ROBIN: Activity Based Robot Management System

Benjamin Hadorn\textsuperscript{1} Adriana Wilde\textsuperscript{2}

19 June 2009

Department of Informatics - Master Course Project Report

1benjamin.hadorn@unifr.ch, PI09-group2, PAI group, DIUF, University of Fribourg
2adrianagabriela.wilde@unifr.ch, PI09-group2, PAI group, DIUF, University of Fribourg
## Contents

1 Introduction  
  1.1 Project goals ........................................ 3  
  1.2 Organisation of the report .............................. 4  

2 Project management ........................................ 5  

3 General concepts  
  3.1 Entities, context, activities, situations, implicit interactions ........................................ 7  
  3.2 Robot as an entity ....................................... 7  
  3.3 Fire as context ......................................... 9  

4 Systemic analysis of the application  
  4.1 System overview ......................................... 9  
  4.2 Situation Manager ....................................... 9  
    4.2.1 Logic of Situation Manager .......................... 10  
  4.3 Entity Tracker .......................................... 12  
  4.4 Robin controller ....................................... 12  

5 Implementation section  
  5.1 Technical description of the components ............... 13  
    5.1.1 uMove ............................................. 13  
    5.1.2 Sensors ............................................. 14  
    5.1.3 Robots .............................................. 15  
    5.1.4 Portable devices .................................... 15  
  5.2 User’s guide ............................................. 15  
    5.2.1 System Controller ................................... 15  
    5.2.2 Setting up the mobile device ....................... 17  
  5.3 Tests and evaluation .................................... 21  
    5.3.1 Checking the data propagation ...................... 21  
    5.3.2 Testing the too_long_in criteria .................... 21  
    5.3.3 Changing situation if the fireman runs .......... 21  

6 Future work ................................................. 21  
  6.1 Changes to uMove Framework ............................. 21  
  6.2 Changes to Robin ....................................... 22  
  6.3 Personal Statements and Conclusions .................... 22  
    6.3.1 Benjamin Hadorn .................................... 22  
    6.3.2 Adriana Wilde ...................................... 22  

A Appendix ................................................... 26  
  A.1 Situation Manager ....................................... 26  
  A.2 Changes in MessageSender Class ....................... 29  
  A.3 Changes in Observer Class ............................. 30  
  A.4 Changes in View Class .................................. 31  
  A.5 Changes in Widget Class ............................... 32  
  A.6 Changes in TemperatureWidget Class ................... 33
Abstract

This report examines the design and implementation of a system to support firefighting exercises based on tools and ideas of pervasive computing. Through the development of this project, students became familiar with a range of technologies, such as sensors, wireless communication, and the programming framework uMove; as well as concepts such as context-awareness, situation management, and activity-awareness which were encountered in the Pervasive Intelligence course of the Masters programme in Computing Science.

In the development of this project, the notions of implicit human-computer interaction and user-centered design were of particular relevance. As an important part of the design cycle, testing was planned and performed not only for the detection and correction of errors within individual modules of the project but to give a way towards a final integration. Details of project management issues are also included, because of their relevance on the development process.

The use of a programming framework for interaction through motion was central, and the present project became one of its proofs-of-concept, having presented opportunities to improve it and recommendations to adapt it for future applications.

Keywords: context-awareness, activity-based systems, situation management, uMove, KUI model, implicit interactions, wireless communication, sensors, robots, Sun SPOTS, Lego NXT, testing
1 Introduction

Firefighting involves the work of skilled personnel who are regularly required to make quick yet important decisions based on rapidly changing situations. They must make these decisions while performing strenuous physical activities using heavy equipment and uncomfortable protective clothings under life threatening conditions. To illustrate the degree of danger involved in this profession, we have found that from 1995 to 2007 there were 1,345 on-duty firefighter fatalities in the United States alone [4], making this job one of the most dangerous of our times. Furthermore, research shows that the highest mortality factor in the profession is over-exertion or stress leading to on-the-job heart attacks, which account for 45% of the total deaths of US firefighters, many of which occur while performing non-emergency duties [13].

Among the many hazards firefighters face which might account for such elevated levels of stress, are chemical exposure, thermal injury and trauma, all of which potentially interfere with the assessment of the rapidly changing state of the situation during search and rescue operations, which are generally conducted in low-visibility, high-heat conditions.

The main objective during primary search operations is to move as quickly as possible through the structure while still being thorough, and although there is a great variety of equipment available to the firefighter to perform his duties, these hinder rather than facilitate the gathering of information about the state of the building. Specifically, we are referring to the inconvenience of handling such equipment because of its weight and volume, even though we acknowledge that this is essential for the purposes of fire extinguishing, protecting the firefighters and allowing access to potential casualties.

Semi-autonomous robots can help the work of the firefighters in collecting data and in fact their use in urban search and rescue operations is not novel as there is a wealth of research devoted to this in recent years, even more so after the terrorist attacks of September 11th, 2001 in the United States (for example see [6], [10], [18] and related works). This is the rationale and fundamental motivation for the Robin project.

1.1 Project goals

The main goal of the project was the development of a context-aware application which would be able to detect activity through motion [2]. In effect, the application would interpret such activities by sensing the motions of firefighters, and would use semi-autonomous robots as tools to collect environmental data that could inform the relevant person about their situation.

In principle, and depending on the firefighters’ movements within a building, the system would possibly control robots sent ahead of the team\(^1\) to gather information such as the state of the building (for example, temperature, the presence of smoke and/or dangerous gases in rooms explored) that might represent a potential physical danger for the rescue team. For example, we considered a scenario where a firefighter might be aware of an injured person trapped in a room in fire. By deploying a robot in advance to the room in question, the system has enough data to determine whether firefighters can proceed safely or whether there are dangers to be taken into account. This would prevent them from finding themselves in a critical situation unexpectedly, and by doing so, it would increase the knowledge about the incident, reducing the levels of stress and allowing them to perform the appropriate rescue operation.

Among the secondary goals of the project there was the reinforcement of theoretical concepts acquired during the lectures and recommended readings given in the course, by the use of the framework uMove, as well as familiarisation with a range of pervasive technologies that are commonplace in this area of research. Additionally, the practical application of project management skills required to bring forward such an ambitious project within the time limits while making use of the variety of available resources and students capabilities was another of the goals of the project.

Figure 1 on page 4 illustrates the various aspects to be taken into consideration in the development of this project: in red, the flow of information related to the detection of motion, via sensors

---

1Ideally via implicit interactions, so not to add to the already heavy cognitive workload of the firefighter. The robot is meant to be of help, not a hindrance. However, as we will describe later with more detail, it soon emerged that controlling the robots solely through implicit interactions was not possible.
(such as accelerometers), to ascertain the activity that the fireman\(^2\) performs at any given time. In dark blue: the information flow carrying contextual data (such as temperature, the presence of smoke, gases, etc), which will, together with the activity information, eventually inform the fireman about his current situation (in dashed blue lines) via a portable device. Finally, in black: the control of the robots.

It is also worth mentioning at this point that the layered system into which this information is either fed or produced, is the KUI\(^3\) model, which the uMove architecture implements, as we will describe with more detail in section 5.1.1.

1.2 Organisation of the report

The rest of this document is organised as follows: Section 2, explains how all various aspects of the project introduced above were divided into modular tasks that could be developed independently by separate groups of students, and how the completion of each assigned task was monitored and met by our group. Section 3 deals with important concepts in pervasive computing which form the theoretical background of the project, as well as our interpretation of them and how they were applied. Section 4 shows how the system works, describing each of the components. Under section 5, various implementation aspects are considered. Firstly, a description of the various technologies that were employed in Robin is given, including some technical considerations. Then an explanation of the operation of Robin is offered as a users’ guide, finishing with a report on the tests and evaluation of the project.

Finally, in section 6, future work and recommendations are offered with the conclusions of the

\(^2\) For simplicity, we will consider the terms firefighters and firemen as equivalent from now on, in the understanding that the actual gender of the firefighter is irrelevant for the discussion.

\(^3\) Kinetic User Interface
present study. Appendices to this report show the pseudo-formal algorithms to the main programs developed, as well as the modifications in the JAVA™ code of uMove.

2 Project management

As can be derived from the diagram in Figure 1 and its description, the project has three different aspects, namely: motion and activity detection from sensor data, context-based situation analysis and application reaction used to control the robots, and finally communication issues between robots and the application and between the application and the fireman.

With a well defined division of tasks, it was appropriate to distribute the responsibilities among the participants of the course, resulting in two groups of two or three students each, working independently but in close collaboration. This collaboration was facilitated by the support and guidance of two teaching assistants, who also worked on the maintainance of the programming framework (uMove). Weekly meetings were scheduled in order to follow the progression of each group and encourage discussions.

The formal presentation of the project proposal took place on the fourth week of the course after a sketch was informally put forward by some students the previous week. Although the primary goal of the project was essentially predetermined by the teaching team (in accordance with the objectives of the course and the research interests of the PAI group) the idea of using robots as an aid for firefighting was volunteered by the students. This was very important, because even though at such an early date the students lacked the theoretical background required and could not start the implementation immediately, the idea of developing a tool that had practical applicability as well as being technologically challenging motivated the students into studying design considerations and investigating the possible requirements that would have emerged in a real case scenario. According to Buckley et al’s research [5]:

students [show] a noticeable lack of attachment and interest when working on projects lacking applicability outside of the classroom... Students became primarily interested in getting a good grade, and they spent more energy anticipating the intent of the instructors (and hence, the grading criteria) than they did deriving real customer requirements... Students in [...] Computer Science and Engineering courses tend to think in terms of ‘coding a solution’ and it is very difficult to focus their energies on customer requirements and design. In our course, students are forced to focus on understanding customer requirements and developing a high-level design before they ‘think code’ because they do not inherently understand the problem and are unfamiliar with the customer’s environment. Thus in this instance it is hard to begin coding or implementing the solution prematurely.

Another feature of the ‘real-world’ which we were fortunate to recreate in our project was of a multidisciplinary approach. Even though we are all students on the same programme, there is a range of experiences, abilities and strengths available to the project management process, in which time constraints also played an important part. The work was distributed among the groups as illustrated in Figure 2.

Our group’s responsibilities included the situational analysis, that is to define the situations that the application takes into account based on a user’s detected activities and the context in which the action takes place and also the integration of this situation reasoning module within the application developed with uMove framework.

Another responsibility of our group was to ensure the appropriate feedback was provided to the users according to the detected situation, as well as the selection of an adequate communication protocol between the users and the application. And finally, our group had to design and implement an algorithm producing a command to the robot as a function of the fireman’s activity.

Individual responsibilities were assigned within our group as follows:

- Technical design and documentation: Adriana Wilde, Benjamin Hadorn

---

4Pascal Brügger and Daniel Ostojic
5on the 10th March 2009
Figure 2: Distribution of work among the students of Pervasive Intelligence

- Situation manager: Adriana Wilde
- Robin System Controller and communication to the mobile device: Benjamin Hadorn

In addition to the weekly meetings during lessons, further opportunities for collaboration with the other group were given at the synchronization points provided by the milestones detailed in Table 1. SVN and TRAC pages were set up so that the groups could disclose to the others their progress (or lack of) in achieving their milestones. Unfortunately this tool was underutilized. However, the Moodle forums were a valuable media for exchange of ideas while outside the classroom, and these were more frequently used by most participants.

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Description</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>Formal project proposal</td>
<td>Week 4</td>
</tr>
<tr>
<td>1.</td>
<td>Setting up TRAC with SVN</td>
<td>Week 10</td>
</tr>
<tr>
<td></td>
<td>First layout of the project</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Getting used to Sun SPOT, NXT and uMove</td>
<td>Week 14</td>
</tr>
<tr>
<td>3.</td>
<td>Situation Manager and Robin Controller</td>
<td>Week 16</td>
</tr>
<tr>
<td>4.</td>
<td>Project Workshop Lo-Fi testing</td>
<td>Week 21</td>
</tr>
<tr>
<td>5.</td>
<td>Project integration</td>
<td>Week 25</td>
</tr>
<tr>
<td>6.</td>
<td>Project report</td>
<td>Week 25</td>
</tr>
<tr>
<td>7.</td>
<td>Project presentation and demo</td>
<td>Week 26</td>
</tr>
</tbody>
</table>

Table 1: Project Milestones
3 General concepts

This section offers a brief discussion about the key definitions of pervasive computing as well as their application into the Robin project.

3.1 Entities, context, activities, situations, implicit interactions

There are many definitions of context, as surveyed by Ensing in [11]. This diversity is a consequence of the fact that this concept belongs to an area of concern that has separate origins, which Dourish explored extensively in [9]. We choose to conform to Dey and Abowd’s definition [7]:

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that can be considered relevant to the interaction between a user and an application, including the user and the application themselves.

These authors also define context-aware systems, as follows:

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s tasks.

The concepts of activity and situation are closely interrelated with this definition, moreover, what Dey and Abowd call ‘user’s tasks’ is the same as activities. According to Li and Landay [14]:

An activity evolves every time it is carried out in a particular situation. A situation involves a set of tasks or actions performed under certain conditions, e.g., locations, times or temperatures.

Activity-based ubiquitous computing affords an implicit interaction style. Interactions in activity-based computing are mainly based on input such as what people are doing as well as in what conditions (e.g., locations and times) an activity is conducted. These inputs are acquired implicitly (e.g., via sensors). This includes not only what a person is doing at the moment but also what people have done in the past and if possible, what people plan to do in the future.

Again, these authors call ‘conditions’ what we know as ‘contextual information’. Additionally, the idea of using future intentions as an input for the systems was particularly interesting to us. In practical terms, and as we have mentioned in previous sections, we expected the robot to be able to explore the conditions ahead of the fireman, using solely implicit interactions in this process. Nevertheless, during early stages of design it soon became evident that explicit interactions were required to input a firefighter’s plans. In other words, the fireman had to give explicit commands to the robot from time to time, for example, when intending to change directions, or go into a new room. This is not necessarily a limitation, since occasional explicit interactions do not make the system less context-aware. Furthermore, Li and Landay [14] affirm:

At the same time, activity-based computing does not exclude explicit or traditional interactions. Instead, explicit and implicit interactions can work together.

3.2 Robot as an entity

The first and most important point to make is that the robot is regarded as an entity. It is defined and handled as such in the current version of the Robin application. However, it was not always this way, as this was a very contentious issue at early stages of the design. The robot could have been seen as a mere sensor carrier, reading environment settings which are associated with the various rooms in the building, or even as an extension of the fireman to whom is assigned. This interpretation of the robot meant that there would not be involvement of robots in the situation analysis.

Eventually, and after great deliberation, it became clear that the robot should be considered as an entity, even if it has a special role. The reasons were two-fold: firstly, the robot can be
Figure 3: The fire is not an entity. The room, the robot and the fireman are entities.

seen as a "special kind of fireman", who is mobile and semi-autonomous, and its role is to help to gather information about potentially dangerous places. Secondly, being an entity its location can be determined easily and the sensor readings can be stored as contextual information to such a location.

This is a more elegant solution, more scalable as it is independent on the actual number of robots and firemen. In addition, it means that, being entities like firemen, robots can be in a situation (such as fully operational or damaged), which therefore can be incorporated in the situational analysis.

Moreover, Scholtz recommended in [18] that the subsystem controlling robots that take part in urban search and rescue operations needs to include a frame of reference to determine the position of the robot relative to environment (and therefore provide awareness of the robot’s surroundings). This information is easily supported by uMove for all entities. Other recommendations that Scholtz offers include:

- indicators of robot health/state;
- information from multiple sensors presented in an integrated fashion;
- the ability to self inspect the robot body for damage or entangled obstacles;
- automatic presentation of contextually-appropriate information⁶.

⁶An example given is to switch automatically to a rear camera view if the robot is backing up. Even so, we did not consider the use of cameras in our project beyond week 5 (after researching about available technologies, such as EyeWeb 4.0) because the conditions in which the robot would operate are typically of low visibility, so, the data gathered would probably not be of enough value to justify the investment in the technology.
3.3 Fire as context

On the other hand the fire is not an entity, despite being ‘mobile’ while it propagates. Unfortunately, fire is too difficult to characterize, as it behaves differently depending on the type of combustible materials, the amount of air available, the humidity, existence of draughts, and many other factors.

Nevertheless, its existence can be inferred in presence of high temperatures, therefore it can be characterized as contextual information of a given spatial location. We can look at fire as a value tuple:

\[
< \text{heat}, X, Y, Z > \equiv \text{Heat at the coordinate (X, Y, Z)}
\]  

(1)

In our project example we look at the heat as a context of the location, for which \((X, Y, Z)\) are spatial coordinates (like the centre of the room).

4 Systemic analysis of the application

4.1 System overview

The system consists of two major components.

1. Mobile device for the fireman. This device is always with the fireman and shows urgent messages to him, like “potentially dangerous” or “leave this room”.

2. System controller. It is outside of the building in which the incident takes place but is used to survey the area. It shows the present location and status (current situation) of both the firefighter and the robot.

The two components are linked through a socket connection (bluetooth). Both the remote visualizer and the visualizer on the system controller receive data through the Data Interface, as shown in Figure 4.

4.2 Situation Manager

To evaluate situations we had to consider 2 different types of sensor values.

1. The sensor values belong to the context of the entity A, as in Figure 5.
2. The sensor values belong to the context of the environment of entity A, as in Figure 6.

In the first case the value is stored within the entity A and a message is sent to the view and all its observers. Each observer is now able to evaluate the new situation of the entity A.

In the second case the value must be propagated to the location of the entity A, because it belongs to the context of the location.

Generally, for each change of the context in a location, the situation of all entities at that location must be reevaluated. For example, if the robot and the fireman are in the same room and the robot measures some dangerous gas in that room, the situation of the three entities (robot, fireman and location) might change, so it must be reevaluated.

4.2.1 Logic of Situation Manager

We designed the situation manager by establishing logical constructs which would determine, via predicates based on contextual information and activities performed by the entities, in which situation any given entity is, in a manner similar to that proposed by Loke [15]. Even though in his paper, Loke proposes situation programs based on sensor predicates, this does not exclude the consideration of activities into the predicate logic. In fact, hidden in a footnote, Loke provides an important clarification: although the terms ‘activity’ and ‘situation’ are distinct,

we consider activity as a type of contextual information which can be used to characterize the situation of a person.

Thus we can easily generalize to all entities capable of performing activities, i.e. including robots but not locations.

Based on our understanding of activities performed by firemen which could be identified with a simple motion detection system (such as the ones in [1] and [17]), we proposed originally five situations, each of which was a logical consequence of the evaluation of boolean values of three different origins: sensorial information (a boolean determining whether a given threshold has been
reached, e.g. too_hot), an activity being performed (e.g. NOT_in(L), which indicates whether the entity is in a given location L), or even relationships between entities (e.g. close_to(Robot)).  

- Exploratory Mode;
- Danger Awareness;
- In_Danger_Now;
- Possibly_Injured_Situation;
- Critical_Situation.

Soon a problem emerged with this definition, as it became technically (and even philosophically!) difficult to define a function of the distance between the fireman (for whom the situation applies) and the robot (which is sensing the danger in any of its presentations). Furthermore, this particular boolean value was critical in differentiating the situations Danger_Awareness and In_Danger_Now. We have now eliminated the need for such a function (close(Robot)) by rewriting these two situations. After all, all readings will be allocated to rooms, and all we need in order to trigger Danger_Awareness is to know whether there exists a location L for which any sensor reading has an abnormal value (and there are no firemen in L). In the case of there being a fireman F in L, he would be In_Danger_Now. So these situations became:

- Situation: Danger_Awareness
  - Actions associated: First encounter with danger. Inform firemen about probable danger in location L, in which currently there are no firemen. Proceed with caution. Localize danger with precision.
  - Activities supporting the situation:

\footnote{In Loke’s example was with_someone_now(john).}
\[(\text{too_hot}(L) \lor \text{too_cold}(L) \lor \text{gas}(L) \lor \text{smoke}(L)) \land \text{NOT_in}(L)\]

\text{NOT_in}(L) becomes true when the location of the entity (say a fireman) is different from \(L\).

- **Situation: In\_Danger\_Now**
  - Actions associated: The actor A (which could be a fireman or a robot) is in a location \(L\) in which a danger has been identified. Warn firemen about the type of danger, indicating time left before the situation becomes critical. Backtrack.
  - Activities supporting the situation:
    \[(\text{too_hot}(L) \lor \text{too_cold}(L) \lor \text{gas}(L) \lor \text{smoke}(L)) \land \text{NOT_too_long_in}(A, L)\]

- **Situation: Critical\_Situation**
  - Actions associated: The actor A (which could be a fireman or a robot) is in a location \(L\) in which a danger has been identified. Warn firemen about the type of danger, indicating time left before the situation becomes critical. Backtrack.
  - Activities supporting the situation:
    \[(\text{too_hot}(L) \lor \text{too_cold}(L) \lor \text{gas}(L) \lor \text{smoke}(L) ) \land (\text{too_long_in}(A, L))\]

Due to the technical limitations of establishing whether a fireman has fallen as required for the Possibly\_Injured\_Situation (see section 5.1.2), we decided to eliminate this situation from the analysis in order to simplify things. Another difficulty that arose was related to the determination of the spatial relation between entities (whether robot and fireman are in a given location), which was required for the Danger\_Aware situation, so it was also removed from the analysis. This left us with three situations, which we renamed: normal situation, potentially dangerous situation, and critical situation.

### 4.3 Entity Tracker

The entity tracker is used to receive situation alerts and activities. It sends the situation alerts to the system controller and to the mobile device of the fireman. The three different situation states are handled differently:

- **system controller**
  - normal situation: level blue
  - potentially dangerous situation: level orange
  - critical situation: level red

- **mobile device**
  - normal situation: no message
  - potentially dangerous situation: warning alert
  - critical situation: critical alert

### 4.4 Robin controller

The Robin controller manages the Robin activities. Since the robot is fully controlled by the system control software it is unnecessary to observe the activity of the robot entity by sensors in the way it is for the firemen’s activities. The way the controller operates is by generating commands to the robot, or interactions, which could be either implicit or explicit, in compliance with Li and Landay’s definition of activity based systems as presented in section 3.

The implicit interactions are automatically executed by the software depending on the situation of the robot, and the activity of the fireman. The decision rules devised are detailed in Table 2:
<table>
<thead>
<tr>
<th>Activity of fireman</th>
<th>Situation of robot</th>
<th>Command to robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being still (not moving)</td>
<td>normal</td>
<td>stop</td>
</tr>
<tr>
<td></td>
<td>potentially dangerous</td>
<td>stop</td>
</tr>
<tr>
<td></td>
<td>critical</td>
<td>retract</td>
</tr>
<tr>
<td>Crouching</td>
<td>normal</td>
<td>go forward</td>
</tr>
<tr>
<td></td>
<td>potentially dangerous</td>
<td>go forward</td>
</tr>
<tr>
<td></td>
<td>critical</td>
<td>retract</td>
</tr>
<tr>
<td>Running</td>
<td>normal</td>
<td>explore backwards</td>
</tr>
<tr>
<td></td>
<td>potentially dangerous</td>
<td>backtrack</td>
</tr>
<tr>
<td></td>
<td>critical</td>
<td>backtrack</td>
</tr>
<tr>
<td>Walking</td>
<td>normal</td>
<td>go forward</td>
</tr>
<tr>
<td></td>
<td>potentially dangerous</td>
<td>go forward</td>
</tr>
<tr>
<td></td>
<td>critical</td>
<td>backtrack</td>
</tr>
</tbody>
</table>

Table 2: Decision rules for implicit interactions within the Robin controller

<table>
<thead>
<tr>
<th>Priority command</th>
<th>Second order commands</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit stop</td>
<td>any other commands</td>
<td>the explicit stop command overrides all commands</td>
</tr>
<tr>
<td>any other explicit commands</td>
<td>any implicit commands</td>
<td>all explicit move commands override the implicit move commands</td>
</tr>
</tbody>
</table>

Table 3: Overriding of commands via explicit interactions

If the fireman is walking or crawling (moving around somehow), then the robot moves implicitly ahead of the fireman. If there is some danger ahead the robot should be able to continue moving on. If the situation gets to critical for the robot it should backtrack from the location.

If the fireman is not moving (standing still) the robot should wait for some explicit commands, like exploring. But if it becomes critical the robot must withdraw from the place (backtrack).

If the fireman is running (there is always a reason for running, even if the software does not know the reason) the robot should backtrack from its location. If the robot is not in danger it might explore on the way back.

Since the robot control can be used also for explicit commands there is a priority handling between both kinds of interactions.

This allows the fireman to control the robot even if the system fails to give relevant implicit commands to the robot for any reason.

5 Implementation section

5.1 Technical description of the components

As introduced in section 1, the Robin project uses the programming framework uMove, sensors (Sun SPOT), robots (Lego MindStorm NXT), and portable devices (Glofiish X500). In the following sections each of these will be considered in detail.

5.1.1 uMove

uMove is a JAVA™-based implementation of concepts of interaction through motion for pervasive systems as represented in the KUI model [3], offering to programmers a standard platform to develop KUI enabled applications. The framework is separated into layers in order to have a clear separation of concerns. The sensor layer contains all the widgets representing the logical abstraction of the sensors connected to the system. The entity layer contains the logical representation of the physical users or objects being observed. The activity manager aggregates the motion events into activities and makes them available to the observer. The contexts manager gets the
sensor information, updates the entities and sends the information to the *observation layer* which analyses the current situation of the entity.

5.1.2 Sensors

For motion detection, we used triaxial accelerometers, as their use is recognised as key to minimize human intervention in ubiquitous computing applications [17]. Specifically, we used Sun SPOTs (Sun Small Programmable Object Technology, [12]), which are small, wireless, battery-powered devices developed at Sun Labs. These devices can be used in a wide range of applications including robotics, environmental monitoring, asset tracking, proactive health care and many others, which makes them suitable for this project. Sun SPOTs are powered by a specially designed small-footprint Java virtual machine, called Squawk, that can host multiple applications concurrently, and requires no underlying operating system. Stackable boards include application-specific sensors and actuators such as accelerometers, light detectors, temperature sensors, LEDs, push buttons and general I/O pins. The devices can be duty cycled to run for months on a single charge of their rechargeable battery.

Once having determined the use of such sensors, we considered the following questions: how many Sun SPOTs should be placed on the fireman’s body to detect and discriminate his motion? Where should they be placed? How fine should the sampling be in order to identify the type of activity the fireman is performing? Is there a particular type of activity that could be difficult to identify?\(^8\)

The first study investigating performance of recognition algorithms is the one by Bao and Intille [1], with multiple, wire-free accelerometers on a range of activities. In this work, five accelerometers were used simultaneously to discriminate motion. However, these were only biaxial, and even so, ‘the recognition performance dropped only slightly’ when using just two, placed in thigh and wrist. The extra cost of providing more than double the amount of resources did not enrich the data significantly.

Ravi et al [17] place only one triaxial accelerometer (the CDXL04M3, by Crossbow Technologies), choosing to place it near the pelvic region. This idea seemed to be also applicable to Robin, given that placing it on a wrist, as considered at one stage, could introduce unwanted noise to the data (e.g. jerky movements with the hand might be interpreted as ‘running’ when in fact the fireman might be walking or just standing still). However, we had originally considered placing it on the wrist and we even discussed the possibility of introducing explicit gestures as motion commands even though it would not be ideal because the interactions should be implicit rather than explicit [16]. Furthermore, Ravi et al imply that ‘short activities’, such as arm gestures might be recognised using accelerometer data but with greater difficulty than general activities spanning over longer periods. Therefore, this solution, in addition to not being elegant, poses potential technical difficulties when it comes to interpret the data, so it was our strong recommendation to emulate Ravi et al, placing one Sun SPOT near the pelvic area of the fireman\(^9\).

The downfall of placing the accelerometer near the pelvis of the fireman is that activities that are limited to the movements of hands or mouth will be hard to recognise. Furthermore, Ravi et al [17] found that activities such as climbing stairs create nearly indistinguishable patterns regardless of the direction of the movement, which might seem counter-intuitive, as it would be logical to think that the z-axis acceleration is different going up than going down. However, their experiments showed that they were easily confused with other activities, particularly running and climbing stairs in the opposite direction. Activities such as standing, walking, and running are well classified even under suboptimal settings, but it is reasonable to expect that activities such as lying down or crouching would be difficult to discriminate against standing, because of the nature of the sampling process. Acceleration in all axes would be zero after the first few seconds, which Ravi et al [17] offer a comparison between a variety of classification algorithms known as base-level classifiers and meta-level classifiers. The ultimate decision on which to use, was again the responsibility of our colleagues so it escapes the scope of this report.

\(^{8}\)Another relevant question is which recognition algorithm should be use to treat the raw data? and Ravi et al [17] offer a comparison between a variety of classification algorithms known as base-level classifiers and meta-level classifiers. The ultimate decision on which to use, was again the responsibility of our colleagues so it escapes the scope of this report.

\(^{9}\)Nevertheless, as it was shown in section 2, the implementation of this part of the project was to be done by our colleagues in another group, and they decided instead to utilize two Sun SPOTs per fireman, one for the pelvic region and another on a thigh, which would produce less jerky data than on an arm.
<table>
<thead>
<tr>
<th>Situation level</th>
<th>Datagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal situation</td>
<td>(1)</td>
</tr>
<tr>
<td>Potentially dangerous situation</td>
<td>(2)</td>
</tr>
<tr>
<td>Critical situation</td>
<td>(3)</td>
</tr>
</tbody>
</table>

Table 4: UDP datagram coding

*al discard anyway ‘to minimize mislabeling’, and is precisely within these few seconds when the change of state occurs (from standing to lying down or crouching, and viceversa).

However, it is worth noticing that the Sun SPOT has an accelerometer capable of a much finer sampling than the CDXL04M3, so it would not be unreasonable to expect that the classification problems that Ravi et al found with the mentioned activities might be resolved simply by increasing the sampling rate\textsuperscript{10}.

5.1.3 Robots

Lego MindStorm [NXT] allows programmers to create robots that are equipped with various sensors (the most relevant to Robin are accelerometer, proximity detection, light). It would have been controlled by the application.

5.1.4 Portable devices

As a mobile device the Glofish X500 is used. The application runs on a Java Virtual Machine designed for Windows CE called Mysaifu. Since the JVM of Mysaifu has some limitations on the graphic and communication libraries, we’ve chosen AWT for drawing a simple graphic user interface showing the alerts. The communication is performed via Wi-Fi (Layer 1) and IP/UDP as Layer 3 and 4. Since UDP (User Datagram Protocol) is a message oriented communication protocol each reevaluation of the fireman-situation is sent to the device. This ensures that package loss doesn’t lead to wrong system behavior, in case of a missed situation alert, which is likely, as this very simple protocol does not guarantee data integrity. Each situation level is translated into a simple datagram according to the Table 4 which is then sent as shown in the Figure 7

5.2 User’s guide

5.2.1 System Controller

The system controller shows all entities, represented as coloured dots in the GUI, within the correct locations. The colour of the entities shows their current situation:

- normal situation: blue
- potentially dangerous situation: orange
- critical situation: red

In Figure 8, the robot and fireman are in different rooms, and the robot has detected some danger. If the entities are in the same location, as shown in Figure 9, both situations might change depending on the reading. To the right handside, a tree shows the current entity space configuration.

For simulation purposes a virtual sensor console was designed, a snapshot of it is shown in Figure 10, containing all important sensors for testing the functionality. The virtual robot contains three sensors: the first slider allows for variations of the temperature (as if measured by the robot), the

\textsuperscript{10}These authors sampled at 50 Hz, extracting features using a window size of 256 samples, with each window representing data for 5.12 seconds. With the Sun SPOTs we could sample to a frequency of up to 160Hz. This means taking a set of readings every 3.125 milliseconds, ‘which is a rate that can be sustained and still allow for other computation to be done - including real-time analysis of the incoming accelerometer values’ [12]. However, Goldman suggests that for gesture recognition ‘a sample rate of 10-20 readings per second will usually suffice’ [12], while for measuring motion of a SPOT mounted in a toy car ‘a rate of 100 readings per second was more than adequate’.
Figure 7: Communication to the mobile device using UDP/IP

Figure 8: Robot and fireman are in different rooms, and the robot has detected some danger
second simulates the smoke intensity, and the third slider the intensity of the methane gas. To simulate the activity of the fireman the following combobox allows the user to select the ‘detected’ motion type of the fireman. This is basically the output of a motion detection sensor. To simulate the movement of the fireman and the robot the next two comboboxes let you select the current location of the entity.

5.2.2 Setting up the mobile device

Before working with the mobile device, both the bluetooth and Wi-Fi have to be activated on the PC and portable device respectively, as shown in Figure 11. The mobile device needs the Fire-ManDev.jar file, which can be downloaded over bluetooth. It will be put into the My Documents folder, as shown in Figure 12. It is important that the old file has to be deleted first on the device. Then Mysaifu VM has to be started out of the program menu. This is a java virtual machine for Windows CE. The jar file must be selected and executed.

When the program runs it shows the current IP address of Wi-Fi on the bottom line. Before running system controller hestia it is important to make sure that the IP address is set correctly in the main.java file. It should contain the following lines:

```java
public class Main
{
    protected static void initSystem() throws Exception
    {
        //IP address of the mobile device
        String m_strMobileIP = "192.168.0.102";
        ...
    }
}
```

To test start the system control and change the sensor values. It should show following screens on the mobile device when changing to different situations, as shown in Figures 12, 12 and 12.
Figure 10: Virtual Sensor Console

Figure 11: Bluetooth and Wi-Fi setup. Both must be activated
Figure 12: Selecting the FireManDev.jar file

Figure 13: Normal situation: Green mode. No message needs to be given to the fireman
Figure 14: Possibly Dangerous situation: Orange mode

Figure 15: Critical situation: red mode
5.3 Tests and evaluation

During the development of the project both groups were able to test each component under development. Two formal opportunities were created to perform evaluation of the whole project. One, at week 21 and another one at week 25, as mentioned in section 2. The first evaluation was Lo-Fi, and each group had the chance to subject their algorithms to a test scenario devised by the teaching team. This enabled us to detect some logical errors, potential problems, and also enabled us to think about issues not yet considered, such as, in our case, the sensitivity of the system to appropriate thresholds. One problem of this testing mechanism is that it required a great deal of time, as well as concentration, because the probability of introducing human errors was very high.

The second opportunity for testing was intended to be a Hi-Fi, after both groups had completed their developments and were ready for an integration of the systems. However, this was not possible, creating the need for a simulation of those sub-systems that were not finalised and their interfaces with the existing components of the system. Therefore, for testing purposes a virtual sensor panel was created (see section 5.2). In the following sections, a possible test sequence is described.

5.3.1 Checking the data propagation

In order to test the data propagation both entities (robot and fireman) must stay in the same room. If the temperature reading gets higher than 60 degrees, both entities change their situation status to orange (potentially dangerous). As the temperature returns below 60 degrees the state changes back to the normal situation for both entities.

If the entities are not in the same location, only the status of the robot should change. The same test can be done for gas (threshold at 4) and smoke (threshold at 20).

5.3.2 Testing the too_long_in criteria

The too_long_in criteria says that if an entity is too long in a potentially dangerous situation, the situation should change to critical. For demonstration purposes the time threshold is set to five seconds. Moving the temperature higher than 60 degrees and staying there for more than five seconds the state of the robot changes to critical. If the fireman is close (i.e. in the same location) his state also changes to critical.

5.3.3 Changing situation if the fireman runs

If the fireman starts to run the situation of the fireman is changed to potentially dangerous. There are different scenarios:

- if the fireman is in normal situation:
  - running changes the fireman into potentially dangerous situation;
  - when ceasing to run the situation changes back to normal.

- when the fireman is in critical situation, running does not change the situation.

6 Future work

6.1 Changes to uMove Framework

The uMove framework was adapted to be usable for the Robin application. The following changes were made:

- Definition of a new role: Sensor Carrier. An entity can be a sensor carrier, if the mounted sensors are observing the environment (location).
  - We discussed this issue with the creator of the framework. For future work there might be a better solution instead of defining this role by characterizing the sensor directly. What is observed, is it the environment or the entity itself?
The distinction has to be made in order to set the context values to the correct entity. If a temperature change is recognized (by a sensor mounted on the robot) the context must be stored with the location (environment).

- Data propagation
  - If a room temperature is changed, the situation of all sub entities might change (robot and fireman situated in this location). In order to do that for each entity the situation has to be calculated.

6.2 Changes to Robin

We acknowledged quite early on in the process that the major danger a fireman encounters on the job is heart attacks, therefore we spent a lot of time considering how to detect such an event. We discussed the technical difficulties of detecting the event of a fireman falling down using accelerometers, because peaks in the signals may not be detected with standard classifiers. Furthermore, we discussed that a sudden drop could possibly go undetected because of being a punctual event (something that happen at a specific moment) and not a status (things that always have a measurable value), and took refuge on Dix’s quote[8]:

Highly contextual interactions must take on board [...] that the most important phenomena are NOT events, but status.

Then we started considering that being still for too long a period of time is an activity which, under the right conditions, may become an indicator of something wrong with the fireman. Perhaps even a cardiac arrest. However, a far simpler solution would have been to include sensors to monitor heart rate, as it been indeed used in [4]. A future version of Robin would need to incorporate that in order to cover the most likely cause of death or danger that firefighters face.

With this contextual information, the situations that we eliminated during the design process could be restored, potentially saving lives of firemen falling in such situations, by taking prompt action in removing them from the incident and providing emergency aid.

Another problem with the system, as encountered during the Lo-Fi testing, is that it is too sensitive to the threshold values chosen, because the predicate logic for the situation manager relies on boolean variables that take values according to real variables reaching (or not) the prefixed thresholds values. Incorporating fuzzy logic may remove or diminish this oversensitivity and make the system more robust.

6.3 Personal Statements and Conclusions

6.3.1 Benjamin Hadorn

It was a very interesting project. It contained a lot of different components we had to work with (uMove, Sun SPOTs and much more). Thanks to the good structure of uMove we could rather easily implement and test the behaviour of the robin application. Unfortunately we could never really integrate our software with the sensor and NXT implementations of group 1. Also the discussion and decision making was not very organized with group 1. I think one of the problems was that many students didn’t show up at our meetings (even those held during class). After all we met our proposed goals. Thanks a lot to Adriana, she did a very good job on the situation manager, which is the heart of our implementation and also for her effort doing the documentation with \LaTeX, which is definitively not my favored tool.

6.3.2 Adriana Wilde

Being part of this project has been for me a great experience. I learned a lot about a variety of tools, some I had never encountered before. It also served as a great opportunity to put in execution ideas that otherwise would have remained just theory. I am also disappointed in the way we approached the project management problems that emerged, but I hope that in a real-world situation, as a project manager, I would have the vision and the skill to bring my team together.
as well as the authority to do so, which I could not possibly have over my peers. Having said
that, it was a delight to work with Benjamin, as he is open to different approaches and ideas,
takes responsibility, delivers what he promises and is very proficient technically speaking, with a
willingness to clarify even the silliest of the questions. I am equally grateful for the support and
advice given by Pascal and Daniel throughout. This project is theirs as much as ours. We would
like to finish with a quote from Buckley et al [5],

In Computer Science and Engineering, [real life] problems tend to be viewed as too
vague for students to understand. However, the design of a complete system starting
from a vague, ill-defined, or incompletely specified problem is an invaluable educational
experience and is exactly the sort of problem practitioners face outside of the academic
sphere. Such problems serve as a welcome contrast from much of the Computer Sci-
ence and Engineering curriculum where students are typically exposed to well-defined
problems with well-defined solutions where grades are based on meeting a professor’s
expectation. By working on a problem that is both socially relevant and not within
the experience of most students, their investment in the project begins at a high level
and at the same time helps them learn to focus on the client’s needs instead of their
own assumptions.
References


A Appendix

Here the pseudo-formal algorithms to the main programs developed are presented.

A.1 Situation Manager

\begin{verbatim}
function checkSituation(in actor : Entity; in place : Entity) : SituationStatus
begin
if (actor.role = ACTOR or actor.role = ACTOR_SENSORCARRIER) then
begin
//If all readings are ok
if (check_NOT_too_hot(place) and check_NOT_too_cold(place) and check_NOT_gas(place)
and check_NOT_smoke(place)) then
begin
if check_running(actor) then
return <POTENTIALLY_DANGEROUS>
else
return <NORMAL>
end
//If at least one reading is out of range
else if (check_too_hot(place) or check_too_cold(place) or check_gas(place) or
check_smoke(place)) then
begin
if (check_too_long_in(actor, place)) then
return <CRITICAL>
else
return <POTENTIALLY_DANGEROUS>
end
else
return <POTENTIALLY_DANGEROUS>
end
//Situation of all locations is normal
else
return NORMAL
end
end
\end{verbatim}

\subsection{Robin Controller}

\verbatim
procedure controlRobot(in eFiremanActivity : Activity;
in eRobotSituation : SituationLevel)
begin
if (pActor.name = "Fireman 1")
begin
// activation of the fireman
switch(eFiremanActivity)
//fire man is not moving
case BEING_STILL:
default:
switch(eRobotSituation)
case NORMAL:
case POTENTIALLY_DANGEROUS:
default:
\end{verbatim}
robot.stop();
break;
// backtrack (retract) from the dangerous place
case CRITICAL:
robot.retract();
break;
endswitch
break;
// fireman is running
case RUN:
switch(eRobotSituation)
  case NORMAL:
default:
robot.exploreBackward();
break;
case CRITICAL:
this.retract();
break;
case POTENTIALLY_DANGERROUS:
this.retract();
break;
endswitch
break;
// walking or crouching
case CROUCHING:
case WALK:
switch(eRobotSituation)
  case NORMAL:
default:
this.goForward();
break;
case CRITICAL:
this.retract();
break;
case POTENTIALLY_DANGERROUS:
this.goForward();
break;
endswitch
break;
endswitch
end
end
end {verbatim}

\subsection{Mobile Device Communication}
\subsubsection{Mobile Device Communication}
\begin {verbatim}
\end {verbatim}
\subsubsection{Mobile Gate}
\begin {verbatim}
procedure sendNotification(in eLevel : SituationLevel)
begin

27
switch(eLevel)
case CRITICAL:
  send(3);
  break;
case POTENTIALLY_DANGERROUS:
  send(2);
  break;
case NORMAL:
  default:
  send(1);
  break;
endswitch
end

\subsubsection{Mobile Device}
\begin{verbatim}
procedure receiveNotification()
var i : Integer;
begin
while(true)
begin
  i = receive();
  if (i == 1) then
    clearMessage();
  else if (i == 2) then
    showMessage("Careful: Danger ahead", OrangeColor);
  else if (i == 3) then
    showMessage("Critical: Leave the room", RedColor);
  end
end
\end{verbatim}

\section{Appendix}

In the following sections, modifications to the \texttt{uMove} code are presented, with the brief description of the problem that we encountered, and how the solution was implemented.

\subsection{Changes in Entity class}
Problem: Sensor values are stored as context only to the entity the sensor is attached to. \texttt{\texttt{\texttt{\texttt{\texttt{\texttt{\texttt{\texttt{\texttt{\texttt{}}}}}}}}}} Solution: The context value are stored depending on the entity role to the correct entity (entity or environment).

\begin{verbatim}
class Entity
{
  protected void processMessage()
  {
    // get the next message received (if exits)
    m = getMessage();
    if (m != null) {
      ....
      else if (m.getKey().equals("Temperature")) {
        //if the entity is a sensor carrier -> update the location entity
        if (this.getRole() == Role.ACTOR_SENSORCARRIER)
          ....
      }\end{verbatim}
Iterator<Entity> pIt = this.getAllLogicalLocations().iterator();
while(pIt.hasNext()) { pIt.next().sendMessage(this, m); }
else
{
    this.setTemperature((Temperature) m.getValue());
}
setChanged();
this.broadcastMessage(new Message("Temperature", this));
System.out.println("Entity: (from " + this.getIdentity().getName() + ") " + "has a change of temperature");
} else if (m.getKey().equals("Smoke") { 
    // if the entity is a sensor carrier -> update the location entity
    if (this.getRole() == Role.ACTOR_SENSORCARRIER) {
        Iterator<Entity> pIt = this.getAllLogicalLocations().iterator();
        while(pIt.hasNext()) { pIt.next().sendMessage(this, m); }
    } else {
        this.setSmoke((Smoke) m.getValue());
    }
} else {
    this.setGaz((Gas) m.getValue());
} setChanged();
this.broadcastMessage(new Message("Gaz", this));
System.out.println("Entity: (from " + this.getIdentity().getName() + ") " + "has a change of gas");
} else {}
...

A.2 Changes in MessageSender Class

Problem: If there is no message listener defined, the message is not sent to any one.
Solution: If no message listener is given, then the message is broadcast to all.

{ class MessageSender
{
protected void unicastMessage(Message m, MessageListener ml)
{
    if (state)
    {
        if(ml == null)
        {
            broadcastMessage(m);
        }
        else if (listenerVector.contains(ml))
        {
            listenerVector.get(listenerVector.indexOf(ml)).sendMessage(this, m);
            state = false;
        }
    }
}

A.3 Changes in Observer Class

Problem: If the context of a location changes only the situation for the location is evaluated. We also removed the code redundancy.
Solution: If the context of a location changes all sub entities has to reevaluate their situation too.

    
    {class Observer
    {
        public void processMessage() {
            
            // get the next message received (if exits)
            Message pMessage = getMessage();
            if (pMessage != null)
            {
                //-------------------------------
                // location change
                if (pMessage.getKey().equals("Changed Location"))
                {
                    EntityLocation me = ((EntityLocation) pMessage.getValue());
                    Entity pEntity = me.getEntity();
                    if (broadcastSituationAlert(pEntity, false))
                    {
                        System.out.println("Observer: " + me.getEntity().getIdentity().getName() + " has moved into " + me.getEntity().getAllLogicalLocations().get(0).getIdentity().getName() + ", " + me.getEntity().getAllLogicalLocations().get(0).getIdentity().getDescription());
                    }
                    else
                    {
                        System.out.println("Observer: invalid activity for [Changed Location]");
                    }
                }
                //-------------------------------
                // activity change
                else if (pMessage.getKey().equals("Activity"))
                {
                    
                }
            
            }
Entity pEntity = ((Entity) pMessage.getValue());
if (broadcastSituationAlert(pEntity, false)) {
    System.out.println("Observer: " + pEntity.getIdentity().getName() + " has changed activity " + pEntity.getActivity().toString());
} else {
    System.out.println("Observer: invalid activity for [Activity]");
}

// special context change
else if (pMessage.getKey().equals("Entity Structure") ||
          pMessage.getKey().equals("Identity") ||
          pMessage.getKey().equals("Status") ||
          pMessage.getKey().equals("Type")) {
    setChanged();
    this.broadcastMessage(pMessage);
}

// general context change
else {
    /*all context is handled this way*/
    if (message.getKey().equals("Temperature")) {
        Entity pEntity = ((Entity) message.getValue());
        if (broadcastSituationAlert(pEntity, true)) {
            System.out.println("Observer: " + pEntity.getIdentity().getName() + " has a context update.");
        } else {
            System.out.println("Observer: invalid activity for [Context update]");
        }
    }
}

A.4 Changes in View Class

Problem: Temperature and other context messages are not sent to observer.
Solution: All messages are broadcast to observer

{ class View
{   public void processMessage() {
        // get the next message received (if exits)
        message = getMessage();
        if (message != null) {
            ... } else if (message.getKey().equals("Removed Entity")) {
                movedEntity = (EntityLocation) message.getValue();
                addWatchedEntity(movedEntity);
A.5 Changes in Widget Class

Problem: Multithreading problem
Solution: use lock to protect critical section

```java
public abstract class Widget extends MessageSender implements Runnable {
    private Semaphore m_pMutex;
    public Widget() {
        super();
        messageVector = new Vector<Message>();
        widgetId = new UID();
        messageSync = new MessageSync();
        m_pMutex = new Semaphore(1);
    }
    abstract protected void processMessage();
    // the logic of capturing sensor data must be implements here
    /**
     * Allow the messageSender to send its message to the listeners.
     * @param s MessageSender
     * @param m Message
     */
    public void setSensorData(SensorData sensorData) {
        try {
            m_pMutex.acquire();
            messageVector.add(new Message("Data", sensorData));
            setChanged();
            messageSync.notifyCommit();
            m_pMutex.release();
        } catch(InterruptedException e) {
        }
    }
    /**
     * Get the next message received if it exits and remove it from the list
     * If no message are available return null value.
     * @return Message
     */
    protected Message getMessage() {
        Message pRetVal = null;
        try {
            //
        } catch (InterruptedStateException e) {
            //
        }
    }
}
```
A.6 Changes in TemperatureWidget Class

Problem: Nullpointer exception.
Solution: Check if the content of the message is not null.

```java
public void processMessage() {
    m = getMessage();
    if (m != null && m.getValue() != null) {
        if (m.getValue() instanceof Temperature) {
            processTemperature((Temperature) m.getValue());
        }
    }
```

```java
} catch(InterruptedException e) {
} return pRetval;
```

```java
} 
```