TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

XIANGBIN XU

AN ONTOLOGY-DRIVEN VISUALIZATION MODEL FOR PRODUCTION MONITORING SYSTEMS

Master of Science Thesis

Examiner: Professor *José Luis Martínez Lastra*
Examiner and topic approved by the Faculty Council of the Faculty of Mechanical Engineering and Industrial Systems on 4th September 2013.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master of Science Degree Programme in Machine Automation
XIANGBIN XU : An ontology-driven visualization model for production monitoring systems
Master of Science Thesis, 82 pages, 12 Appendix pages
OCTOBER 2013
Major: Factory Automation
Examiner: Prof. José Luis Martínez Lastra
Keywords: Visualization, Adaptivity, Context-awareness, Production Monitoring Systems, Ontology, Factory Automation

Contemporary production monitoring systems tend to focus on the integration of information of manufacturing from the field level, production level to the enterprise level. They are desktop based and provides static interface for all the users while requiring a lot of hardcoding logic. The production monitoring system developed by us brings new concepts and ideas such as its mobile platform. Moreover, its adaptive information visualization of the information that takes the context of user into account is expected to improve the usability of the system.

This adaptivity feature is realized by the designed ontology-driven visualization model in this thesis. The visualization model contains sufficient information about the context, user, device, tasks and interface thanks to the powerful semantic technology-ontology. Furthermore, ontology allows defining rules and reasoning for the relationships of the concepts in the visualization model. Hence the logic and intelligence of the adaptivity is separated out of the interface application itself instead of hardcoding.

In order to validate the designed the visualization model and methodology, several adaptation effects are realized by integrating the visualization model in the project for our test bed-Fastory production line, which is a cell phone assembly line.

This thesis provides a complete approach for the design of the visualization model, including the design of adaptation effects, user modelling and logic by ontology. Finally, the test of the visualization model is demonstrated.

PREFACE

The purposes of the thesis are to design a semantic user model for the visualization model and to integrate it in the production monitoring system developed at FAST-Lab of the Department of Production Engineering of Tampere University of Technology. The state-of-the-art of this thesis offers fundamental theory on the user modelling and paves the way for a semantic ontology-driven method for user modelling. Finally, a visualization model based on the designed user model is created to provide adaptivity feature for the monitoring system.

This thesis marks the end of 2 years of my master degree study. During my work on the thesis, many people kindly supported and encouraged me. Here I would like to express my appreciation to Professor Jose L. Martinez Lastra for the support and opportunity offered to me to complete my thesis in the FAST Lab where rigorous and creative atmosphere inspired me.

To the supervisor of my thesis work Angelica, for her endless patience, passionate help and encouragement. I am deeply grateful for everything she has done for my thesis.

I also feel like to give my thanks to the colleagues working with me in the ASTUTE project: Oscar and Ville. During the work, they provided great cooperation and support.

Last but not least, I would like to express the most kind and special gratitude to my family that always support me in every aspect.

To all of you a great thank you.

Tampere, October 07th, 2013

CONTENTS

1. INTRODUCTION

1.1. Background

In the manufacturing industry, human-machine interface (HMI) is used to monitor, supervise and control the production process, which gives the ability to the operator, and management to view the plant in real time. With the proliferation of information and communication technologies (ICTs), HMI has become an essential component in modern industrial automation systems, significantly improving productivity and convenience. More and more manufacturing system designers are recognizing the benefits of using HMI to manage a variety of production activities.

Monitoring systems in manufacturing are more often than not, customized solutions that employ many different technologies at different levels. At the workshop level, the HMI of SCADA system is the apparatus or device which presents processed data to a human operator, and through this, the human operator monitors and controls the process. At production level, the HMI of Manufacturing Execution System (MES) presents the status of the manufacturing process to show the manufacturing decision maker how the current conditions on the plant floor can be optimized to improve production output. At the enterprise level, the HMI of Enterprise resource planning (ERP) system gives a company an integrated real-time view of its core business processes such as production, order processing, and inventory management.

While at the moment, most of the focus in the implementation of current monitoring systems in manufacturing is given to the integration of information necessary to be able to monitor the system at different levels, few concerns are given to how a human-centred design of monitoring systems can further improve the productivity of the manufacturing process as well as the user experience. Moreover, with increasing complexity and amount of information in the manufacturing process, a mediocre design of monitoring system could even decrease the productivity because the user has to spend more time dealing with the monitoring system other than focusing on the manufacturing process itself. Therefore, in order to further improve the productivity of manufacturing process, monitoring systems need to also focus on the usability and user experience.

1.2. Problem definition

1.2.1. Justification of the work

For manufacturing industry, monitoring systems are expected to facilitate the manufacturing process management. However, more often than not, the user could be confused by the HMIs that are complex and unintuitive. For example, the user may receive a large amount of information at the same time. As a result, the overloaded information either distracts the user from the main task or obstructs decision-making. To solve this problem, HMIs need to be optimized and adaptive to let people work more naturally and seamlessly with the help of computers. Ubiquitous computing, now also called pervasive computing is one of the solutions, which was first described by Mark Weiser [1]. Its essence is the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users [2]. In other words, it aims to make computers more helpful and easier to use. Ubiquitous computing systems are built upon relevant context information that is used to respond and adapt to users' behaviours [3]. The first publication related with context-aware systems was presented in 1994. Since then, many developments have been done in pervasive, personal and ubiquitous computing, smart homes, smart vehicles, asset tracking, tour guides, smart factories, health monitors and many others [4]. The context may include the user's role (operator, manager, maintainer, etc.), the device used to show the interfaces, cognitive state, place or any information that can be used to characterize the current situation. Therefore, context-aware HMIs are useful in optimizing the trade-off between the goal of making information available and the limitations of the users, and making appropriate adaptations. This is true also in manufacturing domain since normally manufacturing process involves a large amount of elements such as machine, material, tools and humans. Context-awareness could help the user cope with such complex environment.

On the other hand, the emergence of mobile technologies and devices brings possibilities of designing mobile-based industrial HMIs. Mobile computing poses a series of unique challenges for the development of mobile information systems [5]. In terms of human interface (UI) design and development, Jacob Eisenstein concludes in his study that mobile computing requires UIs be sensitive to platform, interaction and users [6]. This complicates the context that is needed for proper adaptation of the mobile-based HMIs. For example, different mobile devices have various operating systems, diverse modalities available, context changes rapidly and further personalization for the preference of the user is highly expected. However, mobile-based monitoring systems significantly give the user freedom when they deal with everyday tasks in the manufacturing process.

Hence a mobile-based context-aware monitoring system can bring many new concepts, ideas and improvement to the existing monitoring systems used in manufacturing.

1.2.2. Problem statement

ASTUTE is a EU project that aims at the development of an advanced and innovative pro-active HMI and reasoning engine system for improving the way the human being deals with complex and huge information quantities, during real operations that without any type of assistance would saturate his performance and decision-making capabilities in different operative conditions and contexts [7]. The proactive context-aware monitoring system developed by FAST laboratory at Tampere University of Technology is an HMI application of ASTUTE project in the manufacturing process management domain. It implements the reference architecture for the development of human machine interactions defined in the ASTUTE project and ultimately realizes proactive information retrieval and delivery based on the situational context, as well influenced by information content and services, and user state information. Figure 1.1 shows the implementation of HMI application in the manufacturing domain where the users with different roles utilize the mobile devices to perform their specific task. The HMI application on the mobile devices allows the user to check the information of the production line connected to the server.

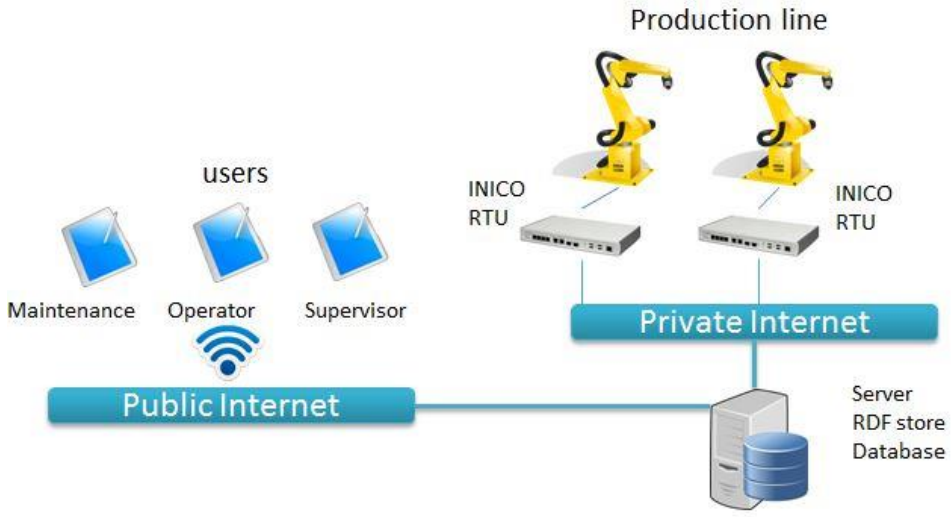


Figure 1. 1 *Diagram of Production Management Demonstrator*

The user interfaces need to consider the user needs, preferences, and the characteristics of the display device as well as the state of the monitored system. It will be needed to create visualization models to adapt the interfaces and allow the possibility of using mobile devices for monitoring purposes. Therefore, the purpose of this thesis is to create and test visualization models for generating user interfaces for the industrial domain which will be used to monitor manufacturing systems at different levels through mobile devices. The definition and deployment of the interfaces for mobile devices will be a contribution to the major improvement for future monitoring systems in the industrial domain that ASTUTE project is developing.

1.2.3. Work description

1.2.3.1 Objectives

The objectives of the thesis are:

1. Design a user model that contains sufficient knowledge about the user profile (role, preference, device etc.) and context so that human-machine interface (HMI) generator can create adapted HMI layout and logic based on the user model.
2. Design the HMI logic and intelligence based on the user model.
3. Develop a visualization Model working with other models to realize context-aware and adaptation features of the monitoring system.
4. Test the visualization model on mobile devices by adapting interfaces.

1.2.3.2 Methodology

The need of context-aware and adaptation behaviours sparks a growing demand for context modelling which includes user modelling and device modelling and so on. Because adequate representations of knowledge about a user, device, context and even cognition, effective elicitation and utilization of related information for making helpful adaptation and proactive decisions are critical factors for the success of adaptive HMIs.

For the purpose of creating necessary models for the adaptation behaviour, semantic technology such as ontologies can be a promising candidate. Because ontology-based context modelling offers the following benefits:

- Representing machine-understandable knowledge about the user, context and devices semantically.
- Facilitating the integration of models and providing better understanding and managing of models' complexity.
- Updating and accessing the knowledge representation
- Querying and inferring implicit knowledge

In the thesis, ontologies are employed to develop the context model for use within a context-aware monitoring system.

For the designing of useful adaptation features, the interaction behaviour and its relevant factors should be understood. The embodied cognition-task-artefact framework is a theoretical framework for understanding interaction behaviour between machine and human, so it serves as a guideline for both modelling and designing adaptation features.

Moreover, for specific adaptation features, a pattern-based approach will be applied because it provides sufficient options of abstract adaptation features.

1.2.3.3 Assumptions and Limitations

In order to provide adaptation features in the monitoring systems, the systems need to know the information about the specific user, such as who is the user, what is the user needs at a moment, what kind of situation the user is in. Only by knowing this information, can the systems make right decisions on how to adapt the interface. User modelling is considered as a solution in the Human-computer interface (HCI) researches as there is the potential that user modelling techniques will improve the collaborative nature of human-computer systems.

This thesis utilized a knowledge-based user modelling method that models user information and context information specifically related to the production environment of the Fast laboratory in Tampere University of Technology. And the users are assumed to be the employees in this production environment and have basic production engineering background.

This thesis aims to design the visualization model that defines the user model and adaptation logic, however, the visualization model is not responsible for deploying the interfaces in the mobile devices, or for collecting real-time context information and user information. In fact, other models will serve this purpose. The models are not extended automatically; it is needed to manually make the changes of the models if necessary. For now, they are only developed for Android devices.

1.3. Outline

This thesis is organized as follows. Chapter 2 presents the theoretical background of the technologies used in the design and the methodologies that guides the design process in the thesis. Chapter 3 introduces explicit design methods of the visualization model. Chapter 4 presents the integration and implementation of the visualization model in the visualization level of the project. The results of the developments are presented in Chapter 5. Finally, Chapter 6 gives the conclusions on the visualization model design and puts forward the possible future work in this domain.

2. THEORETICAL BACKGROUND

2.1. User Interface for production monitoring

A production monitoring system (PMS) is a production tool that helps different participants in the production process to notice and receive information in the shop floor as events are happening. It can also be used to help people handle these events efficiently. Run time PMS is essential in helping the industries to meet realistic production goals, at reduced down time and increase in yield.

The user interface for production monitoring has two main tasks. The first on is the ability of the PMS to collect manufacturing information at run time would enable the production team to respond in a timely manner to handle production related issues that may arise. Second, it is to assist the production team to produce products with available resources at the same time improving quality matters and reducing overheads. Third, it also proactively detects and reacts to the faults by informing the relevant personnel in the departments before they escalate. [8]

The benefits of having an effective and productive user interface for production monitoring is the immediate screen access to all production related information.

- Man power (Operators)

Operators in the manufacturing domain are the employees who directly manipulate or collaborate with the equipment to produce products. The production monitoring interface is a reliable tool for assisting the operators particularly in informing operators of their performance to date or notifying them of production orders. Moreover, it empowers the operator to recognize faults and react to the system in alerting the other cooperative departments to solve problems.

- Supervisors

Supervisors often supervise, plan and manage the production activities. The production monitoring interface also benefits them in enabling them to monitor the performance of the production lines. This will make them be able to keep production output on track.

- Maintenance team

Maintenance team is responsible for implementing preventive maintenance plans, optimizing the manufacturing system and solving malfunctions. The user interface for production monitoring helps the maintenance team quickly trace the source of system error and provide accurate information to assist them to fix the problem. In addition, the record of the equipment efficiency engages maintenance team to optimize the performance of the equipment.

- Management

Management is responsible for making enterprise plans and business operation. The production related information is presented to the management and supervisors via the user interface. This makes reporting easier compared to conventional methods.

2.2. Design methodology of user interface

2.2.1. User-centred design

User-centred design (UCD) is an approach to user interface design and development that involves users throughout the design and development process [9]. In recent years, the need for user-centred design in the development of embedded systems has been recognised ([10], [11], [12]). User-centred design not only focuses on understanding the users of a computer system under development but also requires an understanding of the tasks that users will perform with the system and of the environment (organizational, social, and physical) in which they will use the system. ISO 9241-210:2010, Ergonomics of Human-system interaction – Part 210: Human-Centred Design for interactive Systems [13], provides guidance on and lists the main principles and essential activities for human (user)-centred design, for achieving usability in systems. The six main principles of human-centred design are:

1. The Design is based upon an explicit understanding of users, tasks and environments.
2. Users are involved throughout design and development.
3. The design is driven and refined by user-centred evaluation.
4. The process is iterative.
5. The design addresses the whole user experience.
6. The design team includes multidisciplinary skills and perspectives.

The four essential human-centred design activities are:

1. Understanding and specifying the context of use
2. Specifying the user requirements

3. Producing design solutions
4. Evaluating the design

Applying the approach prescribed by ISO 9241-210 brings several benefits:

1. Increasing the productivity of users and the operational efficiency of organizations
2. Being easier to understand and use, thus reducing training and support costs
3. Increasing usability for people with a wider range of capabilities and thus increasing accessibility
4. Improving user experience
5. Reducing discomfort and stress
6. Providing a competitive advantage, for example by improving brand image
7. Contributing towards sustainability objectives

2.2.2. Situational embodied cognition-task-artefact framework

Apart from UCD, specifically for the understanding of interactive behaviour, Michael Byrne describes the embodied cognition-task-artefact framework known as ETA triad. It is based on the idea of the Cognition-task-Artefact triad introduced by Gray. The central notion is that interactive behaviour of a user interacting with an interface is a function of the properties of three things: the cognitive, perceptual and motor capabilities of the user, termed Embodied Cognition, the Task the user is engaged in and the Artefact the user is employing in order to do the task [14] (see Figure 2.1).

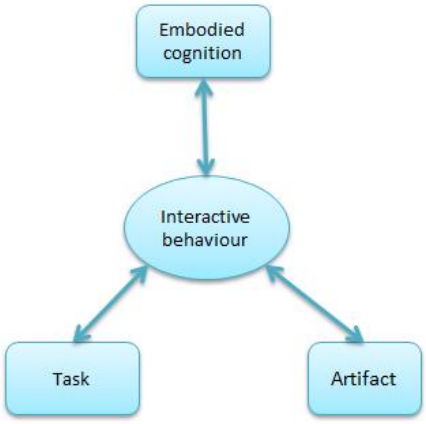


Figure 2. 1 *The embodied cognition-task-artefact triad*

First, the user's Embodied Cognition refers to the cognitive capabilities and limitations of the user, and the perceptual-motor. This makes the coordination of perception, action

and cognition, rather than just cognition itself. Because computer systems become increasingly embedded, mobile and user interfaces are increasingly multimodal, time-critical. The demands they place on the perceptual-motor systems are likely to become central to understanding interactive behaviour. However, the cognitive system has the bulk of the responsibility in coordinating the three.

The second component is the task. The interfaces should always be optimized in order to help users conduct their engaged tasks. This brings another important issue-the way by which success in performing a task is measured. In some high-performance systems, time and errors are regards as the most central measures with user preference and satisfaction being less pivotal.

The last component is the artefact that determines which operators the user can apply to reach their goals and often plays a central role in maintaining state information for the task. The artefact is the component that is most subject to design. Due to increasing popularity of mobile devices, various operating systems, modalities and devices are available. This requires the interface designers to make specific trade-off between the goal of making information available to the user and features of devices. Furthermore, automatic adaption to different devices is highly expected because it could save a lot of time compared with modifying the interface to each single type of devices.

Nevertheless, one limitation of the ETA triad is the absence of environment component. The advances in low-power and the wireless communication capabilities have brought mobility forward. Mobile phones and tablets will become multifunctional and multimodal tools which offer us permanent access to all sorts of equipment. We will be able to work anywhere and anytime using any device we like. Hence another important problem appears concerning knowledge about the environment of interaction as included in the UCD methodology. When we operate today's wire-based systems, the wire installation gives us implicit information about the place of interaction. In mobile applications we will never know exactly where this place is. The user may be seated in front of the machine as well as in the office. And the environment will have influence on all the three components in the ETA triad. It is necessary to add environment component in the ETA triad as a situational ETA triad as shown in Figure 2.2.

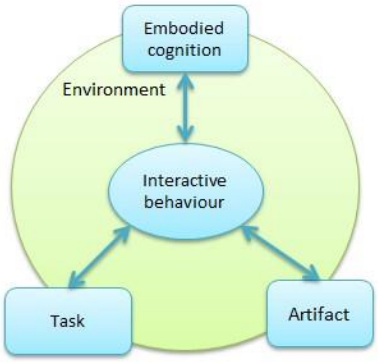


Figure 2. 2 *The situational ETA triad*

In the process of HMI design, UCD approach is adopted to improve the usability of the final system. During the development of the visualization model of the HMI, the situational ETA framework is a major practical theoretical principle in the study.

2.3. User interface adaptation and user modelling

2.3.1. Overview

The design of appropriate human-computer interactions often needs to respond to changing technology and functionality, especially when new types of devices like tablet gain more usability in the industrial domain. As computational capability and data processing become more distributed, static design of human-computer interactions for dynamic environments, for example, for mobile environment, may not always generate intuitive interfaces between people and machines. Static interaction design is often no longer sufficient to meet the complex task and information needs of current systems. Increases in computing power and the appearance of more ubiquitous, distributed capabilities will make the tasks and information that users need even more unpredictable, greatly increasing the difficulty of providing good interaction design. Due to the non-deterministic characteristic of a ubiquitous computing environment with a large number of computational and human agents, stationary design will no longer be sufficient to provide useful and usable interfaces between people and machines. It will be impossible to predict the interactions that will be required for a particular system and user, the information that will be necessary, or even the information that will be available. It will also be impossible to predict or control the hardware and software capabilities that will exist in local and distributed forms or the experience and capabilities of the human and software participants.

A series of projects were conducted to demonstrate the feasibility and usability of adaptation features of human-computer interface. In the EU project AmbieSense [15], services for users at an airport were developed. The system makes use of contextual parameters to adapt what kind of information to present to the user and in which form to

show it. The adaptation depends on the current status of the traveller such as departing, arriving or in transfer. E.g. information about check in counters was only available before the traveller enters the security control. Information about tax free shopping was only shown to international travellers, which reduces the information burden to those who do not need the information at all.

Another example is a HMI application for automotive called Supermarket Guide [16]. Its purpose is to provide information about nearby supermarkets and their current offers. This external application can be integrated to the in-car head unit. The interface is shown in Figure 2.3.

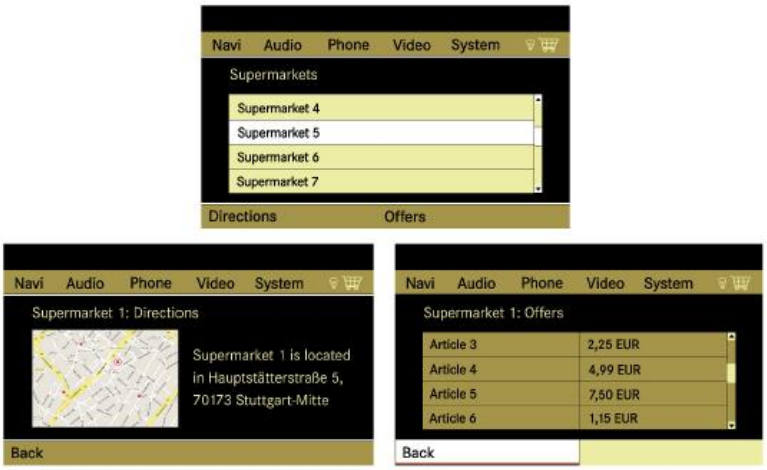


Figure 2. 3 *Supermarket Guide*

The adaptation of the Supermarket Guide relies on the user's driving history; the system can record the driver's regular route for example from his workplace to home and supermarkets he often stops by. The system can show and compare the offers from these supermarkets during the way and decide which supermarket he should go and also the best route. Other than AmbieSense which filters unnecessary information for the user, the adaptation feature of Supermarket Guide provides personalized content, facilitating the user's decision making.

In order to design personalized functionalities of human-machine interface, adaptive system should first establish a user model to be developed for each user during his activity. Wahlster and Kobsa stress that a user model is a knowledge source which is separable by the system from the rest of its knowledge and contains explicit assumptions about the user [17]. In other words, a user model is a representation of information about an individual user developed by an adaptive system in order to provide users with personalized functionalities [18]. Such differentiate behaviour for different users are called the system's adaptation effect and could have many forms:

- Adaptive content presentation: when the user accesses some resources based on a certain event, the system can provide related items of most interest to the particular user, in a preferred form (chart, 3D, text, augment reality etc.) [19].
- Adaptive modality: The system can use suitable modalities to present the information according to the particular situation where the user is using the system.
- Adaptive navigation support: When the user navigates from one resource to another, the system can manipulate the links (e.g., hide, sort, annotate) to provide adaptive navigation support.
- Personalized display: The system can adopt different fonts, colour or layout according to the preference of the user.

User modelling is a subdivision of human–computer interaction and describes the process of building up and modifying a user model. The main goal of user modelling is customization and adaptation of systems to the user's specific needs [20]. The system should collect data about the user in two main ways: 1) implicitly observing user interaction and parameters of the environment, 2) explicitly requesting direct input from the user.

User modelling and adaptation are complementary one to another. The amount and the nature of the information represented in the user model depend on the adaptation effects that the system has to deliver.

2.3.2. User Modelling Dimensions

The core idea of adaptation is based on the assumption that differences in the characteristics of the components in the situational ETA triad should influence the individual utility of the service/information provided; hence if system's behaviour is tailored according to these characteristics, the system usability will be improved. This section will describe the most important dimensions that could be utilized based on the situational ETA framework.

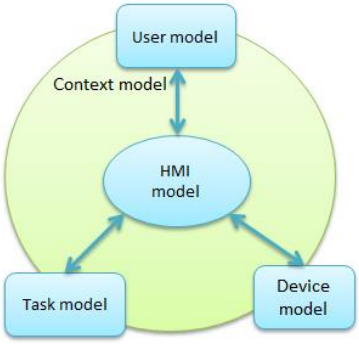


Figure 2. 4 *Four major dimensions of user modelling*

Figure 2.4 demonstrates five aspects of user modelling.

User model includes unique characteristics of the individual user:

- Knowledge and background – These characteristics are especially important for the adaptive systems modelling students [21]. Adaptive Educational Systems are one of the adaptive systems that have the longest history of research. For these systems student's knowledge is the major characteristic defining system's adaptivity. The most popular approach to modelling student knowledge is to map the student's knowledge to a fine-grained conceptual structure of a learning domain. However, this characteristic is not only restricted in the educational system, rather, it applies in any adaptive systems with which the user needs to use his professional knowledge. For instance, an electrical technician may use a system to solve electrical problems. Hence the system could adapt according to the user's knowledge about electronics or electrical engineering, and then provides personalized information to help the user to make decision.

- Roles and interests – Users having different roles tend to have their own information of interest and preferred visualization of the information. For example, when a manager is using the monitoring system to check a robot, he is more likely to be interested in the productivity of the robot, whereas for the operator, the parameters and accuracy of the robot may be the information of interest. Such user interests play an important role in adaptive information retrieval and filtering systems. Moreover, the systems can distinguish users by their role which implies both the responsibilities and aspect of knowledge.

- Preference – User preference is always a major consideration in the adaptive systems. It could be obtained either explicitly or implicitly; many information systems provide setting option for the user to define his preferred font, font colour, font size and even layout so that the user can get customized interface, whereas some recommendation systems use user history, cookies and other techniques to infer the user's preference and recommend contents to the user.

- Cognition – Cognitive model is a representation of mental states of the user. User interfaces requires something similar to mutual understanding in human-human interaction. From communication by means of language, it is known that successful communication requires mutual adjustment of the utterances of the speaker to the listener's state, for example, the listener's knowledge about the topic, emotions, personalities and states like confusion, fatigue, stress, and other task-relevant affective states. A cognitive model is capable of solving tasks using the same cognitive steps as humans use to solve the tasks. Currently, the best way to build models of cognition is to use a cognitive architecture (e.g. ACT-R).

The nowadays most mature framework that works well for building models of cognition is ACT-R/PM [22], a system that combines the ACT-R cognitive architecture [22] with a modal theory of visual attention [23] and motor movements [24]. ACT-R is a cognitive architecture, a theory for simulating and understanding human cognition. ACT-R/PM contains precise methods for predicting reaction times and probabilities of responses that take into account the details of and regularities in motor movements, shifts of visual attention, and capabilities of human vision. A true model of embodied cognition can be made by extending ACT-R/PM incorporating the effects on performance. For example, apart from handling the interactions among vision, memory and body movements, the model can become fatigued over time and distracted when there is too much to attend to. Such a capability can be applied to adaptation systems so that different affective and cognitive diagnoses such as confusion, fatigue, stress, momentary lapses of attention, and misunderstanding of procedures can be captured. Based on this, different adaptation effects can be made such as simplifying the interface, highlighting critical information, and tutoring on selected misunderstandings.

Task model:

- Tasks – The user's tasks represent the purpose for a user's work within an adaptive system. It can be the goal of the work in application systems, an information need in information access systems, or a learning goal in educational systems. The tasks indicate what the user actually wants to achieve. The user's goal is the most changeable user feature especially in adaptive hypertext systems or adaptive educational systems. However, in an application system where several user tasks have already defined, it is possible to get clue about what the user wants to achieve by capturing the user's interaction with the system.

- Event – In monitoring systems, events are those to be monitored by the system. For example, in a manufacturing system, the event can be robot error, communication error or sensors. Events often have close association with other model dimensions. For instance, once an event occurs, it can lead a certain user to a specific task. Moreover, the events are sometimes linked to devices.

Device model:

Device model represents the characteristics of the device that the user is using.

- Device – For a server-based application, the users of the same server-side application may use various devices at different times, adaptation to the user's platform becomes an important feature. One technique is focused on adaptation to

the screen size by either converting the interface designed for desktop browsers to mobile browsers or generating pages differently for these two types of devices. An attempt to standardize the description and use of platform capabilities can be found in [25]. A device model could consist of basic device information, device malfunctions, device capabilities and state machine and device services. The basic device information could contain information about device friendly name, manufacturer data and device model data. Device malfunction could represent possible errors that may occur on devices. The concept Malfunction contains general malfunction information, such as malfunction name, malfunction code. It can even be assigned with several malfunction levels or severity like error, fatal and warning. Device capabilities and state machine represents the state machine linked to a specific device. Device services present a description of the functions that the device can provide for the user. It includes the service capabilities, input and output parameters and supported communication protocols supporting the device interaction.

Environment model:

Environment model specifies the context of the user's work. Early context-adaptive systems explored mostly platform adaptation issues. The growing interest to mobile and ubiquitous systems attracted researcher's attention to other dimensions of the context such as user location, physical environment, and social context.

- User location – Adaptation to location is a major focus for mobile context-adaptive systems. As is often the case, location information is used to determine a subset of nearby objects of interests. So that this subset could define what should be presented or recommended to the user. This kind of adaptation was realized by early context-adaptive systems in a number of contexts such as museum guides [26]. Depending on the type of location sensing it is typically a coordinate-based or zone-based representation. For example, a variety of positioning systems are deployed in the SmartFactory [27] project; the floor is fitted with a grid of RFID tags. These tags can be read by mobile units to determine location data. Other systems for three-dimensional positioning based on ultrasonic as well as RF technologies are also installed and currently tested, especially in terms of the accuracy achievable under industrial conditions.

- Ambient factors – Ambient factors refer to the conditions of the location of the user's work, such as noise level, illumination level, temperature etc. The adaptation to the ambient light is nowadays common feature of mobile devices. Others like noise level and temperature are not utilized often because it requires specific sensors on the device, which are not necessary for basic functions of mobile de-

vices. However, these factors could bring some constrains for the interaction between human and the system. For example, a noisy environment may restrict some modalities to be used by a multimodality system.

2.3.3. User modelling methods

A commonly-used user modelling approach is feature-based user modelling. Feature-based models attempt to model specific features of individual users such as the user's role, knowledge, location and tasks. During the user's interaction with the system, these modelled features may change, so the purpose of feature-based models is to track and update those features in the user model in order to obtain real time adaptation to the current state of the user. Apart from feature-based modelling, stereotype modelling is another option. It is one of the oldest approaches to user modelling. Stereotype user models attempt to gather the same type of users into groups, called stereotypes. All the users belonging to the same stereotype will be provided the same adaptation effect. A user in a classical stereotype-based system is represented simply as his current stereotype. Nowadays, a popular approach is to combine feature-based method with stereotyped method. For example, use a stereotype-based user model to initialize the model for the user and then use feature-based user model to enrich and update the specific features of the individual user.

It is often the case that there is the need to deal with information that is uncertain and imprecise about the user. In feature-based user modelling, sometimes it is hard to say if the user possesses a certain feature. For example in the recommendation system, if the user looks for the information of digital cameras, most probably he has a plan to buy a digital camera, which is uncertain information. In this case, user modelling is a domain in which there are various different sources of uncertainty. To some extent, stereotype-based user modelling solves this issue by assuming the user has the feature according the user type. However, numerically-approximate reasoning techniques are more suitable for this purpose. The two popular methods are Fuzzy logic and Bayesian Networks. Fuzzy logic is an approach to computing based on "degree of truth" rather than the usual "true or false" Boolean logic. It is not a machine learning technique, nevertheless due to its ability to handle uncertainty it is used in combination with other machine learning techniques in order to produce behaviour models that are able to capture and to manage the uncertainty of human behaviour. In [28] fuzzy logic was used to model user behaviour and give recommendation using this fuzzy behaviour model. Bayesian networks (BNs) are one of the most common ways of describing uncertainty and dealing with it. BNs are a probabilistic model inspired by causality and provide a graphical model in which each node represents a variable and each link represents a causal influence relationship. Currently they are considered one of the best techniques available for diagnosis and classification problems.

2.4. Ontology

2.4.1. Overview

As the process of manufacturing becomes more flexible and complicated due to more dynamic market and consumer demands, information systems play a growing active role in the management and operations of production. Departing from their traditional role as simple repositories of data, information systems must now provide more sophisticated support to automated decision making and adaption to context. Specifically, for the monitoring HMI in our case, it must not only answer queries with the events in the production line, but must also be able to intuitively present an adapted visualization of information to the particular individual in the particular context. This requires user models to facilitate user representation, context sharing and semantic interoperability of heterogeneous systems. Here ontology is selected as a mechanism to fulfil the requirements. Because it provides several advantages such as representing machine-understandable knowledge about the user, context and devices semantically, facilitating the integration of models and providing better understanding and managing of models' complexity, updating and accessing the knowledge representation, querying and inferring implicit knowledge. For the application development, it also separates the logic out of the mobile application.

The concept of ontology in the computer science is that an ontology is an explicit and formal specification of a shared conceptualization [29]. In other words, an ontology necessarily represents knowledge of some domain of interest as a set of concepts, their definitions and their inter-relationships. It is shareable and can be understood by computer. An ontology uses five fundamental modelling primitives to model a domain:

- Classes: the terms that denote concepts of the domain; for example, in the family domain, father, mother, son and daughter are the concepts.
- Relations: the relationships between concepts; typically include hierarchies of classes such as father is subclass-of familymember.
- Functions: concept properties; for example, is-father-of (x, y) means x is the father of y.
- Axioms: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. One axiom of the family domain could be that every father must have at least a son or a daughter.
- Instances: basic objects that belong to a class; for example, *Karen* is-a *daughter* means *Karen* is an instance of the class *daughter.*

An ontology brings several benefits that are the reasons why it is employed in our application. First, sharing common understanding of the structure of information among people or software agents is one of the more common goals in developing ontologies [30].

For example, suppose several different web-based applications contain production information or provide production monitoring services. If these applications share and public the same underlying ontology they all use, then another desktop application can also use this ontology to provide services to the user within the same domain while remaining consistency of the information with other web-based applications. Secondly, making explicit domain assumptions makes it possible to change these assumptions easily if our knowledge about the domain changes. Hard-coding assumptions about the world in programming language code make these assumptions not only hard to find and understand but also hard to change. Whereas in an ontology, knowledge is represented in a human-readable manner and it is easy to understand and make changes. Analysing domain knowledge is possible once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them.

2.4.2. OWL 2 Web Ontology Language

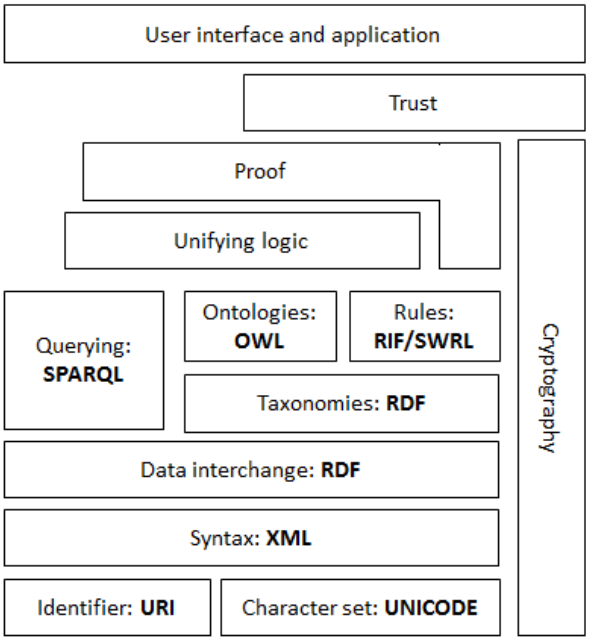


Figure 2. 5 *The Semantic Web Stack* **[34]**

OWL 2 Web Ontology Language is one of the ontology languages used to construct ontologies. OWL 2 is an extension and revision of the OWL Web Ontology Language developed by the W3C Web Ontology Working Group and published in 2004 (referred to hereafter as "OWL 1") [31]. The languages are characterised by formal semantics and RDF/XML-based serializations for the Semantic Web. The architecture of the Semantic Web is illustrated by the Semantic Web Stack shown in Figure 2.5. In this stack, XML is a surface syntax of structured documents, it doesn't have any semantic constraints on the document. XML Schema defines the structure constraints of XML documents. RDF

[32] is a data model of resources and their relationships expressed by XML syntax. It provides simple semantics for the data model. RDF Schema [33] is a vocabulary describing the attributes and types of the RDF resources. It provides generic semantics for the attributes and types. OWL adds more vocabulary to describe attributes and types, such as disjointness, cardinality in types and symmetry in attributes.

OWL has more mechanisms to represent semantics in comparison with XML, RDF and RDFSchema. Several syntaxes can be used to store OWL 2 ontologies and to exchange them among tools and applications. Table 2.1 shows a comparison of various syntaxes for OWL 2. The primary exchange syntax for OWL 2 is RDF/XML. It means RDF/XML is the only syntax that must be supported by all OWL 2 tools.

Table 2. 1 *Comparison of syntax*

Name of Syntax	Status	Purpose
RDF/XML	Mandatory	Interchange (can be written and read by all conformant OWL 2 software)
OWL/XML	Optional	Easier to process using XML tools
Functional Syntax	Optional	Easier to see the formal structure of ontologies
Manchester Syntax	Optional	Easier to read/write DL Ontologies
Turtle	Optional	Easier to read/write RDF triples

One simple example of ontology expressed in RDF/XML Syntax is shown below:

```
<rdf:RDF xml:base="http://www.semanticweb.org/ontologies/test">
<owl:Ontology rdf:about="http://www.semanticweb.org/ontologies/test"/>
<owl:Class rdf:about="http://www.semanticweb.org/ontologies/test#father"/>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/ontologies/test#John">
<rdf:type rdf:resource="http://www.semanticweb.org/ontologies/test#father"/>
</owl:NamedIndividual>
</rdf:RDF>
```

In the example, it is asserted that the type of a NamedIndividual John is class father, which semantically means John is a father.

2.4.3. Reasoning and SWRL rules

One power of ontology is its support for reasoning. By reasoning we mean deriving facts that are not explicitly asserted in the ontology. For example, A is father of B, B is father of C, and then A is ancestor of C. A reasoner is a piece of software able to performing reasoning tasks-inferring logical consequences from a set of asserted facts in

the ontology. A great number of reasoners are available such as FaCT++, Pellet and HermiT. Among these, Pellet is one of the most common reasoning engines used for reasoning with OWL models. Pellet supports reasoning with the full expressivity of OWL-DL and has been extended to support OWL 2.

A few examples of tasks required from reasoner are as follows:

- Satisfiability of a concept - determine whether a description of the concept is not contradictory.
- Subsumption of concepts - determine whether concept A subsumes concept B.
- Consistency of ABox (A fact associated with a terminological vocabulary within a knowledge base.) with respect to TBox (a conceptualization associated with a set of facts) – determine whether individuals in Abox do not violate descriptions and axioms described by TBox.
- Check an individual – check whether the individual is an instance of a concept
- Retrieval of individuals – find all individuals that are instances of a concept
- Realization of an individual – find all concepts which the individual belongs to

Besides, the reasoning capabilities can be further expanded by using the Semantic Web Rule Language (SWRL). It is an expressive OWL-based rule language which allows users to write rules that can be expressed in terms of OWL concepts. Semantically, SWRL is built on the same description logic foundation as OWL and provides similar strong formal guarantees when performing inference [35]. In a human readable syntax, a rule has the form as (1):

$$\text{Antecedent} \Rightarrow \text{consequent} \quad (1)$$

where both antecedent and consequent are conjunctions of atoms written $a_1 \wedge \ ... \wedge \ a_n$. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., ?x). Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written:
$parent(?x,?y) \wedge \ brother(?y,?z) \Rightarrow uncle(?x,?z)$ [36].

2.4.4. SPARQL

Users and applications can interact with ontologies and data by querying the ontology model using the SPARQL query language [37], which was standardized in 2008 by the World Wide Web Consortium (W3C). The standard query evaluation mechanism is based on subgraph matching and is called simple entailment since it can equally be defined in terms of the simple entailment relation between RDF graphs [38]. Given a data source D, a query uses a pattern to be matched against D, and the values obtained from this matching are processed to give the answer. A SPARQL query contains three parts.

The pattern matching part, which includes several fundamental features of pattern matching of graphs, such as optional parts, union of patterns, nesting, filtering (or restricting) values of possible matchings, and the possibility of choosing the data source to be matched by a pattern. The solution modifiers, which once the output of the pattern has been computed (in the form of a table of values of variables), allows to modify these values applying classical operators like projection, distinct, order, limit, and offset. Finally, the output of a SPARQL query can be of different types: bool queries (true/false), selections of values of the variables which match the patter ns, construction of new triples from these values, and descriptions of resources. The following example shows its general syntax. If IRIs are abbreviated using the prefix "ns", a SPARQL query is

```
PREFIX ns:    < http://www.semanticweb.org/ontologies>
SELECT ?father
WHERE  {
        ns:John  ns:hasFather ?father .
      }
```

Programme 2.1 *An example of a SPARQL query*

John is an individual and hasFather is an object property. The answer for the query tells who the father of John is.

2.4.5. Protégé

In the study, Protégé 4.3 is used to build the ontological user model for the HMI application. Protégé is a free, open source ontology editor and knowledge-base framework [39]. It was developed by Stanford Centre for Biomedical Informatics Research at the Stanford University School of Medicine. Initially, it was a small application designed for a medical domain (protocol-based therapy planning), but has evolved into a much more general-purpose set of tools. More recently, Protégé has developed a world-wide community of users, who themselves are adding to Protégé's capabilities, and directing its further evolution. The original goal of Protégé was to reduce the knowledge-acquisition bottleneck by minimizing the role of the knowledge engineer in constructing knowledge bases. In order to do this, Musen posited that knowledge-acquisition proceeds in well-defined stages and that knowledge acquired in one stage could be used to generate and customize knowledge-acquisition tools for subsequent stages [40]. Thus, the original version of the Protégé software was an application that took advantage of structured information to simplify the knowledge-acquisition process. Musen described Protégé as follows:

Protégé is neither an expert system itself nor a program that builds expert systems directly; instead, Protégé is a tool that helps users build other tools that are custom-tailored to assist with knowledge-acquisition for expert systems in specific application areas [41].

The latest version of Protégé is Protégé Desktop 4.3 as of June 2013, its graphic user interface is shown in Figure 2. 6.

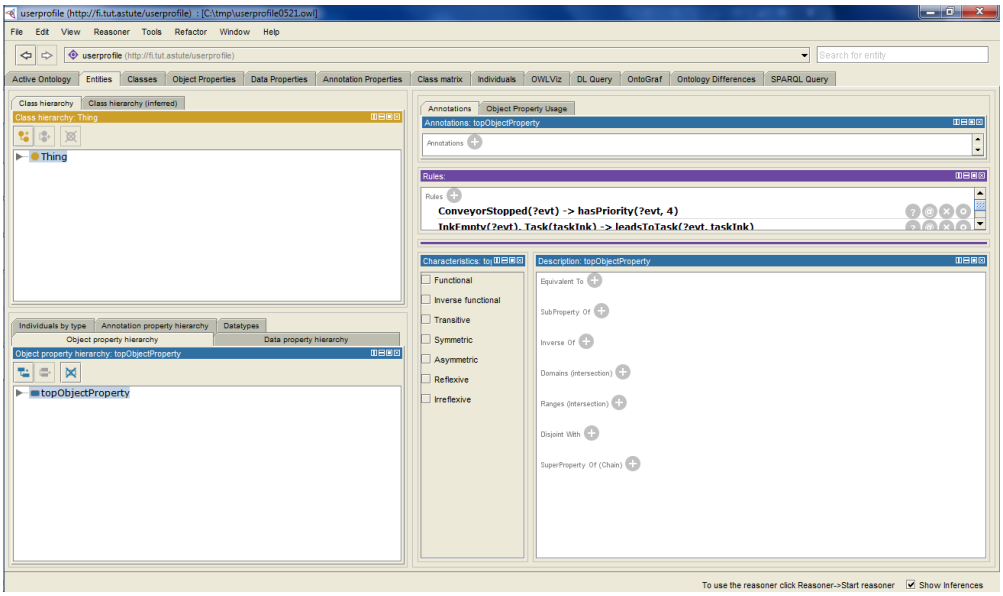


Figure 2. 6 *Protégé desktop 4.3*

It enables users to [39]:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

3. DESIGN AND INTEGRATION OF VISUALIZATION MODEL

3.1. Overview

This model that is described in this thesis generates the definition about how to visualize the information. It relies on the ontology-driven user model and pattern-based SWRL rules to relate device, task, user and environment of work in the user-computer interaction to provide a more intuitive, context-aware visualization for the user. The model is stored in a remote server to support the needed computing power required by the extensive models, which means the mobile device needs internet access to have access to the visualization model.

3.1.1. Description

The online visualization model is part of an adaptive HMI engine at the adaptive HMI level of the ASTUTE project architecture illustrated in Figure 3.1 [19]. The objective of the visualization model is to generate a definition of how to present the information to the user on mobile devices, particularly for Android devices according to the state of the monitored system, the context and users.

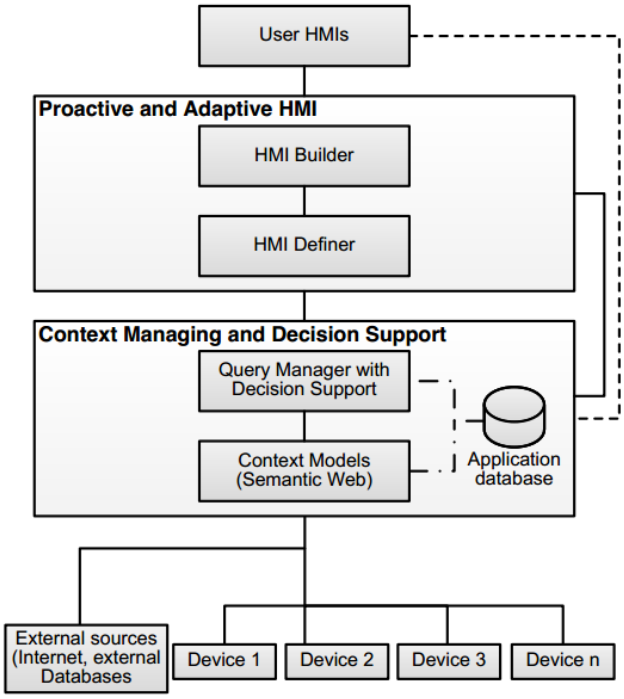


Figure 3. 1 *Architecture of ASTUTE Production Management Demonstrator* **[19]**

Figure 3.2 illustrates the main functionality of the visualization model. The generated definition of the visualization is then used by another module of the HMI Engine which takes care of the communication details and deploys the interface into the mobile devices [42]. The visualization model is built using Java SE 7, so it can be seamlessly integrated into the application on the server. It makes use of ontology-driven user modelling and pattern-based SWRL rules and it was developed for monitoring manufacturing systems.

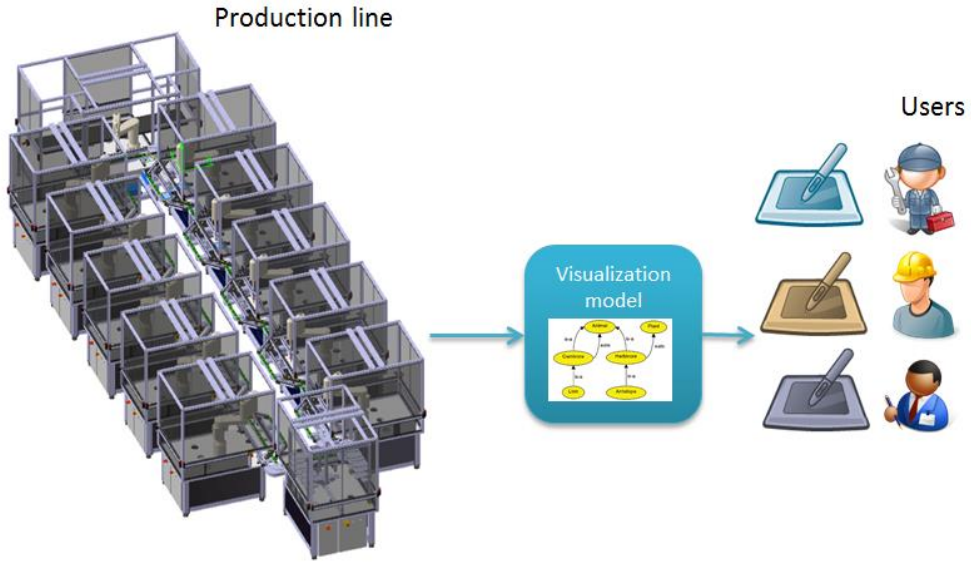


Figure 3. 2 *Functionality of the visualization model*

An ontological user model was built as a knowledge and rule base for the adaptive HMI model by using Protege. The user model contains domains involved in the user-computer interaction. As the visualization model is developed for the personnel involved in a production system, it includes the role and knowledge of the user, task, device and environment of work. The concepts in these domains are related through properties. In addition, the user model also contains an interface domain model to represent the components needed in the user interface. The adaptation effects were designed and realized through SWRL rules according to a set of generic patterns that were identified previously in ASTUTE project. The Jena api is used to manage and update the user model in the run time and the Pellet api is utilized to conduct reasoning on the user model.

The visualization model creates a new user model for a new user when he/she registers an account based on the ontological user model. For already registered users, when they log in the system, the model loads the corresponding stored user model, updates the

model while the application is running, then generates and sends the definition of visualization to the HMI Builder at run-time. Several adaptation effects were realized by the model to provide more intuitive user interface. Figure 3.3 demonstrates the aforementioned process of the visualization model.

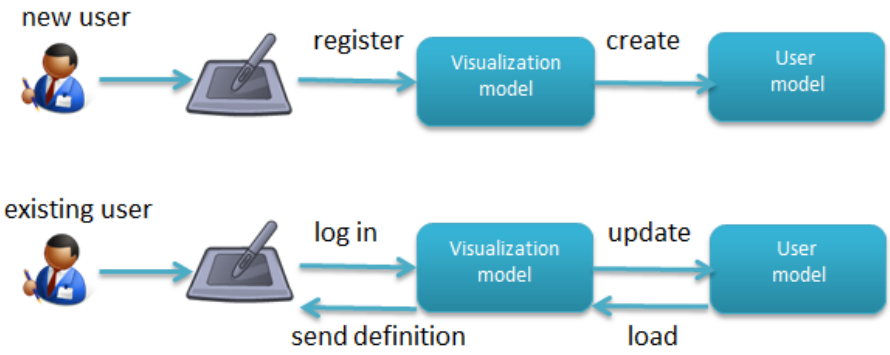


Figure 3. 3 *Functionality diagram of the visualization model*

3.1.2. Design platform and environment

The following platforms and libraries are the tools that are used to create the visualization model, including the ontology file and the server-side application.

1. Windows PC: The operating system minimum requirement is Windows XP or higher. The windows PC was used as a basic platform to create
2. Java Development Kit: Java SE Development Kit SE 7u1, available in the oracle main web page.
3. Integrated Development Environment (IDE): NetBeans IDE 7.2.1
4. Jena api 2.10.0, Apache Jena™ is a Java framework for building Semantic Web applications. Here the TDB library must be included in the Jena api, it is a native RDF database for storing the model.
5. Pellet api 2.3.1, Pellet is an OWL 2 reasoner. Pellet provides standard and cutting-edge reasoning services for OWL ontologies.
6. Protégé 4.2.0 or higher version, it is a free, open source ontology editor and knowledge-based framework.

3.2. Pattern-based HMI design

The design of the interface complies with user-centred design (UCD) approach as introduced in the section 2.1. UCD methodologies are being developed by HMI designers to create a systematic approach to their activities ([43], [44], [45]). Throughout these methodologies HMI design patterns play an important role in exploring and specifying the interaction between the human user and the computer. They enable to reuse concrete

solutions through appropriate descriptions. The main goal of a pattern-based HMI design method is to reuse HMI design knowledge that was successfully used in multiple applications.

There are thirty patterns identified for different domains in the ASTUTE project, either from literature review or from development process of the project. However, some of the patterns are similar and some of them are not necessary for the production management domain. After a detailed review in the early stages of the project, 18 patterns were selected for manufacturing domain. Below are the descriptions of three of the selected patterns as an example. A complete list of chosen pattern is in Appendix 1.

Table 3. 1 *Descriptions of patterns*

Pattern name	**Problem**	**Solution**
Context Adaptation [46]	How can interaction (input and output) be adapted to the current situation, environment and user without the user having to perform additional interaction steps?	The system should analyse as much assured context information as available to setup system configuration autonomously.
Non-disruptive Notification [47]	An event occurs, but the user's attention has to be directed to an important, potentially safety critical activity. The user wants to decide when to retrieve new information. You are developing an application which involves user notification about events. The application scenario (either the task or the situation) involves safety critical activities or requires high concentration.	Use output modalities of high spatial selectivity, such as graphics or (for blind users) haptic. The information should be displayed at a consistent place. If there might be a lot of information to be notified about, the user should be given a standardised command for retrieving it. The presence of new information should be indicated at a consistent place on the display.
Proximity Activates /Deactivates [48]	What: This pattern is for performing the simplest of all gestures, requiring only the presence of a person (or object) without any direct body contact. Use when: Use Proximity Activates/Deactivates to trigger simple on/off settings, such as lighting, display changes, sound, and other environmental controls.	How: The presence of a person can be detected with a variety of means: camera, motion detector, infrared "tripwire," pressure sensor, or microphone.

In the next section, the adaptation effects are designed based on the selected patterns.

3.3. Adaptation effects

As mentioned in section 2.21, adaptation effects refer to the differentiation behaviour of the system for different users and contexts. They are the major features that the visualization model is intended to achieve. Based on the pattern design methodology and the situational ETA triad, several adaptation effects are proposed in the initial phase of the design. These adaptation effects can be classified based on the nature of the patterns. In each pattern, they can be classified based on the domain in the situational ETA triad. The proposed adaptation effects for the system are introduced bellow.

1. Pattern name: ContextAdaptation

    - Situational ETA triad Domain: User domain

      Adaptation effects:

        - Set the system font according to user's preference.

        - When user is tired, the system font is enlarged (It implies the information should be more straight-forward, using less words)

        - When user is relaxed, the system font can be smaller. (more words can be used to present the information)

    - Situational ETA triad Domain: Environment domain

      Adaptation effects:

        - When ambient light is dark, use dark background, white font colour. Whereas if ambient lightness is bright, use bright background and black font colour.

2. Pattern name: Non-disruptive Notification/Redundant output/ Enrich sound notifications/ Audio visual workspace

    - Situational ETA triad Domain: Task domain

      Adaptation effects:

        - When the new event notification is less important than the user's primary task, the event notification will not be triggered; instead,

the interface only shows the number of non-triggered notification. No sound notification.

- When the new event notification is more important than the user's primary task, the event notification will disrupt the user. The user needs to choose if he wants to change the primary task or continue previous task by selecting "Accept" or "Remind later". Sound notification triggered.

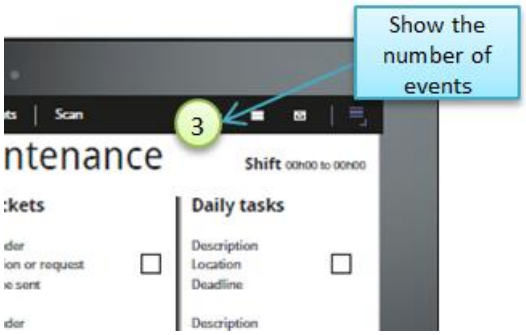


Figure 3. 4 *Notifications for less important events*

3. Pattern name: Alert/Important message

   - Situational ETA triad Domain: Task domain

     Adaptation effects:

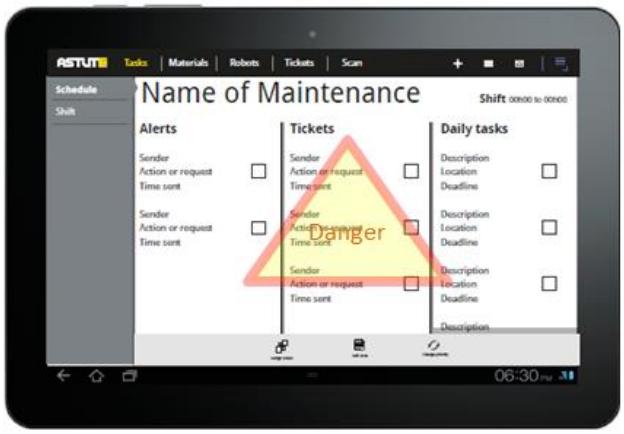
     - Alert and important message have the highest priority. They will disrupt the user no matter what he is doing. Different sounds will be used.

4. Pattern name: Proximity Activates /Deactivates

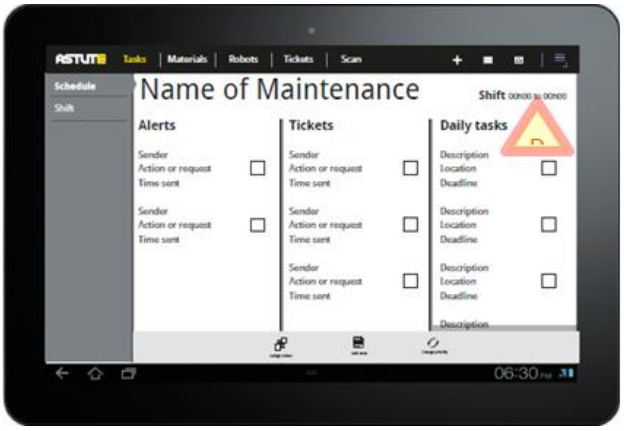
   - Situational ETA triad Domain: Environment domain

     Adaptation effects:

     - If the user has not completed his security training, when he comes close to the dangerous working area, a transparent icon will appear in the background. The user with sufficient security training will see a smaller one.



(a) Large safety icon



(b) Small safety icon

Figure 3. 5 *Different ways of showing safety information*

5. Pattern name: Multiple ways of input

    - Situational ETA triad Domain: Device domain

      Adaptation effects:

        - Provide all possible modalities that the device has for the user to choose.

6. Pattern name: Composed Command/ Multimodal instruction.

    - Situational ETA triad Domain: Task domain

      Adaptation effects:

        - List the steps of the specific task.

- For helping users get familiar with the system, use multi-modal instructions for the steps.

7. Pattern name: Audio visual presentation.

   - Situational ETA triad Domain: User domain

     Adaptation effects:

     - Use this pattern to present information only when the user is free (Particularly for the manager to check the chart of the analysis on the production line, KPIs). When the user is tired or stressed, use other straightforward pattern.

Figure 3. 6 *Audio visual presentation*

8. Pattern name: Multiple Alerts dissemination.

   - Situational ETA triad Domain: Task domain

     Adaptation effects:

     - When multiple alerts appear at the same time, put the alerts of the same type in one group, showing the number. When the user clicks the alert, it extends to show all the alerts.

Figure 3. 7 *Multiple Alerts dissemination*

9. Pattern name: Redundant output/ Enrich sound notifications/ Audio visual work-space/ Complementary modalities for alarms.

    - Situational ETA triad Domain: Task domain

      Adaptation effects:

        - Different alarm sounds will be assigned to different kinds of events
        - Assign different colours and icons for different type of alerts

10. Pattern name: Spatial representation

    - Situational ETA triad Domain: Environment domain

      Adaptation effects:

        - Location information will be captured.

11. Pattern name: Metaphor/ Simulation

    - Situational ETA triad Domain: User domain

      Adaptation effects:

        - Depending on the role of the user, present Augment reality or 3D model.
        - Depending on the role of the user, simulation option could be available.

12. Pattern name: Warning (as dismissible)

- Situational ETA triad Domain: Task domain

  Adaptation effects:

  - If a non-disruptive message is not urgent but will be critical, it becomes a warning message, which will show text or sound without interrupting the user.

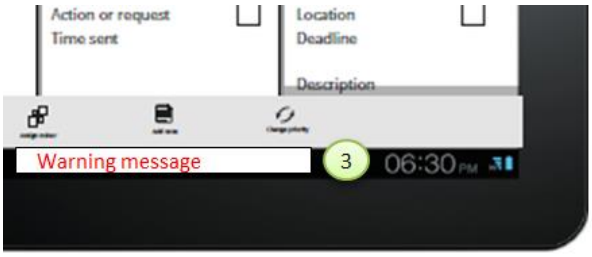


Figure 3. 8 *Warning messages*

13. Pattern name: Mode switch

    - Situational ETA triad Domain: User domain

      Adaptation effects:

      - An option is available for users to switch off the adaptation.

14. Pattern name: Detail levels/ Abstract simplification

    - Situational ETA triad Domain: User domain

      Adaptation effects:

      - For the same task, manager and maintenance will see different level of detail.
      - The team leader first sees a more abstract representation of the site and then he can dig into the details.
      - When the user is tired, the font size is bigger, and the information will be shown using less words.

15. Pattern name: Layer

    - Situational ETA triad Domain: Task domain

Adaptation effects:

- Use more layers for a large amount of information.

16. Pattern name: Task management

    - Situational ETA triad Domain: Environment domain

      Adaptation effects:

      - At certain time, the interface presents the general task, and leads the user to go through the routine tasks.

17. Pattern name: Log

    - Situational ETA triad Domain: Task domain

      Adaptation effects:

      - Event history log.

3.4. Ontology-driven user modelling

3.4.1. User model

The user model needs to fulfil the requirements of the intended adaptation effects. In other words, the user model must provide sufficient information that is needed to realize the adaptation effects. However, a huge user model with redundant information is not always necessary. The user model is designed according to the user modelling theories described in the section 2, the ontology editor Protégé serves as a platform for creating the user model. The IRI of the ontology model is set as http://fi.tut.astute/userprofile. In terms of semantic modelling abstraction, the user model contains two levels: domain and application knowledge.

In the two level of abstraction, the domain knowledge contains the generic information about the necessary dimensions in the user modelling. It includes information on users, environment, tasks and their devices. In general, all the domain concepts are classified into five kinds of abstract concepts: *DeviceDomain*, *EnvironmentDomain*, *TaskDomain*, *UserDomain* and *InterfaceDomain*. One benefit of this classification is that it is easier to understand and manage the concepts in the user model. Another reason is that it complies with the methodology introduced in the situational ETA framework and user modelling dimension in section 2.2. In *DeviceDomain* class, included are class *Device*, *Mal-*

function, *Modality* representing respectively a device that is being used by a user, the device's possible malfunctions and its supported modalities. The modalities that a device supports could either be an *InputModality* or *OutputModality*. These three aspects of device domain are selected because device is the fundamental carrier of the user interface application, and the system will support multimodalities of the device. Since the varying mobile devices that a user uses may have different availability of modalities, and in different situations, there might be restrictions or preference on certain modalities, the user model needs to provide the knowledge to facilitate the decision making process. Figure 3.9 depicts the domain level model.

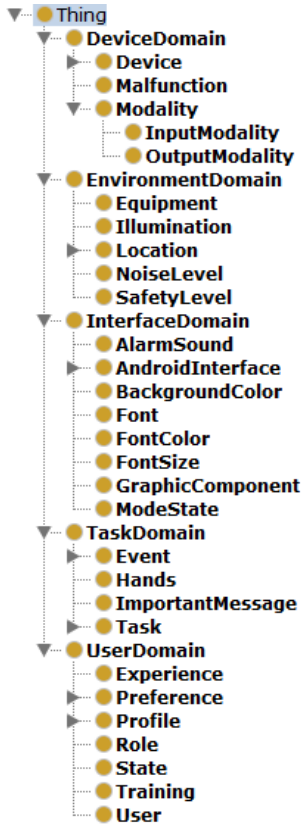


Figure 3. 9 *Upper-level domain concepts in the user model*

Class EnvironmentDomain contains class *Equipment* that represents equipment existing in the working environment of a user. Because our application is in the industry domain, it involves the interaction between workers and the equipment in the workshop. And also most of the system events and messages are related with the equipment. The knowledge of the existing equipment and their relevant information become necessary in terms of context-awareness. The class *Location* represents the physical location where a device is being used. This class is further specified as *Home*, *Office*, *Outdoor* and *Workshop*. Due to the nature of mobile-based system, the location of the user could change rapidly and has implication about where the user is and what kind of task he is working on. So the model of location provides possible places where the user can be.

Once the location information is captured at run-time, some other parameters should be obtained. For example, the *class Illumination, NoiseLevel and SafetyLevel* represent the environmental parameters-ambient brightness, noise and safety of a location. Such environmental parameters characterize the features of a location. Based on the environment parameters, the application is expected to provide user with adaptation to improve the user experience.

In the *TaskDomain*, system events are represented by the class *Event*, and an event could lead a user to a certain task which in here is represented by the class *Task. PrimaryTask* refers to the primary task that a user is performing with the device. System events include any events in the application domain of which the user should be aware. Different events may be shown in different ways or in a non-disruptive way. Therefore, information about the system events helps to filter and postpone the notification. Based on the notification of system events, the user may start to perform a certain task. System event closely relates the task of the user.

UserDomain contains the concepts about a user. Users are the major participant in the interaction. A comprehensive knowledge about the user is significant for the user centre design methodology. First, the class *Role* represents the role that a user has in the company. Users possessing different roles have varying tasks, interests and even thinking logic. By differentiating the user's role, the application can provide more relevant content and interaction behaviour for the user. The class *Experience* represents how much a user is familiar with the HMI system. A user's previous experience on the system is indicative for determining how much detail the system should provide the user, because some details could be seen as redundant by old users, but as necessary for new users. The user's preference and general profile are represented by the class *Preference and Profile.* User's preference is a direct indication about what the user likes. The class *State* represents a user's temporal mental state. Since we would like to take user's cognitive state into account in the interaction, it is necessary to model a user's mental state. *Training* represents what kind of training a user has completed already. The user may have completed some training before he starts to use the application. By modelling the user's previous training history, more information of the user can be understood.

InterfaceDomain composes the elements used in the interface application. For example, the class *BackgroundColour* represents the background colour of the user interface. The purpose of the visualization model is to provide the definition of the user interface, hence first it must have understanding in what components the user interface consists of.

Only modelling the domain concepts is not enough because domain knowledge includes modelling the relationship between the concepts. The concepts described above are related by the properties defined in the user model as shown in Table 3.2.

Table 3. 2 *Properties between domain concepts*

Property name	Specifies	Domain	Range
hasMalfunction	A device has a malfunction	Device	Malfunction
isInLocation	A device is in a location		Location
supportsInputModality	A device supports an input modality		InputModality
supportsInputModality	A device supports an output modality		OutputModality
usesInterfaceComponent	A device uses application interface components		InterfaceDomain
hasEquipment	A working location has equipment	Location	Equipment
hasIllumination	A location has ambient light level		Illumination
hasNoiseLevel	A location has noise level		NoiseLevel
hasSafetyLevel	A location has safety level		SafetyLevel
leadsToTask	A event relates to a task	Event	Task
hasDevice	A user has a device	User	Device
hasExperience	A user has experience with the application		Experience
hasPreference	A user has preference		InterfaceDomain
hasProfile	A user has user profile		Profile
hasRole	A user has a role		Role
hasState	A user has a mental state		State
hasTask	A user has a task		Task
hasTraining	A user has completed a training		Training
hasScreenSize	A device has screen size	Device	float (built-in datatype)
has Age	A user has age	User	Int (built-in datatype)

hasInputModalityState	A device has input modality state	Device	Int (built-in datatype)
hasOutputModalityState	A device has output modality state	Device	Int (built-in datatype)

Next, application knowledge is modelled by enriching the domain knowledge and creating application-specific concepts, instances and SWRL rules. In the *DeviceDomain*, device modalities are modelled as the instance of the class *InputModality* and *OutputModality as* demonstrated in Table 3.3

Table 3. 3 *Instances representing modalities of device*

SubClass Of	Instance	Modality
InputModality	bCameraInput	The rear camera
	fCameraInput	The front-facing camera
	hapticInput	Input by using accelerometer
	keyboardInput	External keyboard input
	lightSensorInput	Ambient light sensor input
	nfcInput	Near field communication Input
	screenInput	Touch screen input
	voiceInput	Speaker input
OutputModality	flashOutput	Flashlight
	hapticOutput	Viberation
	nfcOutput	Near field communication Output
	screenOutput	Screen display
	soundOutput	Loudspeaker

Device is specified into *MobilePhone* class and *Tablet* class because the application is for these two types of mobile devices. In each of these two abstract types of devices, the

concrete device type is modelled by using the unique model number. It represents a unique type of devices. For example, the model number of Google Nexus 7 tablet is Nexus7asus, so class *NEXUS7ASUS* represents this unique type of devices. In addition, the features of this type of devices are defined by adding class expression to the device type classes. For instance, the modalities supported by the Nexus 7 are modelled as

(supportsInputModality value fCameraInput) and (supportsInputModality value lightSensorInput) and (supportsInputModality value nfcInput) and (supportsInputModality value screenInput) and (supportsInputModality value voiceInput) and (supportsOutputModality value nfcOutput) and (supportsOutputModality value screenOutput) and (supportsOutputModality value soundOutput) and (hasScreenSize value 7.0f)

Further, for the specific physical device being used, it is modelled by using the device's unique ID, as the instance of device type class. As shown in Figure 3.10, for example, assume "nexus03", "nexus02" and "nexus01" are the unique ID of three google Nexus 7 tablets so they are the instances of the class *NEXUS7ASUS* which represents the model number of the tablets.

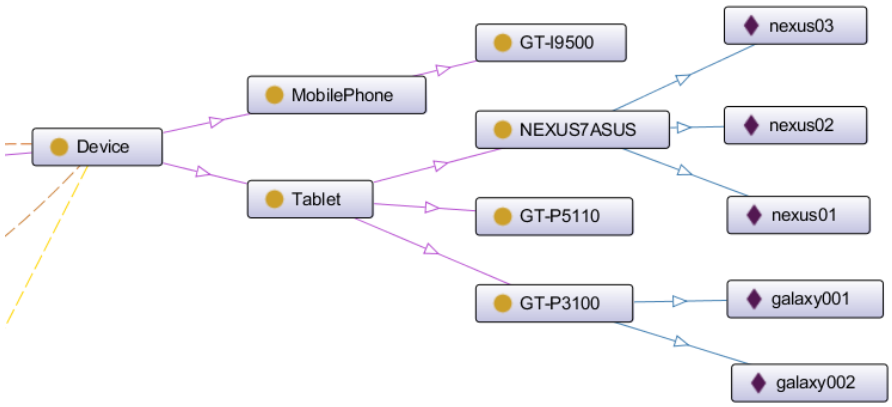


Figure 3. 10 *Modelling concrete devices and device type*

Once a physical device is instantiated in the device type class, it inherits the device specifications defined in the device type class. The device specifications are pre-defined by asserting class expression in the device type class. For instance, the class expression shown in figure 3.11 defines that all the nexus 7 devices support the front-facing camera, an ambient light sensor, near field communication, display screen, speaker, touch screen, loudspeaker and their screen size is 7 inch.

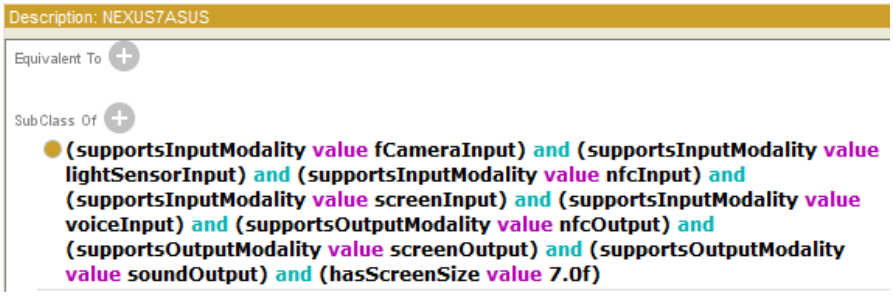


Figure 3. 11 *Class expression of device type*

In the Environment domain, the application knowledge is specified by creating instances for the concept classes according to the real situation of the production line. The production line of FAST laboratory has twelve robot cells performing different functions. They are connected by a conveyor. Hence we create twelve instances (cell1, cell2 … cell12) in Equipment class to represent the cells and one instance (conveyor) represents the conveyor. Each cell contains a major device. Specifically, except cell 7 contains a buffer, each of the other eleven cells has a Sony robot. Hence in the environment model, it is defined as for example, *cell1* (instance of Equipment class) *hasEquipment* (object property) *robot1* (instance of Equipment class). In the Illumination class, three instances (averageLight, bright and dark) define three range of brightness, here by brightness we mean the environment illumination. The location of the device can only be associated one of the three instances, depending on the output of the ambient light sensor. In the same manner, Noiselevel has three instances (noisy, normalNoise and quiet) to define the noise level of the location whereas in SafetyLevel class, safetyHigh, safetyLow and safetyUnknown instances are used to pre-define the safety of the location. Since one location can only relate to a single environment parameter, we create the object restriction for the Location class by asserting all the instances of Location class:

hasIllumination **only** Illumination
hasNoiseLevel **only** NoiseLevel
hasSafetyLevel **only** SafetyLevel

In the InterfaceDomain, we specify the AndroidInterface Class because we are focusing on the development of Android platform. Based on the principle of Android programming, we define Activity class and View class. These two classes represent the components in the Android application. For example, there will be a screen to show the robot error in the android interface, this screen is actually an android Activity called *astuteNamRobotErrorActivity*. For convenience and consistence, we create an instance with the same name in Activity class in the ontology. The same principle applies to View class. In the View class, there are Button, ListView, TextView classes. They represent the layout components in the Android application. In these classes, we define the components that we expect to be adaptive as individuals. For instance, *instruction_button* is an individual in the *Button* class. It could have parameters such as “colour” and “size”.

Then the actual interface will adapt the visualization of this button based on the parameters.

In the Task domain, *Event* class is extended to *NewEvent* class, *PendingEvent* and *TriggeredEvent* such that a system event notification can be classified to different groups depending on the priority. In the *NewEvent* class, the following classes are created to model the event types that the system has, they are the possible errors in the production system:

- *BufferStuck* : When the fork of the buffer releases a pallet in the conveyor, the pallet is not always well placed and therefore it can be stuck there.
- *CellDoorOpen:* Door of the cell is opened.
- *ConveyorStopped:* Conveyor is stoped unexpectedly
- *EndeffectorFailure*I*:* End effector failed to catch the pen
- *InkEmpty:* Ink is about to run out
- *PalletRetrieved:* A pallet has been retrieved.
- *PalletStuck:* A pallet is stuck.
- *QualityError:* the quality error of each drawing produced by the robots.
- *RobotStuck:* A robot is stuck.

3.4.2. SWRL rules

Since OWL 2 is not able to express all kinds of relationships of the concepts, SWRL rules are designed to enrich the user model. More importantly, the SWRL rules are designed to provide rules to realize adaptation effects at run-time. Table 3.4 describes the SWRL rules defined in the user model. For example, if the priority of a system event NewEvent(?evt) is higher than that of the primary task PrimaryTask(?pt), the system event is asserted as a triggered event TriggeredEvent(?evt). This relationship is expressed by the following SWRL rule:

NewEvent(?evt), PrimaryTask(?pt), hasPriority(?evt, ?pri), hasPriority(?pt, ?pript), greaterThan(?pri, ?pript) -> TriggeredEvent(?evt)

Table 3. 4 Examples of the SWRL rules defined in the user model.

Usage	**SWRL rules**
Define the route map according to the user's role	RobotStuck(?evt), Role(operator), User(?u), hasRole(?u, operator) -> usesMap(?evt, "operator_path.png"^^string)
Define the layout parameters for the activity	Button(instruction_button), Button(route_button), Device(?d), Outdoor(?l), User(?u), hasDevice(?u, ?d), isInLocation(?d, ?l) ->hasParameter(instruction_button, "\"instr_btn_color\":\"default\""^^string),hasParameter(route_button,

	"\"route_btn_color\":\"green\""^^string),hasParameter(route_button, "\"route_btn_vsblty\":\"true\""^^string)
Define the system font size	• CurrentUser(?u), Device(?d), FontSize(largeFont), State(fatigue), hasDevice(?u, ?d), hasState(?u, fatigue) -> usesFontSize(?d, largeFont)
Define the system font	• CurrentUser(?u), Device(?d), Font(?f), hasDevice(?u, ?d), prefersFont(?u, ?f) -> usesFont(?d, ?f)
Associate the system event to the task of the user	• AlarmSound(alarmSound3), BufferStuck(?evt), Task(taskBufferStuck) -> hasAlarmSound(?evt, alarmSound3), leadsToTask(?evt, taskBufferStuck) • AlarmSound(alarmSound1),PalletRetrieved(?evt), Task(taskPalletRetrieved) -> hasAlarmSound(?evt, alarmSound1), leadsToTask(?evt, taskPalletRetrieved)
Define background colour, font colour, and screen brightness	• BackgroundColour(white), CurrentUser(?u), Device(?d), FontColour(black), Illumination(averageLight), Location(?l), hasDevice(?u, ?d), hasIllumination(?l, averageLight), isInLocation(?d, ?l) -> usesBackgroundColour(?d, white), usesFontColour(?d, black), usesScreenBrightness(?d, 70)
Define the GUI component (hazard icon) and decide if it will be hidden, or shown with a small one or a bigger one.	• CurrentUser(?u), GraphicComponent(dangerIcon), Location(?l), SafetyLevel(safetyHigh), hasDevice(?u, ?d), hasSafetyLevel(?l, safetyHigh), isInLocation(?d, ?l) -> usesGraphicComponent(?d, dangerIcon), hasComponentSize(dangerIcon, "hidden"^^string)
Determine that if the importance of the new event is not greater than the primary task, the event becomes a pending event.	• NewEvent(?evt), PrimaryTask(?pt), hasPriority(?evt, ?pri), hasPriority(?pt, ?pript), lessThanOrEqual(?pri, ?pript) -> PendingEvent(?evt)
Determine that if the importance of the new event is greater than the primary task, the event becomes a triggered event.	• NewEvent(?evt), PrimaryTask(?pt), hasPriority(?evt, ?pri), hasPriority(?pt, ?pript), greaterThan(?pri, ?pript) -> TriggeredEvent(?evt)
Define the visibility of the instruction according the experience level of the user	Experience(vintage), User(?u), (PrimaryTask or SecondaryTask)(?t), hasExperience(?u, vintage) -> hasInstructionVisibility(?t, true)
Define the layout parameter	Button(instruction_button), Button(route_button), Device(?d), Home(?l), User(?u), hasDevice(?u, ?d), isInLocation(?d, ?l) -> hasParameter(instruction_button, "\"instr_btn_color\":\"default\""^^string), hasParame-

	ter(route_button, "\"route_btn_color\":\"green\""^^string), hasParameter(route_button, "\"route_btn_vsblty\":\"true\""^^string)

3.5. Visualization model design

The design of the visualization model is compatible with the overall architecture of the ASTUTE project. Figure 3.12 shows the components of the visualization model. It runs on the server along with other codes of the server side application. It consists of an interface definer, context manager, a pre-defined OWL user model and a native RDF database. Interface definer is used to manage the manipulation of the ontology-driven user model and retrieve HMI definition for deployment module. The OWL user model introduced in section 3.4 is stored on the server as a template. At the time a new user account is registered in the system, the HMI definer will load the OWL file and create a specific user model for the new user. This user model is stored in a native RDF database on the server by using Jena TDB. Another way to store the user model is using Jena SDB, which is also a component of Jena for RDF storage. The difference is Jena SDB is based on SQL database so that it allows storing the user model to a remote database server. During the design process, both of the two storages were tested. The results showed in the system configuration, TDB provides faster speed for saving the user model and querying the user model than does SDB. Hence Jena TDB is selected as the user model storage on the server. Context manager is responsible for updating the user model according to the contextual information at run time. The user model contains all the context information in it but it has to be updated according the real situation in order to generate correct HMI definition for the situation.

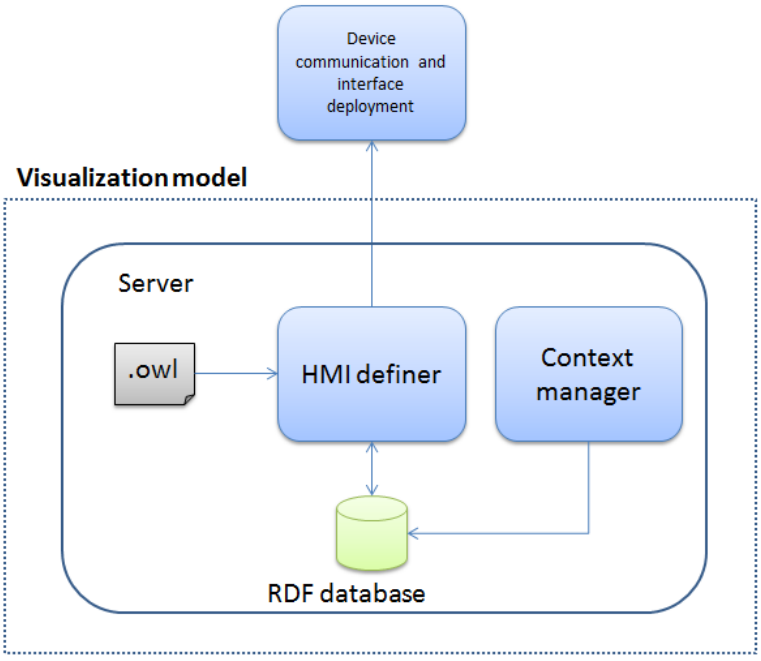


Figure 3. 12 *Functional diagram of Visualization model*

To realize the visualization model described above, two Java classes - HMIdefiner, ContextManager are created.

HMIdefiner Class:

HMIdefiner is designed to create the user model for the user and query the user model to provide real-time HMI definition for the user. It contains several methods as described in Table 3.5 below. There are four variables in the data field including *base_iri*, a string defining the ontology IRI of the user model, *currentUser* (String), *currentDevice* (String) and cur*rentDeviceType* (String).

Table 3. 5 *Method summary of HMIdefiner class*

Modifier and Type	Method and Description
Void	createNewUser(String newUserID, String userRole) Create a user model in the RDF datebase. **Parameters:** newUserID – A unique user ID for the user mdel to be created. userRole – The role of the user in the application domain. For example, “operator”, “maintenance” and “supervisor”.
String	getHMIDefinition() Answer a Json formatted message defining the font, font size, font colour, backgroundcolour and brightness of the display screen.
String	getModalityState() Answer a Json formatted message defining the volume of loudspeaker, activation/deactivation of haptic.
String	getGraphicComponent() Answer a Json formatted message defining the graphic components that the device should show.

String	getEventMsg() Answer a Json formatted message defining the event messages that the device should show and how the message are shown.
Void	setDefinitionTarget(String userID, String device) Set the user and the device that the HMI definition will be retrieved for. **Parameters:** userID – A unique user ID of the user model to be load. device – The unique serial number of the mobile device.

The realization of the methods is briefly explained in the following section. The methods *createNewUser(String newUserID, String userRole)* makes use of Jena TDB api to load the OWL file and store it in a RDF database as the user model for the user. The procedures are that the dataset folder is set on the server, read the OWL file and store the OWL file in the dataset using the user ID as the its name. After the user model is created, a user instance and a corresponding role instance are created in the user model. Programme 3.1 shows the procedures in this method. Note in order to deal with the concurrency of transaction on dataset, it is necessary to end the transaction once the update of the model is committed.

```
// Set the dataset path as "/home/fast/tdb” on the server
String directory = "/home/fast/tdb";
Dataset dataset = TDBFactory.createDataset(directory);
// Enable WRITE transaction of dataset
dataset.begin(ReadWrite.WRITE);
try {
   // Create user model
   Model model = dataset.getNamedModel(newUserID + "profile");
   // Load the owl file and store it in the user model
   String source = "c:/tmp/userprofile0207.owl";
   FileManager.get().readModel(model, source);
   // Create ontology model from the user model
   OntModelSpec ontSpec = new OntMod-
elSpec(PelletReasonerFactory.THE_SPEC);
   OntDocumentManager mgr = new OntDocumentManager();
   ontSpec.setDocumentManager(mgr);
   OntModel m = ModelFactory.createOntologyModel(ontSpec, model);
   // Create a user individual
   Individual newUser = m.createIndividual(base_iri + newUserID,
m.getOntClass(base_iri + "User"));
   ObjectProperty hasRole = m.getObjectProperty(base_iri + "hasRole");
   // Create a role individual
```

```
        Individual role = m.getIndividual(base_iri + userRole);
        newUser.setPropertyValue(hasRole, role);
        ObjectProperty prefersHand = m.getObjectProperty(base_iri + "prefersHand");
        // Persist the user model
        dataset.commit();
      } finally {
        // End the transaction
        dataset.end();
      }
```

Programme 3. 1 *Major procedures in method createNewUser*

setDefinitionTarget(String userID, String device) simply assigns value to the String variables *currentUser* and *currentDevice*. In *getHMIDefinition()* method, the ontology model is loaded in the same way described early on. Then the HMI definition message is created by querying the ontology model. There are two ways of querying the ontology model, here the first way is shown in the Programme 3.2.

```
OntModel m = ModelFactory.createOntologyModel(ontSpec, model);
//Get the device instance
 Individual device = m.getIndividual(base_iri + currentDevice);
//Get the useFont object property
 ObjectProperty usesFont = m.getObjectProperty(base_iri + "usesFont");
 //Get system font by getting the property value
 String systemFont = device.getPropertyResourceValue(usesFont).getLocalName();
 //Get system font size
 ObjectProperty usesFontSize = m.getObjectProperty(base_iri + "usesFontSize");
//Get system font size
 String systemFontSize = de-
vice.getPropertyResourceValue(usesFontSize).getLocalName();
```

Programme 3. 2 *Query the ontology model*

This method will return a Json-format string containing the definition of system font, font size, font colour, background colour and brightness. One example is {"type":"dspParameter","font":"font_arial" ,"fontsize":"normalFont","fontcolour":"white","backgroundcolour":"grey","brightness": "30"}. We can see another way of querying the ontology model in the method get-ModalityState() in Programme 3.3. This method returns the definition of the status of loudspeaker and haptic. One example of the string answered by this method is {"type":"mdltyParameter", "volume": "30", "haptic": "0"}. It means the volume of the loudspeaker should be thirty per cent of the maximum volume and haptic is deactivated.

```
String mdltState = "{\"type\":\"mdltyParameter\", \"volume\": ";
OntModel m = ModelFactory.createOntologyModel(ontSpec, model);
    try {
      // Assign query string
       String queryString =
```

```
        "PREFIX : <http://fi.tut.astute/userprofile#> "
        + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
        + "ASK "
        + "WHERE { :"+currentDevice+" :supportsOutputModality :soundOutput.}";
// Create query instance
Query query = QueryFactory.create(queryString);
// Execute query
QueryExecution qe = QueryExecutionFactory.create(query, m);
Boolean results = qe.execAsk();

if (results) {
  // If the device supports outspeaker modality then make another query
   queryString =
         "PREFIX : <http://fi.tut.astute/userprofile#> "
         + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
         + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
         + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
         + "SELECT ?volume "
         + "WHERE { :" + currentDevice + " :hasVolumeState ?volume.}";
   query = QueryFactory.create(queryString);
   qe = QueryExecutionFactory.create(query, m);
   ResultSet results2 = qe.execSelect();
   while (results2.hasNext()) {
      QuerySolution querySolution = results2.nextSolution();
      String volume = querySolution.getLiteral("?volume").getString();
      mdltState = mdltState
            +"\""+ volume + "\", \"haptic\": ";
   }
```

Programme 3. 3 *Another way of querying the ontology model*

getGraphicComponent() and *getEventMsg()* follow the same procedures but only query different informations from the user model. One example of the message answered by *getGraphicComponent()* is {"type":"graphComponent", "component":[{"name":"dangerIcon","size":"small"},]}. It means the device should show a small “danger” icon on the screen.

ContextManager Class:

ContextManger is designed to update the user model when the instances need to be set or be changed. For instance, when the user uses a different device to log in the system, this class can be used to update the device being used in the user model. It contains several methods as described in Table 3.6 below. There are four variables in the data field including *base_iri*, a string defining the ontology IRI of the user model, *currentUser*

(String), *currentDevice* (String), *m* (OntModel) and ds (Dataset). The source of this class is shown in Appendix 2.

Table 3. 6 *Method summary of ContextManager class*

Modifier and Type	Method and Description
void	addDevice(String userID, String deviceID) Set the device in the user model. **Parameters:** userID – A unique user ID to which the device belongs. deviceID – A unique device serial number.
void	addEvent(String eventName, String equipment) Add a system event in the user model. **Parameters:** eventName – A system event name. equipment – An equipment name.
void	addUserTraining(String trainingName) Add the name of training completed by the user in the user model. **Parameters:** trainingName – the name of the training.
void	setCurrentUser(String userName) Set the user in the user model. **Parameters:** userName – the name of the user.
voice	setDeviceLocation(String location) Set the location of the device in the user model. **Parameters:** location – the name of the location.

void	setIllumination(double light) Set the ambience lightness of the location in the user model. **Parameters:** light – The value of the ambience lightness from the sensor.
void	setNoiseLevel(double noiseValue) Set the noise value of the location in the user model. **Parameters:** noiseValue – The value of the ambience noise from the sensor.
void	setUserExperience(String experience) Set the user's familiarity level on the system. **Parameters:** experience – The user's familiarity level.
void	setUserFont(String font) Set the user's preferred font. **Parameters:** font – The user's preferred system font.
void	setUserRole (String role) Set the user's role. **Parameters:** font – The user's role.
void	setUserRole (String role) Set the user's role. **Parameters:** font – The user's role.

4. IMPLEMENTATION OF THE VISUALIZATION MODEL

4.1. Test bed for the implementation of the model

The production monitoring system is designed for FASTory line at Tampere University of Technology. Hence the adaptivity feature offered by the visualization model should be supportive for the production activities of this system. This pallet-based production line consists of twelve workstations; ten assembly stations, one loading station and a buffer station. Each of the assembly stations contains a robot cell and a conveyor. The workstations are connected by the conveyors so that the pallets can travel all along the stations and carry the assembly components. The robots perform the operation in each cell. Figure 4.1 demonstrates the physical configuration of the FASTory line.

Figure 4. 1 *FASTory line*

The production line is used for assembling electronic components for cellphones. The production monitoring system is used to monitor and facilitate the production activities associated with this line such as system errors, daily tasks and productivity of the work-stations. For personnel, three roles are selected to be involved in the production activity. They are supervisor, maintenance and operator, which are the most common roles in the

production industry. And they have very different information needs and tasks to perform. The FASTory line as the test bed for the implementation of the model is expected to demonstrate why the specific adaptation effect is provided by the visualization model and how they are presented to the users involved in the FASTory line. For instance, the operator and maintenance may have different information of interest. The end effector of the robot in this line is a pen which periodically needs to re-fill the ink. Once the ink is finished, the operator needs to be notified while the maintenance does not need this type of information. So the visualization model decides to send this notification only to the operator.

4.2. Integration of the visualization model

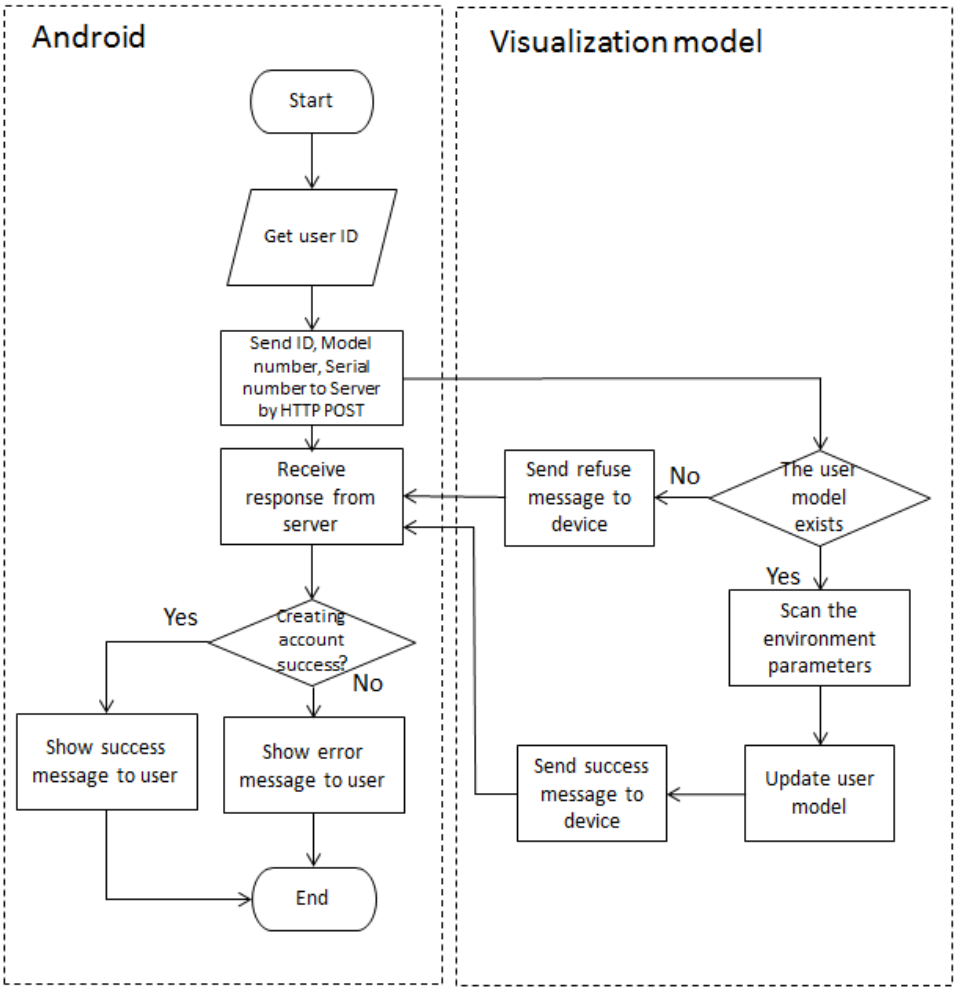


Figure 4. 2 *Procedure of creating user accounts*

This chapter describes how the visualization model is integrated with the android application that already existed for ASTUTE project, which had static interfaces for monitoring the production system, and the realization of its basic functionality, which provides adaptive interfaces to the monitoring system. The Log-in model is responsible for creating new user accounts and enabling the user to log in the system using existing ac-

counts. In FASTory Line, each user such as an operator can only have one user account for the monitoring system. The user model of the visualization model is used to identify the existence of the user account and manage the log-in behaviour of the user. The communication between Log-in model and visualization model is by httpurlconnection in the android api using http post method. Figure 4.2 demonstrates the procedure of creating new user accounts. The procedure of creating a new user model by the visualization model was described in section 3.5. So here we only show how to send data to the visualization model.

```
// Create a new HttpClient and Post Header
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost(NEW_USER_REQUEST_URL);
try {
 // Adding data
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
nameValuePairs.add(new BasicNameValuePair("UserID", userId));
nameValuePairs.add(new BasicNameValuePair("Role", role));
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
// Executing HttpPost
// Check for response if it succeeded
HttpResponse response = httpclient.execute(httppost);

StringBuilder sb = new StringBuilder();
String line = null;
try {
BufferedReader reader = new BufferedReader(new InputStreamRead-
er(response.getEntity().getContent()), 65728);

while ((line = reader.readLine()) != null) {
sb.append(line);
}
}
catch (IOException e) { e.printStackTrace(); }
catch (Exception e) { e.printStackTrace(); }
if(response.getStatusLine().getStatusCode() != HttpStatus.SC_OK){
msg = "Couldn't create new user";
}
} catch (ClientProtocolException ex) {
msg = "Couldn't create new user";
} catch (IOException ex) {
msg = "Couldn't create new user";
}
```

Programme 4. 1 *Sending data to server by HTTP POST*

For logging in the system, the user needs to input the user ID, and the android application will again use HTTP POST to communicate with the visualization model to check the existence of the user account, if there is the user account, the user will be led to the

main activity of the android application. In this process, the user device will also send its model number and serial number to the server as a unique identification of the device. Moreover, after the visualization model recognizes the user and device, it will scan the environment parameter such as ambience illumination, noise level and user state so that the user model can provide adaptive definition for the interface to adjust the brightness of the screen, sound volume and status of haptic. Figure 4.3 shows the procedure of logging process.

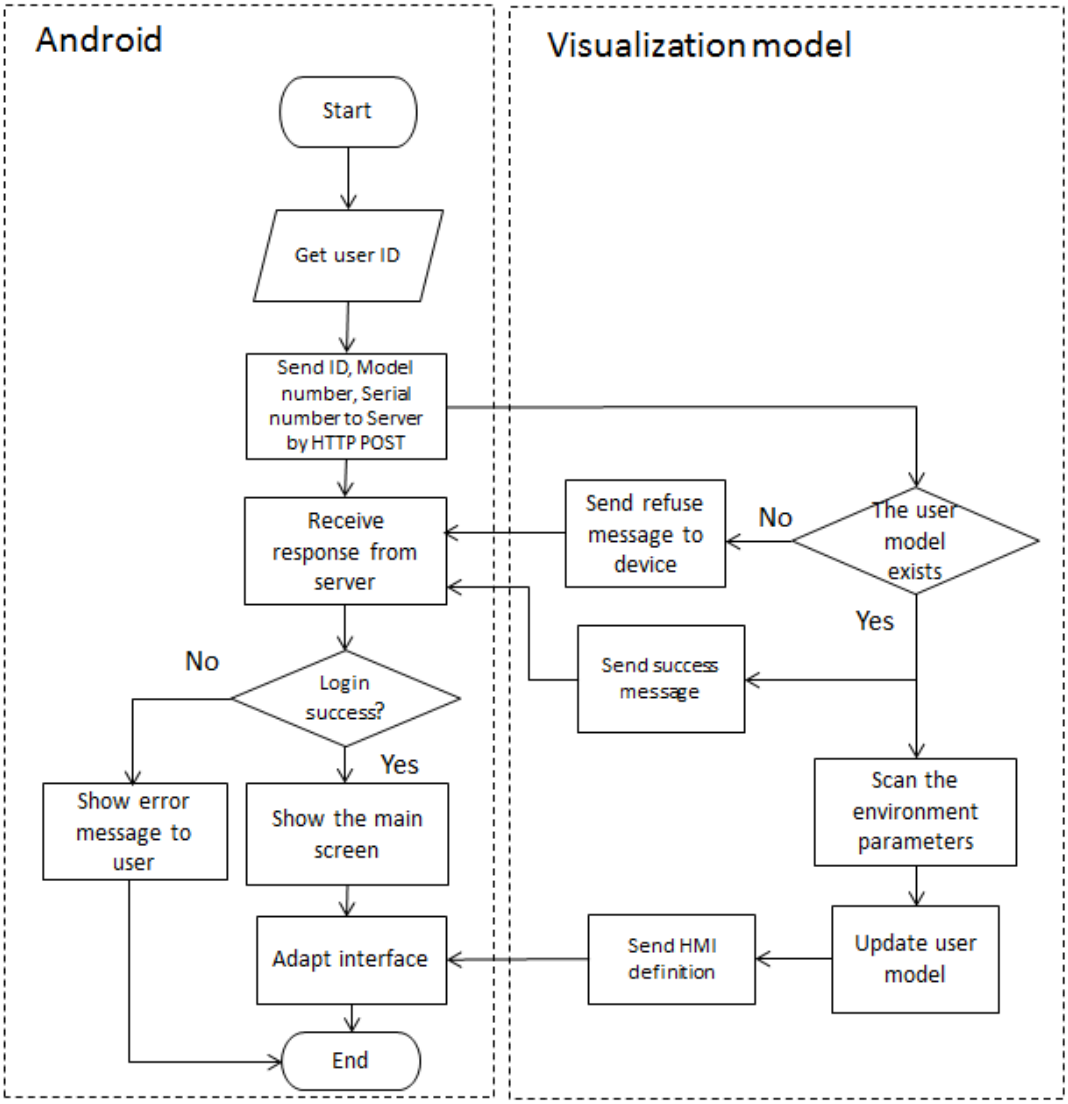
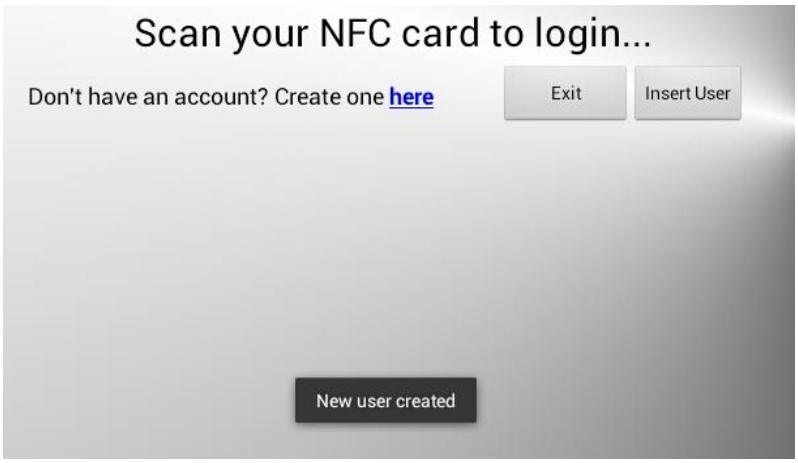


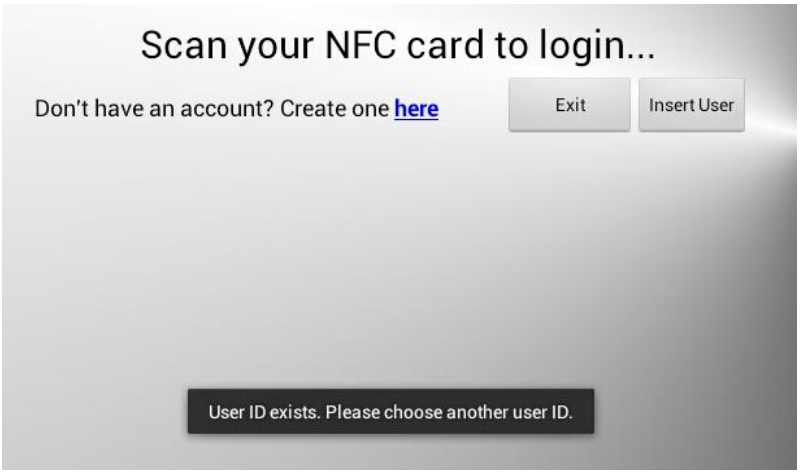
Figure 4. 3 *Procedure of logging in the system*

4.3 The implementation of the model

The figures that will be presented in this section are the proof of the implemented interfaces and its functionalities. For the first time that a user starts to use the monitoring system, he/she needs to create a new user account and provide the system with a unique user ID, and his role such as operator. If the user model is created successfully, a success notification will be shown on the screen. If the user ID already exists in the system, the user will be notified to choose another user ID. Figure 4.4 shows the notifications the user may see.



(a)



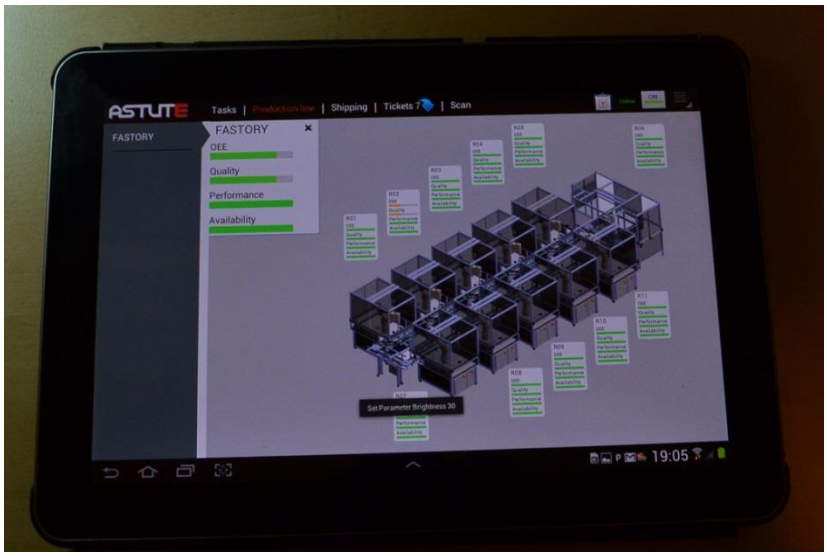
(b)

Figure 4. 4 *User account creation confirmation (a) and error notification (b)*

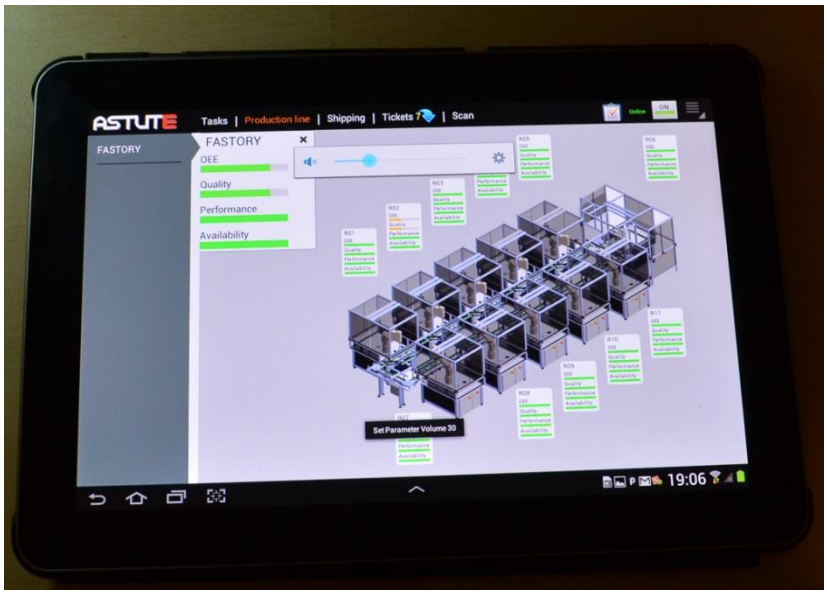
For login in the system, the user should enter his user ID and the main screen of the interface will appear. If the user ID is incorrect, a notification will be shown.

The users may work in different locations when they log in the system. Different locations may have different environment parameters. For example, the illumination in the workshop will change during the day, and typically the noise level of the office is much lower than that of the workshop. Based on the visualization model, the monitoring sys-

tem is able to provide the contextual adaptation feature to the user according to the context of use. First when the user logs in the system, the screen brightness, volume and haptic status will be adapted according to the current environment. For example, Figure 4.5 shows the difference of the brightness of the screen in dark environment and bright environment.



(a)



(b)

Figure 4. 5 *Adaptation of screen brightness*

At the same time, the volume of the device is also adjusted. In noisy environment, the volume will be higher and haptic is active. In quiet environment, the volume is lower and haptic is off.

Second, a danger icon may be visible to the user depending on where the user is. Once the user comes into the workshop, the icon will be visible in the low left corner. Also

depending on the user's training, this icon might be bigger. Figure 4.6 shows the visualization of the danger icon.

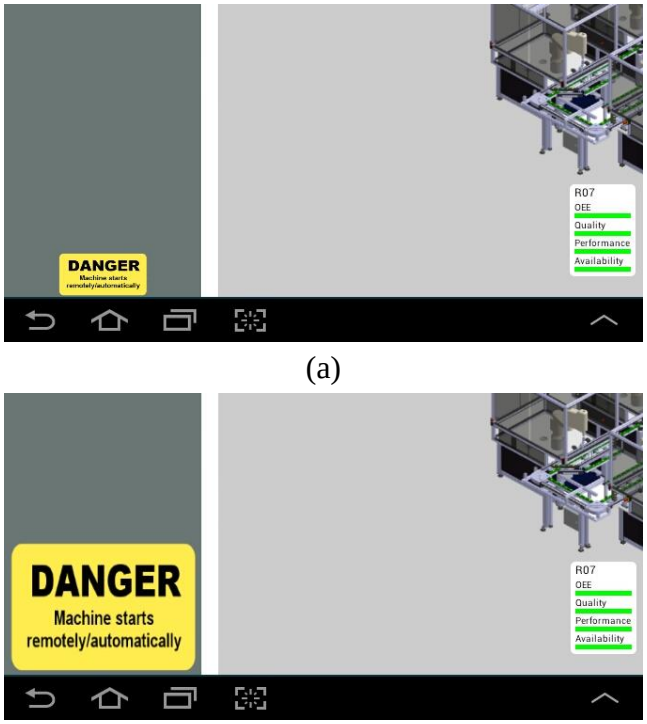


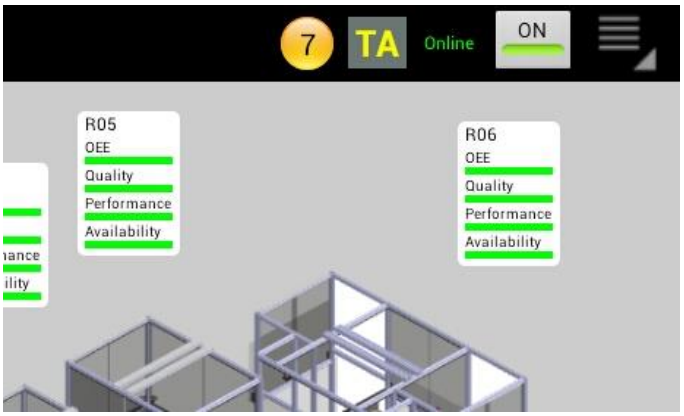
Figure 4. 6 *Danger icon for staff with safety training (a) without safety training (b)*

Third, when system errors happen in the production, the system will publish the system events to notify the users. System events and user task management correspond to the non-disruptive notification that is described in section 3.3. It allows the user to check the system events like errors in the production line. The user will also use it to track his current tasks. The principle is the user will be able to see different types of errors according to their roles. When a system event occurs in the production line, the system will send an event notification to the relevant user. The user can accept or cancel the notification. If the user accepts the notification, a task management screen will be shown as illustrated in Figure 4.7.

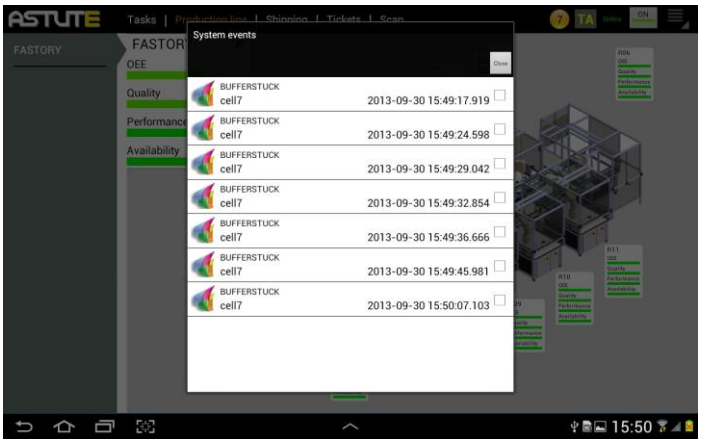
Figure 4. 7 *Task management panel*

The yellow round icon  on the top bar is an event indicator, which shows the number of system events that the user has. The  icon is a task indicator, which means the user has a primary task at the moment. The task management panel shows user's primary task, error information and tools for this error lick instruction, route to error, contact person and complete task button. The tools have some adaptation effects. For example the content of the instruction will be adapted to the type of error and also to the role of the user. The route to error tool will show the path to the error from the user's current place, if the user is already near the error place, this button will be invisible. For contact person tool, the contact person will be changed to the person who is related to the error event. The performance chart in the centre of the interface will also show the correct equipment related to the specific error. The toggle button on the top bar enables the user to switch off or on the adaptivity feature.

Meanwhile different types of events have different priorities, so have the user tasks. When the new event notification does not have higher priority than the current task, the user will not be interrupted by the new event notification. Instead, the new event notification only increases the number of event indicator on the top bar of the interface. In this way, the user can always focus on his primary task.



(a) Number of event indicator



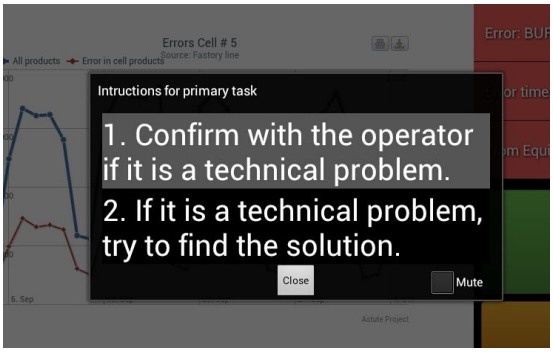
(a) Event list view

Figure 4. 8 *Event indicator on the top bar*

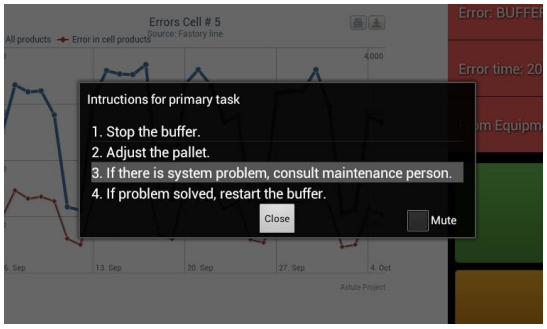
When the user accepts an event notification, he will have a specific task for this event until he completes this task. The task management panel is shown in Figure 4.4 (b). So the system can track what task the user is currently working on. This panel provides information about the user's primary task, such as system event name, event time and the equipment related to the event, so the user could know what happened in the production line. Several tools are provided to help the user handle the task. The tools include:

- Instruction – namely shows the instruction steps for the specific task. The instruction steps are predefined in the ontology user model. It not only differs according to the task, but also according to the user's role. In other words, employee with different roles will see their own instruction for the same task. If the tiredness of the user is detected, the font of the instruction will be enlarged. Figure 4.9 shows the described features. Furthermore, text-to-speech technology is adopted here, in adaptation mode, the instruction text will be read out by using TTS engine built-in the android os.

(a) Instruction for maintenance



(b) Instruction for maintenance tired



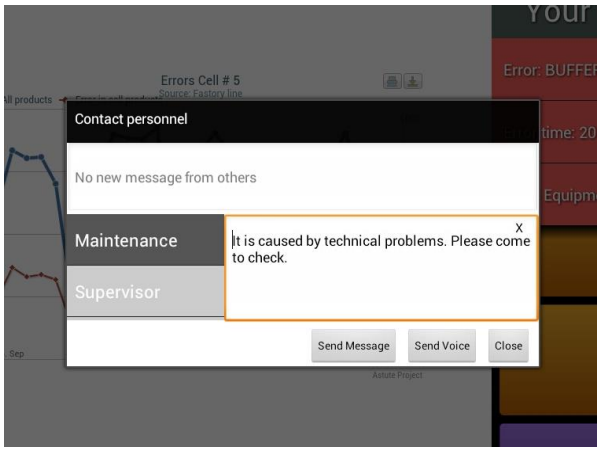
(c) Instruction for operator

Figure 4. 9 *Instruction of buffer stuck for different roles*

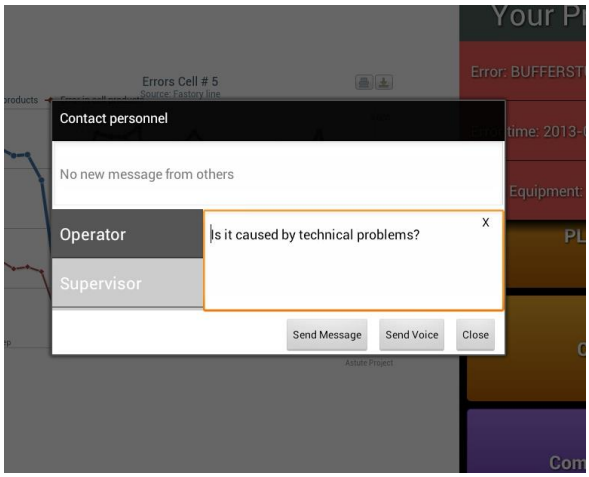
- Manual – provides the user manual of the equipment related to the event. For example, the user manual button will link to the pdf user manual of a SONY robot in case of a robot stuck event occurs.

- Route – provides a map service for the user to find the interior route of the premise to the equipment related to the error.

- Contact – provides contact person information in case the user needs to send messages to other workers. It changes the information according to user's role. For instance, the maintenance technician will see the contact information of the operator and supervisor. Moreover, template messages are defined for the user to help them communicate with others efficiently.



(a) For operator



(b) For maintenance

Figure 4. 10 *Different template messages for different roles*

- Controller configuration – Only the maintenance can see and use this tool. It is used to get access to the wireless controller of the production line through the local network if applicable.

- Complete task – Once the user has finished his task, he can use this button to complete the primary task. If the user has tasks in queue, the interface will automatically shows the next task for the user. Otherwise nothing will be shown.

5. CONCLUSIONS AND FUTURE WORK

With respect to the objectives stated in the section 1.2.3, the achievements of the thesis are as following:

1. Creation of an ontology user model representing the contextual information of the production system in the FAST-Lab, including the environment, device, user, and task. The user model contains sufficient information to provide adaptation effects for the interface.
2. Design of the adaptation logic and effects based on the selected patterns and user-centred methodology.
3. Design of the visualization model to integrate the ontology user model in the production monitoring system.
4. Test of the visualization model on the android devices by realizing the designed adaptation effects.

In the thesis work, ontology is employed to create the user model representing the contextual information. It has strong expressive power due to its freedom to define concepts and properties to relate the concepts. The ontology semantic representation of the user model not only provides the information data, but also makes it understandable by both computers and human. It can be shared, reused and extended if necessary depending on the needs of system development. In this case, along with the development of the android application, more adaptation effects could be achieved. So it is possible to extend the ontology user model to provide necessary information, which makes the design more flexible. SWRL rules are a powerful tool to enrich the ontology model in the sense of extending the reasoning capability of the user model. By designing SWRL rules, the contextual information data are related to affects the parameters of the layout and modality parameters of the device. It is the mechanism for designing the interface logic. In addition, SPARQL query language provides rich syntax to retrieve information from the user model. One important advantage of it is its support for the SWRL rules with the help of reasoners. In our study, Pellet reasoner is selected to infer the knowledge of the user model. It also supports the SWRL rules and provides fast reasoning speed.

For storing the ontology user model, in general there are two options. One is store the ontology model in SQL database, the other is in RDF dataset storage. In the study, the user model is stored in the RDF dataset storage because it offers faster speed for reason-

ing. Both of these two storage methods provide transaction mechanism to avoid concurrency issues, which allows multi readers or single writer to the dataset.

The visualization model is completed by creating the application on the server. The application is based on servlet and http protocol to communicate with the android device. Along with the changes of the contextual information such as location, illumination, the visualization model would generate adaptive parameters for the layout and modality of the android application and send it to the device through interface deployment model. This method shows fast response and reliability. Several adaptation effects are achieved based on the visualization model. In summary, the study in this work shows that the ontology is a practical candidate for creating personalized context-aware user model while providing rich semantic meanings and flexibility.

Nevertheless, there are also limitations in the study. First, at the moment the model for capturing context information is not ready yet. So the context information was collected by using a web interface manually. Once the model is ready, it can use the same http request method as the web interface to update the context data. Second, the user model in the study specifies that one user only can use one device at the same time so the location of the device is also the location of the user. The number of the users is limited in the test.

In terms of future work, one improvement could be combining uncertainty in the user model. In reality, not all activities can be identified or given a certain status. In the user model of the study, it is asserted that the user is performing a certain task. However, sometimes we are not certain if the user is performing this task. In this case, using techniques like Bayesian network or Fuzzy logic could help us deal with this uncertainty. Hence the combination of the ontology model and uncertainty could further improve the user model.

Additionally, this study is focused on android tablets. In the future, it is possible to extend the user model to support other types of mobile devices (mobile phone, laptop) or other operating system like ios.

Due to the presented facts, the designed visualization model presents a good solution for the production monitoring system and flexibility for the system upgrade in the future.

REFERENCE

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94-104, 1991.

[2] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *Personal Communications, IEEE,* vol. 8, no. 4, pp. 10-17, 2001.

[3] L. C. C. D. N. M. P. D. W. B. S. Kerry-Louise Skillen, "Ontological User Profile Modeling for Context-Aware Application Personalization," in *Lecture Notes in Computer Science*, vol. 7656, Springer, 2012, pp. 261-268.

[4] A. Nieto and J. Martínez, "Enhancement of industrial monitoring systems by utilizing context awareness," in *IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, 2013.

[5] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," New York, 1996.

[6] J. Eisenstein, "Adapting to mobile contexts with user-inerface modeling," in *Mobile Computing Systems and Applications*, Alamitos, CA, 2000.

[7] "ASTUTE," [Online]. Available: http://www.astute-project.eu. [Accessed 10 1 2013].

[8] S. K. A. SUBRAMANIAM, S. H. B. HUSIN, Y. B. YUSOP and A. H. B. HAMIDON, "Real time production performance monitoring system a production aid for all industries," in *6th WSEAS International Conference on CIRCUITS, SYSTEMS, ELECTRONICS,CONTROL & SIGNAL PROCESSING*, Cairo, Egypt, 2007.

[9] D. Stone, C. Jarrett, M. Woodroffe and S. Minocha, User Interface Design and Evaluation, San Francisco: Elsevler, 2005, p. 704.

[10] W. Hudson, "Adopting User-centered Design within an Agile Process: A Conversation," *Cutter IT Journal,* vol. 16, 2003.

[11] E. Schaffer, Institutionalization of Usability: A Step-By-Step Guide, Addison-Wesley Professional, 2004.

[12] D. P.-O. Bos, M. Poel and A. Nijholt, "A Study in User-Centered Design and Evaluation of Mental Tasks for BCI," in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 122-134.

[13] I. 1. 4, Part 210: Human-centred design for interactive systems, American National Standards Institute (ANSI), 2010.

[14] M. D. Byrne, “ACT-R/PM and menu selection: applying a cognitive architecture to HCI,” *International Journal of Human-Computer Studies*, vol. 1, no. 55, pp. 41-84, 2001.

[15] A. Goker and H. Myrhaug, “AmbieSense - interactive information channels in the surroundings of the mobile user.,” *Universal Access in the Information Society*, vol. 1, no. 3, pp. 230-233, 2003.

[16] S. R. Garzon and M. Poguntke, “The Personal Adaptive In-Car HMI: Integration of External Applications for Personalized Use,” in *Advances in User Modeling* , Berlin, Springer Berlin Heidelberg, 2012, pp. 35-46.

[17] Wahlster and Kobsa, “Dialog-Based User Models,” *Journal Proceedings of the IEEE,*, vol. 74, no. 4, pp. 948-960, 1986.

[18] P. Brusilovsky and E. Millán, The adaptive web, Heidelberg, Berlin: Springer-Verlag, 2007, pp. 3-53.

[19] A. Nieto, Y. Evchina, A. Dvoryanchikova and J. Martínez, “Pro-active ContentManaging System for Efficient Human Machine Interaction in Data Intensive Environments,” in *11th IEEE International Conference on Industrial Informatics*, 2013.

[20] Wikipedia, “Wikipedia,” [Online]. Available: http://en.wikipedia.org/wiki/User_modeling. [Accessed 16 05 2013].

[21] S. Sosnovsky and D. Dicheva, “Ontological technologies for user modelling,” *Semantics and Ontologies*, vol. 5, no. 1, pp. 32-71, 2010.

[22] J. R. Anderson and C. J. Lebiere, The Atomic Components of Thought, Psychology Press, 1998.

[23] J. R. Anderson, M. Matessa and C. Lebiére, “ACT-R: A theory of higher-level cognition and its relation to visual attention,” *Hum.–Comput. Interaction*, vol. 12, p. 439462, 1997.

[24] S. X. Ju, M. J. Black and Y. Yacoob, “Cardboard people: A parameterized model of articulated image motion,” in *Proc. Int.Conf.Automatic Face and Gesture Recognition*, Nara, Japan, 1995.

[25] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler and L. Tran, “Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0,” W3C, [Online]. Available: http://www.w3.org/TR/CCPP-struct-vocab/. [Accessed 10 05 2013].

[26] E. Not, D. Petrelli, M. Sarini, O. Stock, C. Strapparava and M. Zancanaro, “Hypernavigation in the physical space: adapting presentations to the user and to the situational context,” *The New Review of Hypermedia and Multimedia*, pp. 33-46, 1998.

[27] D. Zuehlke, “SmartFactory - A Vision becomes Reality,” in *Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing*, 2009.

[28] L. Ardissono and A. Goy, “Tailoring the Interaction with Users in Electronic Shops,” in *Proceedings of the 7th International Conference on User Modeling*, Canada, 1997.

[29] G. T. R, “A translation Approach to Portable Ontology Specifications,” *Knowledge Acquisition,* vol. 5, no. 2, pp. 199-220, 1993.

[30] Musen, “Dimensions of knowledge sharing and reuse,” *Computers and Biomedical Research,* vol. 25, pp. 435-467, 1992.

[31] W. O. W. Group, “OWL 2 Web Ontology Language Document Overview (Second Edition),” 11 December 2012. [Online]. Available: http://www.w3.org/TR/owl2-overview/. [Accessed 17 April 2013].

[32] R. W. Group, “Resource Description Framework (RDF),” [Online]. Available: http://www.w3.org/RDF/. [Accessed 02 05 2013].

[33] D. Brickley and R. Guha, “RDF Vocabulary Description Language 1.0: RDF Schema,” [Online]. Available: http://www.w3.org/TR/rdf-schema/. [Accessed 02 05 2013].

[34] M. Obitko, “Semantic Web Architecture,” [Online]. Available: http://obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html#semantic-web-layers. [Accessed 03 05 2013].

[35] “SWRLLanguageFAQ,” 21 March 2012. [Online]. Available: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ. [Accessed 17 April 2013].

[36] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML,” 21 May 2004. [Online]. Available: http://www.w3.org/Submission/SWRL/. [Accessed 17 April 2013].

[37] E. Prud'hommeaux and A. Seaborne, “SPARQL Query Language for RDF,” 15 January 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query. [Accessed 18 April 2013].

[38] I. Kollia, B. Glimm and I. Horrocks, “SPARQL query answering over OWL ontologies,” in *The Semantic Web: Research and Applications*, Berlin, Springer Berlin Heidelberg, 2011, pp. 382-396.

[39] “Protégé,” Stanford Center for Biomedical Informatics Research, [Online]. Available: http://protege.stanford.edu/. [Accessed 30 05 2013].

[40] M. A. Musen, “Automated Support for Building and Extending Expert Models,” *Machine Learning,* vol. 4, pp. 347-375, 1989.

[41] M. A. Musen, Automated Generation of Model-Based Knowledge-Acquisition Tools, Morgan Kaufmann, 1989, p. 2.

[42] O. Neira, A. Nieto and R. C. J. Martinez, “A Builder for Adaptable Human Machine Interfaces for Mobile devices,” in *11th IEEE International Conference on*

Industrial Informatics, 2013.

[43] A. Cooper, About Face: The Essentials of Interaction Design, IDG Books,, 1995.

[44] K. Goodwin and A. Cooper, Designing for the Digital Age: How to Create Human-Centered Products and Services, Wiley, 2009.

[45] D. Browne, Studio Structured User-Interface Design for Interaction Optimisation, Prentice Hall, 1994.

[46] A. Ratzka, "Identifying User Interface Patterns from Pertinent Multimodal Interaction Use Cases.," in *Mensch & Computer*, Oldenbourg Verlag, 2008, pp. 347-356.

[47] A. Ratzka, "Design Patterns in the Context of Multi-modal Interaction," in *Proceedings of the Sixth Nordic Conference on Pattern Languages of Programs*, 2007.

[48] D. Saffer, Designing Gestural Interfaces, O'Reilly Media, Inc., 2008.

APPENDIX 1: SELECTED DESIGN PATTERNS IN THE STUDY

Pattern name	**Behaviour**
Context Adaptation	Interaction (input and output) is able to be adapted to the current situation, environment and user without the user having to perform additional interaction steps.
Non-disruptive Notification	The application involves user notification about events. It is necessary to help the user focus on more important task instead of interrupting him.
Alert	The application involves user notification about events. It is necessary to disrupt the user if an urgent event occurs.
Important message	The ways to convey important messages to the user.
Proximity Activates /Deactivates	This pattern is for triggering actions only by the presence of human body without other gestures.
Redundant output	Assure information output when communication channels are distorted in an unforeseeable way?
Spatial input interaction	Multimodal input interaction adapted to tasks that involve the expression of spatial information. (Selecting objects in a 2D space, modifying spatial attributes of an object, designating points in a 2D space.)
Audio visual workspace	Minimize distraction in dual-task scenarios.
Audio visual presentation	Convey complex information such as the functioning of technical systems or complex processes.
Multimodal interactive maps: Multi-modal Spatial Search	Interactive map service.
Multiple Alerts dissemination	Avoid overwhelming the user by multiple alerts conveyed to the user at the same time or in a short interval of time.
Perspective information detailing or Detail and Adaptation	The user is preforming a task with a goal or intention known by the system. Present essential information to the user.
Modality combination in enriched information	The user is faced with a complex and dense amount of information. Convey simpler and meaningful information with combination of modality without removing information.
Redundant output for short instructions	Convey a command or a short instruction in a stressful environment.
Complementary modalities for alarms	The user is confronted with an urgent alarm that requires immediate action. Convey the maximum of pertinent information in a short way, without the user taking any steps.
Enrich sound notifications	The user is confronted with a sound notification (in a non-safety critical situation). Convey in a short way, extra information using the same channel.
Spatial representation	Represent the location of a user, object or point.

Multimodal instruction.	Make use of multimodality to convey understandable instructions and allow user input.
Proactive Feed forward	Help the user to anticipate certain situations.
Information and Solution presentation (articulation)	During his journey, the user faces different events. Adapt the information transmitted so it can be effective in a determined situation.
Metaphor	A system can be used by users with different expertise with the system. Make sure the user knows what to do (understand what is displayed and how to interact and command).
Warning (as dismissible)	Inform the user about an issue that is not urgent but critical.
Mode switch	Deal with change of system mode and/or user mode.
Viewpoint / Abstract selection	Information in a system can be consulted by different users, with different perspectives and different goals.
Detail levels	Assure the information provided by the system is specific enough or general enough for each users and situation.
User state	Safely distribute user state information.
Simulation	Support future user decisions with the current available data.
Map navigation	Adequately display user location and route in a map.
Task management	Overview and execute tasks.
Deliberate Modality selection	Adapt the modalities supported by the system to the user.
Log	Document and keep track of events.

APPENDIX 2: CONTEXTMANAGER CLASS

```
package Definer;

import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ReadWrite;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.tdb.TDB;
import com.hp.hpl.jena.tdb.TDBFactory;
import com.hp.hpl.jena.tdb.base.block.FileMode;
import com.hp.hpl.jena.tdb.sys.SystemTDB;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import org.mindswap.pellet.jena.PelletReasonerFactory;

public class ContextManager {

        private OntModel m;
        private String currentUser;
        private final String base_iri = "http://fi.tut.astute/userprofile#";
        private Dataset ds;
        private String currentDevice;
        private String tdbPath="/home/fast/tdb/";

       public void setCurrentUser(String userID){
       this.currentUser=userID;
       }

       public ContextManager(String userID, String deviceID) {

           this.currentUser = userID;
           this.currentDevice = deviceID;
           TDB.getContext().set(SystemTDB.symFileMode, FileMode.direct);
           tdbPath+=userID;
           String directory = tdbPath;
           ds = TDBFactory.createDataset(directory);

           ds.begin(ReadWrite.WRITE);
           Model model = ds.getNamedModel(userID + "profile");
           OntModelSpec ontSpec = new OntModelSpec(PelletReasonerFactory.THE_SPEC);
           OntDocumentManager mgr = new OntDocumentManager();
```

```
            ontSpec.setDocumentManager(mgr);
            m = ModelFactory.createOntologyModel(ontSpec, model);
        }

        public OntModel getModel() {
            return m;
        }
        public void update() {
            try {
                if(!ds.isInTransaction()){
                ds.begin(ReadWrite.WRITE);
                }
                ds.commit();
            } finally {
                ds.end();
            }
        }
        public void setIllumination(double light) {
            try {
                Individual device = m.getIndividual(base_iri + currentDevice);
                ObjectProperty isInLocation = m.getObjectProperty(base_iri + "isInLocation");
                Individual location =
m.getIndividual(device.getPropertyResourceValue(isInLocation).getURI());
                ObjectProperty hasIllumination = m.getObjectProperty(base_iri + "hasIllumination");
                Individual dark = m.getIndividual(base_iri + "dark");
                Individual bright = m.getIndividual(base_iri + "bright");
                Individual average = m.getIndividual(base_iri + "averageLight");

                if (light < 30) {
                    location.setPropertyValue(hasIllumination, dark);

                } else if (30 <= light & light <= 70) {
                    location.setPropertyValue(hasIllumination, average);

                } else if (70 < light) {
                    location.setPropertyValue(hasIllumination, bright);

                }
            } catch (Exception ex) {
             ds.end();
            }
        }
        public void setUserRole(String role) {
            try {
                ObjectProperty hasRole = m.getObjectProperty(base_iri + "hasRole");
                Individual user = m.getIndividual(base_iri + currentUser);
                Individual userRole = m.getIndividual(base_iri + role);
                user.setPropertyValue(hasRole, userRole);
            } catch (Exception ex) {
                ds.end();
```

```
            }
        }

        public void setUserState(String state) {
            try {
                ObjectProperty hasState = m.getObjectProperty(base_iri + "hasState");
                Individual user = m.getIndividual(base_iri + currentUser);
                Individual st = m.getIndividual(base_iri + state);
                user.setPropertyValue(hasState, st);
            } catch (Exception ex) {
                ds.end();
            }
        }
        public void addEvent(String eventName, String equipmentName) {
            try {
                Individual user=m.getIndividual(base_iri+currentUser);
                ObjectProperty hasRole=m.getObjectProperty(base_iri+"hasRole");
                Individual supervisor=m.getIndividual(base_iri+"supervisor");
                Individual operator=m.getIndividual(base_iri+"operator");
                Individual maintenance=m.getIndividual(base_iri+"maintenance");
                if(user.hasProperty(hasRole, supervisor)){
                    String eventCatalog = eventName.substring(0, eventName.indexOf("_"));
                String     queryString =
                        "PREFIX : <http://fi.tut.astute/userprofile#> "
                            + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                            + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                            + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
                            + "SELECT ?type  "
                            + "WHERE { "
                            + " :" + "supervisor" + " :receivesEventType ?type.}";
                      Query  query2 = QueryFactory.create(queryString);

                       QueryExecution qe = QueryExecutionFactory.create(query2, m);
                        ResultSet results2 = qe.execSelect();

                        while (results2.hasNext()) {

                            QuerySolution querySolution = results2.nextSolution();

                            Literal ltr = querySolution.getLiteral("type");
                            if(ltr.getString().equals(eventCatalog)){
                            OntClass eventType = m.getOntClass(base_iri + eventCatalog);
                    OntClass eventType2 = m.getOntClass(base_iri + "NotAcceptedEvent");
                    Individual event = m.createIndividual(base_iri + eventName.toLowerCase(),
eventType);
                    event.addOntClass(eventType2);
                    Individual equipment = m.getIndividual(base_iri + equipmentName);
                    ObjectProperty isFromEquipment = m.getObjectProperty(base_iri + "isFromEquip-
ment");
                    event.setPropertyValue(isFromEquipment, equipment);
```

```
            qe.close();
                    }
                  }
            }else if(user.hasProperty(hasRole, operator)){
               String eventCatalog = eventName.substring(0, eventName.indexOf("_"));
            String     queryString =
                   "PREFIX : <http://fi.tut.astute/userprofile#> "
                        + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
                        + "SELECT ?type  "
                        + "WHERE { "
                        + " :" + "operator" + " :receivesEventType ?type.}";
                  Query  query2 = QueryFactory.create(queryString);

                   QueryExecution qe = QueryExecutionFactory.create(query2, m);
                    ResultSet results2 = qe.execSelect();
                    while (results2.hasNext()) {
  QuerySolution querySolution = results2.nextSolution();

                      Literal ltr = querySolution.getLiteral("type");
                      if(ltr.getString().equals(eventCatalog)){
                      OntClass eventType = m.getOntClass(base_iri + eventCatalog);
            OntClass eventType2 = m.getOntClass(base_iri + "NotAcceptedEvent");
            Individual event = m.createIndividual(base_iri + eventName.toLowerCase(),
eventType);
            event.addOntClass(eventType2);
            Individual equipment = m.getIndividual(base_iri + equipmentName);
            ObjectProperty isFromEquipment = m.getObjectProperty(base_iri + "isFromEquip-
ment");
            event.setPropertyValue(isFromEquipment, equipment);
            qe.close();
                    }
                  }
                  }else if(user.hasProperty(hasRole, maintenance)){
            String eventCatalog = eventName.substring(0, eventName.indexOf("_"));
            String     queryString =
                   "PREFIX : <http://fi.tut.astute/userprofile#> "
                        + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
                        + "SELECT ?type  "
                        + "WHERE { "
                        + " :" + "maintenance" + " :receivesEventType ?type.}";
                  Query  query2 = QueryFactory.create(queryString);
                   QueryExecution qe = QueryExecutionFactory.create(query2, m);
                    ResultSet results2 = qe.execSelect();

                    while (results2.hasNext()) {
                      QuerySolution querySolution = results2.nextSolution();
```

```
                    Literal ltr = querySolution.getLiteral("type");
                    if(ltr.getString().equals(eventCatalog)){
                    OntClass eventType = m.getOntClass(base_iri + eventCatalog);
            OntClass eventType2 = m.getOntClass(base_iri + "NotAcceptedEvent");
            Individual event = m.createIndividual(base_iri + eventName.toLowerCase(),
eventType);
            event.addOntClass(eventType2);
            Individual equipment = m.getIndividual(base_iri + equipmentName);
            ObjectProperty isFromEquipment = m.getObjectProperty(base_iri + "isFromEquip-
ment");
            event.setPropertyValue(isFromEquipment, equipment);
            qe.close();
                    }}
    }
    } catch (Exception ex) {

        }
      }

      public void postponeEvent(String eventName) {

        try {
          Individual event = m.getIndividual(base_iri + eventName);
          OntClass pendingEvent = m.getOntClass(base_iri + "PendingEvent");
          OntClass notAcceptedEvent = m.getOntClass(base_iri + "NotAcceptedEvent");
          event.removeOntClass(notAcceptedEvent);
          event.addOntClass(pendingEvent);
        } catch (Exception ex) {
    ds.end();
        }
      }
      public void addTask(String eventName) {
    try {
          String taskName = "task_" + eventName;
          String taskType = "Task" + eventName.substring(0, eventName.indexOf("_"));
          OntClass taskTypeClass = m.getOntClass(base_iri + taskType);
          Individual event = m.getIndividual(base_iri + eventName);
          Individual newTask = m.createIndividual(base_iri + taskName, taskTypeClass);
          ObjectProperty leadsToTask = m.getObjectProperty(base_iri + "leadsToTask");
          event.addProperty(leadsToTask, newTask);
          OntClass primaryTaskClass = m.getOntClass(base_iri + "PrimaryTask");
          OntClass notAcceptedEvent = m.getOntClass(base_iri + "NotAcceptedEvent");
          OntClass acceptedEvent = m.getOntClass(base_iri + "AcceptedEvent");
          if(event.hasOntClass(notAcceptedEvent)){
           event.removeOntClass(notAcceptedEvent);

          }
          event.addOntClass(acceptedEvent);

          String queryString =
```

```
                "PREFIX : <http://fi.tut.astute/userprofile#>"
                + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
                + "ASK  "
                + "WHERE { :task_generalview rdf:type :PrimaryTask.}";
            Query query = QueryFactory.create(queryString);
            QueryExecution qe = QueryExecutionFactory.create(query, m);
            Boolean results = qe.execAsk();
            if (results) {

                newTask.addOntClass(primaryTaskClass);
                Individual generalView = m.getIndividual(base_iri + "task_generalview");
                generalView.removeOntClass(primaryTaskClass);
            } else {
                OntClass secondaryTaskClass = m.getOntClass(base_iri + "SecondaryTask");
                newTask.addOntClass(secondaryTaskClass);
            }
            qe.close();
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void endEvent(String eventName) {
        try {
            Individual event = m.getIndividual(base_iri + eventName);
            event.remove();
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void endTask(String taskName) {

        try {
            Individual task = m.getIndividual(base_iri + taskName);
            task.remove();

            endEvent(taskName.substring(taskName.indexOf("_") + 1));
taskName.substring(taskName.indexOf("_") + 1));
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void resetPrimaryTask() {
        try{
        String queryString =
                "PREFIX : <http://fi.tut.astute/userprofile#> "
                + "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                + "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
```

```
                + "SELECT ?sndTask "
                + "WHERE { ?sndTask rdf:type :SecondaryTask."
                + " ?sndTask :hasPriority ?pri.}"
                + " ORDER BY  DESC(?pri)";
            Query query = QueryFactory.create(queryString);
            QueryExecution qe = QueryExecutionFactory.create(query, m);
            ResultSet results = qe.execSelect();
            if (results.hasNext()) {

               QuerySolution querySolution = results.nextSolution();
               Resource ltr = querySolution.getResource("?sndTask");
               String ltrURI = ltr.getLocalName();
               Individual task = m.getIndividual(base_iri + ltrURI);
               OntClass priTask = m.getOntClass(base_iri + "PrimaryTask");
               OntClass secTask = m.getOntClass(base_iri + "SecondaryTask");
               task.removeOntClass(secTask);
               task.addOntClass(priTask);
            } else {
               Individual task = m.getIndividual(base_iri + "task_generalview");
               OntClass priTask = m.getOntClass(base_iri + "PrimaryTask");
               task.setOntClass(priTask);
            }
              qe.close();
            }catch(Exception ex)
            {
            ds.close();
            }
         }
         public void setTask(String eventName) {
            try {
               Individual event = m.getIndividual(base_iri + eventName);
               ObjectProperty leadsToTask = m.getObjectProperty(base_iri + "leadsToTask");
               Individual taskInstance =
m.getIndividual(event.getPropertyResourceValue(leadsToTask).getURI());
               OntClass primaryTask = m.getOntClass(base_iri + "PrimaryTask");
               OntClass task = m.getOntClass(base_iri + "Task");
               ExtendedIterator instances = primaryTask.listInstances();
               Individual thisInstance = null;
               while (instances.hasNext()) {

                  thisInstance = (Individual) instances.next();

               }
               thisInstance.setOntClass(task);
               taskInstance.setOntClass(primaryTask);
               OntClass notAcceptedEvent = m.getOntClass(base_iri + "NotAcceptedEvent");
               event.removeOntClass(notAcceptedEvent);
               OntClass acceptedEvent = m.getOntClass(base_iri + "AcceptedEvent");
               event.addOntClass(acceptedEvent);
            } catch (Exception ex) {
```

```
            ds.end();
        }
    }

    public void setUserTraining(String trainingName) {
        try {
            Individual training = m.getIndividual(base_iri + trainingName);
            Individual user = m.getIndividual(base_iri + currentUser);
            ObjectProperty hasTraining = m.getObjectProperty(base_iri + "hasTraining");
            user.setPropertyValue(hasTraining, training);
        } catch (Exception ex) {
            ds.end();
        }
    }

    public void addUserTraining(String trainingName) {
        try {
            if (trainingName.equals("noTraining")) {
                this.setUserTraining(trainingName);
            } else {
                Individual training = m.getIndividual(base_iri + trainingName);
                Individual user = m.getIndividual(base_iri + currentUser);
                ObjectProperty hasTraining = m.getObjectProperty(base_iri + "hasTraining");
                user.addProperty(hasTraining, training);
                Individual noTraining = m.getIndividual(base_iri + "noTraining");
                if (user.hasProperty(hasTraining, noTraining)) {
                    user.removeProperty(hasTraining, noTraining);
                }
            }
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void setDeviceLocation(String location) {
        try {
            Individual device = m.getIndividual(base_iri + currentDevice);
            OntClass loc = m.getOntClass(base_iri + "Location");
            Individual lctInd = m.getIndividual(base_iri + location);
            if (lctInd == null) {
                lctInd = m.createIndividual(base_iri + location, loc);
            }
            ObjectProperty isInLocation = m.getObjectProperty(base_iri + "isInLocation");
            device.setPropertyValue(isInLocation, lctInd);
        } catch (Exception ex) {
            ds.end();
        }
    }

    public String addDevice(String device, String deviceType) {
        try {
```

```
                Individual user = m.getIndividual(base_iri + currentUser);
                Individual devInd = m.getIndividual(base_iri + device);
                if (devInd == null) {
                    OntClass dev = m.getOntClass(base_iri + deviceType);
                    devInd = m.createIndividual(base_iri + device, dev);
                }
                ObjectProperty hasDevice = m.getObjectProperty(base_iri + "hasDevice");
                Resource prevDevice =user.getPropertyResourceValue(hasDevice);
                String prevDeviceID=null;
                if(prevDevice!=null&&(!prevDevice.getLocalName().equals(device))){
                prevDeviceID=prevDevice.getLocalName();
                }
                user.setPropertyValue(hasDevice, devInd);
            return prevDeviceID;
            } catch (Exception ex) {
                ds.end();
                return null;
            }
        }
        public void setModeOnOff(Boolean mode) {
            try {
                Individual user = m.getIndividual(base_iri + currentUser);
                ObjectProperty prefersMode = m.getObjectProperty(base_iri + "prefersMode");
                if (mode) {
                    Individual modeOn = m.getIndividual(base_iri + "modeOn");
                    user.setPropertyValue(prefersMode, modeOn);
                } else {
                    Individual modeOff = m.getIndividual(base_iri + "modeOn");
                    user.setPropertyValue(prefersMode, modeOff);
                }
            } catch (Exception ex) {
                ds.end();
            }
        }
        public void setUserHandedness(String handedness) {
            try {
                Individual user = m.getIndividual(base_iri + currentUser);
                ObjectProperty prefersHand = m.getObjectProperty(base_iri + "prefersHand");
                Individual hand = m.getIndividual(base_iri + handedness);
                user.setPropertyValue(prefersHand, hand);
            } catch (Exception ex) {
                ds.end();
            }
        }
        public void setNoiseLevel(double noiseValue) {
            try {
                ObjectProperty isInLocation = m.getObjectProperty(base_iri + "isInLocation");
                Individual location = m.getIndividual(m.getIndividual(base_iri + cur-
rentDevice).getPropertyResourceValue(isInLocation).getURI());
                ObjectProperty hasNoiseLevel = m.getObjectProperty(base_iri + "hasNoiseLevel");
```

```
            Individual noisy = m.getIndividual(base_iri + "noisy");
            Individual quiet = m.getIndividual(base_iri + "quiet");
            Individual normal = m.getIndividual(base_iri + "normalNoise");
            if (noiseValue < 30) {
                location.setPropertyValue(hasNoiseLevel, quiet);

            } else if (30 <= noiseValue & noiseValue <= 70) {
                location.setPropertyValue(hasNoiseLevel, normal);

            } else if (70 < noiseValue) {
                location.setPropertyValue(hasNoiseLevel, noisy);

            }
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void setUserFont(String font) {
        try {
            Individual user = m.getIndividual(base_iri + currentUser);
            ObjectProperty prefersFont = m.getObjectProperty(base_iri + "prefersFont");
            Individual fontInd = m.getIndividual(base_iri + font);
            user.setPropertyValue(prefersFont, fontInd);
        } catch (Exception ex) {
            ds.end();
        }
    }
    public void setUserExperience(String experience) {
        try {
            Individual user = m.getIndividual(base_iri + currentUser);
            ObjectProperty hasExperience = m.getObjectProperty(base_iri + "hasExperience");
            Individual expInd = m.getIndividual(base_iri + experience);
            user.setPropertyValue(hasExperience, expInd);
        } catch (Exception ex) {
            ds.end();
        }
    }
}
```