO in collecting and managing well-being context and supporting the development of MSN applications through real experiments. Currently we are extending CAMEO in several directions, from the efficient management of heterogeneous context information (mainly related to heterogeneous sensing devices) up to the implementation of services based on the emerging paradigm of *opportunistic computing*. We are also designing efficient reasoning techniques to make the system able to provide personalized and situation-aware feedback to applications and final users.

References

[1] V. Arnaboldi, M. Conti, F. Delmastro, Cameo: a novel context-aware middleware for opportunistic mobile social networks, Pervasive and Mobile Computingdoi:http://dx.doi.org/10.1016/j.pmj.2013.09.010.

[2] G. Percivall, C. Reed, J. Davidson, Open Geospatial Consortium Inc . Sensor Web Enablement : Overview And High Level Architecture . (2007).

[3] V. Arnaboldi, M. Conti, F. Delmastro, G. Minutiello, L. Ricci, Sensor Mobile Enablement (SME): a Light-Weight Standard for Opportunistic Sensing Services, in: International Workshop on the Impact of Human Mobility in Pervasive Systems and Applications, 2013.

[4] C. Boldrini, M. Conti, A. Passarella, Exploiting users social relations to forward data in opportunistic networks: The HiBOp solution, Pervasive and Mobile Computing 4 (5) (2008) 633–657.

[5] Android developer guide.
URL `http://developer.android.com/guide/topics/fundamentals.html`

[6] Java corba idl.
URL `http://www.omg.org/spec/I2JAV/1.3`

[7] C++ corba idl.
URL `http://www.omg.org/spec/CPP/1.2`

[8] Wifi alliance: Wifi direct specifications.
URL `http://www.wi-fi.org/discover-and-learn/wi-fi-direct`

[9] V. Arnaboldi, M. Conti, F. Delmastro, Implementation of CAMEO : a Context-Aware MiddlEware for Opportunistic Mobile Social Networks, in: 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM), BEST DEMO AWARD, 2011.