ABSTRACT

Next generation sensing applications will be large-scale with multiple human actors, including: administrators, application developers, participatory sensing volunteers and consumers of sensor data. In these sensing scenarios, both human actors and software artefacts should be reconfigurable at runtime. A middleware solution for this class of sensing application should provide a relationship amongst equals, wherein human actors and software artefacts may be explicitly configured and reconfigured using a single consistent development model. In recent years, Online Social Networks have proven to be a successful and popular mechanism through which millions of people have created digital representations of themselves and their relationships. We combine a reconfigurable component model with online social networks to create a middleware that allows software and human actors to be composed into reconfigurable distributed applications. This allows developers to create dynamic sensing applications with the human explicitly in the loop.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems, Distributed Applications

General Terms
Management, Human Factors, Languages.

Keywords
Sensor Networks, Online Social Networks, Component-Based Software Engineering.

1. INTRODUCTION

Human actors are becoming an increasingly important element of large-scale sensing systems. This may include the consumers of sensor data, sensor network administrators, application developers and participatory sensing volunteers. These users may access the system using a range of static and mobile computing devices. Users may also be dynamically added to, or removed from application compositions and user roles may change over time. The emerging challenge for sensor network middleware is to seamlessly integrate users into distributed, reconfigurable sensing applications.

While a number of middleware platforms exist for developing WSN applications [8] [6], these platforms do not treat the user as a first-class entity in the sensing application and provide no specific abstractions for modelling users as a system resource. Conversely, middleware for mobile application development [11] [10] provides mechanisms to realise user-friendly local software for mobile devices. However, this class of middleware typically does not support the creation of distributed applications running across multiple mobile devices. To the best of our knowledge, there is no unifying middleware platform that allows sensor nodes and mobile devices to be composed into coherent distributed applications.

In parallel with the maturation of WSN technologies, Online Social Networks (OSNs) [5] [2] have gained popularity in recent years as a mechanism through which users may create digital representations of themselves and their relationships. OSNs provide a number of explicit interactions that provide communication between users as well as the creation and maintenance of explicit links between users. OSNs are platform and device independent and users may interact with them over the web, through mobile applications, email or text message. In our view, OSNs provide an ideal platform for providing seamless interaction between sensing applications and users. Thus, both software modules and human actors would interact via OSNs.

In this position paper, we introduce @LooCI, an extension of the LooCI [8] component model for WSN. @LooCI provides services that allow users to be modelled as standard software components. As both users and software are modelled using a single approach, the standard LooCI API may be used to compose distributed sensing applications that include both software artefacts and humans. We achieve this by wrapping each user’s OSN presence as a standard LooCI component and extending the LooCI binding model such that messages may be routed between IP-based sensor nodes and OSNs. The contributions of @LooCI are three-fold:

- @LooCI is the first middleware that provides a first class abstraction for human actors and allows for the composition of software with human actors.
The user-binding model of @LooCI allows transparent communication between software and human actors.

By using a combination of both IP networking and application-level networking via OSNs, @LooCI significantly increases user availability.

The remainder of this paper is structured as follows: Section 2 provides a motivating scenario that demonstrates the need for humans as first class reconfigurable entities in sensing applications. Section 3 provides an overview of the LooCI component model. Section 4 maps the LooCI API on to the Facebook [2] and Twitter [5] social networks. Section 5 describes the design of @LooCI. Section 6 evaluates an implementation of @LooCI for Twitter. Finally, Section 7 concludes and discusses promising directions for future work.

2. MOTIVATING SCENARIO

We use the example of a smart audience measurement system to motivate our work. In our example system, people don’t need a special-purpose device to provide information (such as a ‘people meter’ [3]) and they can instead use their smartphone, tablet or PC to provide real time data. Embedded sensor nodes located in the television provide additional context data. The data gathered from persons watching a specific radio or TV channel together with data from embedded sensor nodes, is analysed by an associated software element that determines the ratings for each channel, and uses this information to bill advertisers for broadcasted commercials based upon their audience. Considering this motivating scenario, it can be seen that there are three key requirements for a supporting middleware platform:

1. Integration of heterogeneous hardware, OS and network technologies (from motes to PCs).
2. Support for the dynamic composition of software elements with humans.
3. Provision of mechanisms to provide seamless connectivity to users who use multiple devices.

As we will discuss in Section 3, the LooCI middleware provides an excellent platform to manage heterogeneous hardware, OS and network technologies, fulfilling requirement 1. @LooCI additionally contributes support for the composition of sensors and users, and the enhancement of user’s availability by using OSNs, thus meeting requirements 2 and 3.

3. THE LOOCI COMPONENT MODEL

The Loosely-coupled Component Infrastructure (LooCI) is a runtime reconfigurable middleware for WSN. LooCI is composed of a platform-independent execution environment, a reconfigurable component model and an event-based binding model. These are briefly described in Section 3.1 to 3.3 respectively. A full description of the LooCI is provided in [8] and [9].

3.1 Execution Environment

The LooCI execution environment is composed of a Network Framework, Event Manager and Reconfiguration Manager as shown in Figure 1.

Figure 1: The LooCI middleware core

LooCI provides interoperability across various underlying platforms. At the time of writing LooCI supports Contiki [7], Squawk [4] and Android [1]. The Network Framework standardizes the networking services offered by the underlying platform and offers a uniform API to the upper layers. The Event Manager implements an event bus to which every component and all Reconfiguration Manager modules are connected. Event bus communication follows a decentralized topic-based publish-subscribe model. The Reconfiguration Manager maintains references to all local components and enacts incoming deployment, control, introspection and binding commands that are received over the event bus.

3.2 Reconfigurable Component Model

The LooCI component model allows developers to implement components in various languages (C, Java SE and Java ME) and for different operating environments (Contiki, Squawk, OSGi and Android). LooCI may therefore be viewed as an interoperability layer that allows the application developer to compose together software resources that may be distributed across heterogeneous nodes. LooCI components implement fine-grained application level functionality and can be composed together into a distributed application via bindings over the event bus. For this purpose, each component includes a number of interfaces. Provided interfaces describe the events a component may publish to the bus and required interfaces describe events that a component may read from the bus.

3.3 Event-based Binding Model

The LooCI event-bus is an asynchronous, event based communication medium. All data in LooCI is semantically typed and all communication occurs via explicit bindings. In general, bindings are defined by a <source event type, source component ID, source address, destination event type, destination component ID, destination address> tuple. LooCI supports a rich set of binding modalities including:

- A one-to-one binding is implemented by sending a `wireTo` event to the source node that identifies a unique provided-interface and a remote destination. A `wireFrom` event is sent to the destination node that identifies the unique source interface and the local destination interface.
- A one-to-many binding is enacted by sending a `wireTo` event to the source node that identifies a unique...
provided-interface and a wild-card destination (network broadcast or one-hop broadcast). A **wireFrom** event is then broadcast to all destination nodes that specifies the source address, a unique source interface and the local destination interface. A many-to-one **binding** is established using the inverse set of operations.

- **Opportunistic bindings** are enacted by sending a **wireTo** event to all source nodes that specifies a provided interface type and the 1-hop broadcast wildcard. A **wireFrom** event is then sent to the destination node(s) that specifies a wild-card source address, a wildcard source component id, the source interface type and a local destination interface.

LooCI bindings are established by explicitly connecting compatible interfaces using the operations described in Listing 1 [9]. The event bus is agnostic to underlying network protocols, and thus can be used on top of IPv6 or over Twitter, as will be explored in Section 5.

### Listing 1: The core LooCI binding API

```
Boolean wireFrom (EventTypeOut, SrcCompID, SrcNodeID, DestCompID, EventIn, DestNodeID)
Boolean wireTo (EventTypeOut, SrcCompID, SrcNodeID, DestCompID, DestNodeID)
Boolean wireFromAll (EventTypeOut, EventIn, DestCompID, DestNodeID)
Boolean wireToAll (EventTypeOut, SrcCompID, SrcNodeID)
```

4. **MAPPING THE LOOCI API TO OSNS**

To allow transparent bindings between humans and software artefacts, matching schemas between the LooCI binding model and OSN interaction models are defined. This guarantees that the complete set of modalities is supported. In this paper, we focus on the two most widely used OSNs: Facebook [2] and Twitter [5]. It should be noted that both software modules and humans will interact with the sensing application via OSNs. In either case, OSNs provide a device independent communication mechanism and provide an application layer network that mitigates mobility issues and connectivity issues arising from the use of NAT.

4.1 **Twitter**

A twitter account is assigned to every physical sensor node and human actor. The matching schema for Twitter is shown in Table 1. It can be seen that Twitter provides inherent support for one-to-one bindings without extension. To simulate broadcast in Twitter, we provide a specialized broadcast Twitter account that follows all @LooCI Twitter accounts and is also followed by those accounts. The broadcast account uses the *retweet* action (i.e. resending a tweet from a followed user) to implement broadcast. Therefore, (1) a user that follows the broadcasting account will receive all broadcasted information, and (2) a user that is followed by the broadcasting account can broadcast information.

Twitter has two inherent limitations as a communication medium. For each Twitter ID, only 350 tweets can be received per hour and only 1000 tweets per day may be published. Therefore, a node can publish a tweet every 86.4 seconds and can receive a tweet each 10.29 seconds. Considering that a tweet can only contain 140 bytes of information, this gives rise to an upstream bandwidth of 12.96 bps and a downstream bandwidth of 108.89 bps (where n is the number of recipients). An initial evaluation of the Quality of Service (QoS) that can be provided by Twitter is provided in Section 6.

4.2 **Facebook**

As with Twitter, every physical sensor node and human actor is assigned a Facebook account. Table 2 proposes a matching schema for the Facebook OSN. To simulate broadcast in Facebook communications, we build on two Facebook concepts: subscriptions and the ‘Like’ feature and provide a specialized Broadcasting Facebook account. The broadcasting Facebook account automatically likes any publication that any friend makes to its own wall. In addition, this account allows any friend to publish messages to its wall, without any visibility restrictions. Therefore, (1) a user that is subscribed to the broadcasting account will receive all broadcasted information, and (2) a user that is a friend of the broadcasting account can broadcast information.

Unlike Twitter, Facebook imposes no specific rate limit on API calls. A key area of future work will be to empirically evaluate the QoS that can be achieved on Facebook.

5. **THE DESIGN OF @LOOCI**

@LooCI extends the LooCI middleware by incorporating a mobility framework in order to support the requirements identified in Section 2. Section 5.1 describes the Mobility Framework and Section 5.2 discusses how human actors are treated as reconfigurable components by the @LooCI middleware.

5.1 **Mobility Framework**

The mobility framework is placed between the Network Framework (layer belonging to LooCI middleware core) and the Underlying Platform. As shown in Figure 2, it is composed of a Communication Dispatcher, a number of OSN modules and an Addressing System. These are described in Section 5.1.1 to 5.1.3 respectively.

5.1.1 **The Communication Dispatcher**

The Communication Dispatcher is the middleware element that dispatches outgoing messages via either the IP network or various OSN plugins. Where a software element is connected via both IP and OSN plugins, the most optimal communication channel will be automatically selected and the message will be dispatched to the appropriate plugin or via the network stack. The communication dispatcher also provides translation between the header format of IP messages and OSN messages. For example by incorporating human readable ‘mark up’ elements into messages that will be dispatched using OSN modules.

5.1.2 **OSN Modules**

OSN modules like Twitter Module or Facebook Module are responsible for the communications from and to the social network by using the public APIs that allow the access to
them (e.g. Twitter Module implements the REST API offered by Twitter). Therefore, these modules require data connections with the Communication Dispatcher and the Underlying Platform. In addition, these modules also require a control connection with the IPv6 Address/OSN ID Addresses Management element. So that element can use the modules to perform required actions (e.g. send a friendship request in Facebook or follow a user in Twitter). These modules all implement the standard LooCI network manager API, allowing for seamless use of both IP and OSN-based bindings for components.

### 5.1.3 Addressing

The addressing element of the Mobility Framework is composed of an OSN ID Discovery element and an Address Management Element:

**OSN ID Discovery:** This element is responsible for the discovery of OSN accounts that represents a node based upon its IPv6 address. For this purpose, these elements exchange some control messages with remote nodes. The connection with the IPv6 Address/OSN ID Management element is used to report the results of the discovery process.

**Address Management:** This element stores a matching

<table>
<thead>
<tr>
<th>Binding modality</th>
<th>LooCI commands</th>
<th>Twitter actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>one (A) to one (B)</td>
<td><code>wireFrom (EventTypeOut, SrcCompID, A, EventTypeIn, DestCompID, B)</code></td>
<td>B sends a following request to A. Once it is accepted, B follows A. So A can send direct messages to B.</td>
</tr>
<tr>
<td></td>
<td><code>wireTo (EventTypeOut, SrcCompID, A, B)</code></td>
<td>A accepts the following request from B.</td>
</tr>
<tr>
<td>many (U) to one (D)</td>
<td><code>wireFromAll (EventTypeOut, EventTypeIn, DestCompID, D)</code></td>
<td>C (Twitter account that retweets all tweets from followed accounts). D follows C.</td>
</tr>
<tr>
<td></td>
<td><code>wireTo(EventTypeOut, SrcCompID, U, D)</code></td>
<td>As there is no previous following request from D, C follows U.</td>
</tr>
<tr>
<td>one (E) to many (W)</td>
<td><code>wireFrom (EventTypeOut, SrcCompID, E, EventTypeIn, DestCompID, W)</code></td>
<td>W sends a following request to E. As there is no answer, W follows C.</td>
</tr>
<tr>
<td></td>
<td><code>wireToAll (EventTypeOut, SrcCompID, E)</code></td>
<td>C follows E.</td>
</tr>
<tr>
<td>opportunistic bindings</td>
<td><code>wireFromAll (EventTypeOut, EventTypeIn, DestCompID, DestNodeID)</code></td>
<td>DestNodeID follows C.</td>
</tr>
<tr>
<td></td>
<td><code>wireToAll (EventTypeOut, SrcCompID, SrcNodeID)</code></td>
<td>C follows SrcNodeID.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binding modality</th>
<th>LooCI commands</th>
<th>Facebook actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>one (A) to one (B)</td>
<td><code>wireFrom (EventTypeOut, SrcCompID, A, EventTypeIn, DestCompID, B)</code></td>
<td>B sends a friendship request to A. Once it is accepted, A can publish message at B’s wall.</td>
</tr>
<tr>
<td></td>
<td><code>wireTo (EventTypeOut, SrcCompID, A, B)</code></td>
<td>A accepts the friendship request from B.</td>
</tr>
<tr>
<td>many (U) to one (D)</td>
<td><code>wireFromAll (EventTypeOut, EventTypeIn, DestCompID, D)</code></td>
<td>C (Facebook account that <em>likes</em> every message anyone publishes in its wall). D subscribes to C.</td>
</tr>
<tr>
<td></td>
<td><code>wireTo(EventTypeOut, SrcCompID, U, D)</code></td>
<td>As there is no previous friendship request from D, U sends a friendship request to C. C follows U.</td>
</tr>
<tr>
<td>one (E) to many (W)</td>
<td><code>wireFrom (EventTypeOut, SrcCompID, E, EventTypeIn, DestCompID, W)</code></td>
<td>W sends a friendship request to E. As there is no answer, W subscribes to C.</td>
</tr>
<tr>
<td></td>
<td><code>wireToAll (EventTypeOut, SrcCompID, E)</code></td>
<td>E sends a friendship request to C. C accepts the request, so E can publish in C’s wall.</td>
</tr>
<tr>
<td>opportunistic bindings</td>
<td><code>wireFromAll (EventTypeOut, EventTypeIn, DestCompID, DestNodeID)</code></td>
<td>DestNodeID subscribes to C.</td>
</tr>
<tr>
<td></td>
<td><code>wireToAll (EventTypeOut, SrcCompID, SrcNodeID)</code></td>
<td>SrcNodeID sends a friendship request to C. C accepts the request, so SrcNodeID can publish in C’s wall.</td>
</tr>
</tbody>
</table>
table between every IPv6 address and related OSNs that can be used to connect to the same node or human actor. The Address Management element, provides control information to the Communication Dispatcher each time it is requested, for example, resolving an OSN ID that matches an IPv6 address. Nevertheless, this element needs to be provided with control information related with the component model (e.g. received or sent wireTo command, wireFrom command…) to perform the subsequence action at OSNs. Such actions are reported by the control connection from this element to OSNs modules.

Figure 2: Mobility Framework that supports @LooCI features

Finally, this framework makes completely transparent to other layers the use of OSNs. Even more, it keeps LooCI core middleware completely isolated since it only requires some information provided by the Reconfiguration Manager that does not affect to the core. In addition, since it is possible to add new modules that support the communication with new OSNs, this framework can be easily scaled.

5.2 Human Actors as Reconfigurable Components

Basing on the @LooCI framework, it is indeed possible to treat human actors and LooCI components as equivalent conceptual entities. To illustrate this, we revisit the audience monitoring example from Section 2. The requirements on the human actors are: (1) to have an account with an OSN supported by LooCI and (2) to be able to access this OSN from the various devices they wish to use. If users wish to host sensing software artefacts on their device it is also necessary for them to install the @LooCI middleware. Remote LooCI components can interact with users and software artefacts running on user devices using the standard LooCI binding and communication API.

For example, the composition shown at Figure 3 allows our system to measure the number of people watching a specific TV channel. LooCI binding API (wireTo command, wireFrom command…) allow us to establish the shown composition including both software artefacts (Component) and users (User A and User B). LooCI communication API allows the component to request information from users (by publishing an event through the interface) and receive their answers (as events though its corresponding receptacles). And OSNs allow users to receive the request and to send the answer. In addition, LooCI binding API also allow us to dynamically connect these users to other components.

Figure 3: Composition example with users and components

6. EVALUATION

In this section, we provide an initial evaluation of the availability and latency of social networks, in order to demonstrate the feasibility of connecting users to sensing applications using OSNs. We also investigate the overhead of running @LooCI on a smartphone. It should be noted that for embedded sensor motes, OSN functionality is implemented on the gateway and thus imposes no overhead on embedded motes.

6.1 Experiment

We performed a real world trace of network performance for 4 users of mobile devices over 3 days using Android mobile phones. Each mobile device had its own Twitter account and ran an evaluation programme that attempted to send a tweet and an UDP packet at regular intervals, which were used to measure latency and availability. The experiment server polled each Twitter account regularly in order to obtain new tweets and extract device availability and latency.

6.2 Results

Table 3 shows the availability of each user during the experiment. That table also highlights the percentage of that availability when Twitter was the only communication medium available. As can be seen from Table 3, Twitter access provides at least the 50% of the availability in all users, due to the extensive use of NAT and firewalls. Thus, the use of Twitter significantly increases the availability of all users.

Table 3: Availability per user

<table>
<thead>
<tr>
<th></th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>75.10%</td>
<td>95.57%</td>
<td>75.31%</td>
<td>92.49%</td>
</tr>
<tr>
<td>Twitter exclusively</td>
<td>66.38%</td>
<td>59.75%</td>
<td>51.56%</td>
<td>92.49%</td>
</tr>
</tbody>
</table>

Table 4 shows the observed latency results when the mobile device uses Twitter. As it can be seen, although the latency is high compared with other protocols (e.g. latency
in UDP is always around a milliseconds), it is well within the timeliness requirements of typical sensor network applications.

<table>
<thead>
<tr>
<th>User</th>
<th>Average</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>27,119 s</td>
<td>15,117 s</td>
</tr>
<tr>
<td>User 2</td>
<td>27,605 s</td>
<td>14,398 s</td>
</tr>
<tr>
<td>User 3</td>
<td>28,962 s</td>
<td>16,442 s</td>
</tr>
<tr>
<td>User 4</td>
<td>25,091 s</td>
<td>15,124 s</td>
</tr>
</tbody>
</table>

We have evaluated the overhead of running the @LooCI framework on mobile devices. Table 5 shows the memory consumption of @LooCI. These figures are well within the capabilities of modern smart phones.

<table>
<thead>
<tr>
<th>Device Relative</th>
<th>Absolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 0,15 %</td>
<td>~ 800 kB</td>
</tr>
<tr>
<td>~ 12,50 %</td>
<td>~ 32 MB</td>
</tr>
</tbody>
</table>

This evaluation is limited by the number of users involved (4 users) and by the duration of the experiment (3 days). However, the presented results hint that OSNs are a promising medium through which users can be connected to sensing applications, offering significantly higher availability than UDP-based communication.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed to create a large-scale sensing infrastructure by incorporating humans into distributed sensing applications. In this heterogeneous environment, with different hardware platforms and network technologies, using a common middleware provides major benefits such as consistent development, deployment and management support, including reconfiguration at runtime. However, this middleware should support a relationship amongst equals, wherein human actors and software artefacts may be explicitly configured and reconfigured using a single consistent development model. Our solution is to use social network technologies to accomplish global reach and widespread availability for users and to allow users to be explicitly specified as resources in distributed sensing applications. To achieve this, we have defined a unifying middleware, called @LooCI, that supports the execution of dynamic distributed sensing applications on a mixed infrastructure of fixed sensor networks, consumer mobile devices and humans. Our initial evaluation results show that using OSNs such as Twitter significantly increases the availability of users as part of the large-scale sensing infrastructure. By selecting the most optimal communication medium, @LooCI provides optimal availability using the best performing communications medium.

7.1 Future Work

The use of OSN-based bindings as a part of a reconfigurable component-based middleware raises several exciting challenges. One challenge is to explore new trade-off mechanisms that maximize users’ availability using multiple communication channels at the application and network level, while minimizing latency and packet loss. A second challenge is to analyse how more advanced mechanisms of Twitter like hashtagging can be used to emulate a distributed event bus with only one Twitter account. The last challenge is to explore Human-Machine Interfaces techniques that contribute with a better user experience.

8. ACKNOWLEDGMENTS

This research is partially funded by the Inter-university Attraction Poles Programme of the Belgian Science Policy, the Flemish agency for Innovation by Science and Technology (IWT Vlaanderen), and the Research Fund, KU Leuven. It is conducted in the context of the IWT-SBO-STADiUM project No. 80037 and IWT-SBO-SymbioNets project No. 090062.

9. REFERENCES