Center for Pervasive Computing

Report Series

Publication

CfPC-2001-PB-1

Author(s): Henrik Bærbak Christensen, Jakob Bardram, Søren Dittmer
Version: 1.3
Status: Closed

1   Content and Version Control

1.1   Table of Content

1.2 Version Control

Version	Date	Author	Summary of Changes
1.0	6-11-2001	HBC	Initial version
1.1	15-11-2001	JB	Version control added, details on EPJ added
1.2	29-11-2001	JB	Version created after input from AKO, HBC, and CB. Input from SSE/SD still missing.
1.3	8-1-2002	JB	Released version made after input from HBC and SD. Final input from AKO still missing.

2 Introduction

2.1 Purpose

This report describes results from theme one in the Pervasive Computing in Health Care project. The project is a collaboration between Århus Amtssygehus (AAS), Systematic Software Engineering A/S (SSE), and Center for Pervasive Computing, University of Aarhus (CfPC). The subject of theme one is *medication* i.e. to provide mobile and pervasive computing support for the activities of pouring, giving, and documenting medicine given to patients by nurses.

The project builds and evaluates a number of prototypes. Prototypes are build along two distinct lines: *mobile* prototypes and *pervasive* prototypes. SSE is the major stakeholder on the mobile prototypes while CfPC is focusing on the pervasive ones. The present report describes both strands of prototype work.

2.2 Audience

This technical report is primarily meant as report and evaluation of the workshops, prototype and experiences gained in theme one.

2.3 Overview of theme one

Theme one has been driven by three workshops with participation from AAS, CfPC, and for the last two workshops also SSE:

1. **Vision workshop**: Held at Amtssygehuset on 27th June 2001. Purpose: To brain-storm feasible pervasive computing support to identified problems in the daily work of staff at department 170, AAS, especially related to the forth-coming introduction of an electronic patient record system. A summary of the workshop discussions is outlined in Technical Report 1 [TR1].

2. **Design workshop**: Held at the Center for Pervasive Computing on 15th August 2001. Based on the vision workshop, 1 + 4 scenarios for the use of mobile and pervasive technology respectively, were developed. The feasibility of the different scenarios was evaluated by role-plays by the hospital staff. The Analysis and Design consideration for these prototypes, as well as the role-playing at the workshop is documented in Technical Report 2 [TR2].

3. **Evaluation workshop**: Held at the Center for Pervasive Computing on 9th October 2001. The mobile scenario and a scenarios merging aspect of the 4 pervasive scenarios from the design workshop was considered the most promising and selected for a prototype implementation effort. This resulted in two prototypes, one developed by SSE focusing on mobility and one developed by CfPC focusing on pervasive computer technology. These prototypes, and their underlying ideas, were evaluated at the workshop.

3 The Mobile Prototype

The mobile prototype addresses the issue that an electronic health record (EHR) transforms the versatile and highly mobile paper-based medical record to an updated, consistent and available collection of data in a computer, *but* at the cost of the mobility and possibly versatility.

By augmenting standard equipment with handheld, mobile computing devices, we create the means for extending the reach of the EHR to the patient's bedside.

The handheld computing devices tested are so-called Personal Digital Assistants, PDAs. They exist in several varieties, and the differences include computational capabilities, storage amount, connection capabilities, and operating systems.

Our choice of PDA was a Compaq iPAQ 3660 with 64 MB RAM. It contains a 206 MHz StrongARM RISC processor, and thus the same computing power as an average desktop pc just a few years ago. This means that it is entirely possible to develop and execute programs with a considerable amount of operations involved. The iPAQ offers several possibilities for expansion through an expansion jacket system. This system provides "jackets" with various expansion possibilities, such as Flash ROM, Microdrives (physically very small hard disk drives with storage capacity currently up to 5GB), Wireless LAN, Bluetooth short-range communication, GSM and/or GPRS long-range data communication and telephony, etc.

3.1 Prototype Architecture

The prototype is designed as a thin client to the EHR system developed for Aarhus County. None of the clients store data within, hence, they are all on-line with the EHR system. A schematic overview of the EHR system architecture is shown in Figure x.
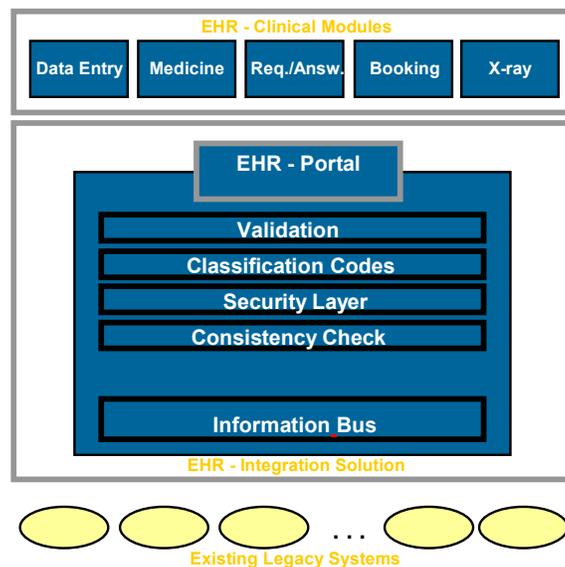
Figure x: Schematic overview of the EHR system for Aarhus County.

The Integration Solution integrates the existing legacy systems using the Information Bus. This means that there is access to existing information through EHR. Built-in data validation and consistency checks keeps data "in shape", and the security layer enforces access control to data. The classification codes are public codes for various events, such that information can be understood across administrative boundaries.

Residing on top of the Integration Solution are the Clinical Modules. These five modules handle events within their areas, but they are transparent to the end user. The end user's graphical interface to the EHR system is a flexible and modifiable view of information provided by these modules.

The prototype is designed for a very narrow functionality, that is, the prototype is applicable to a well-defined and delimited task. This means that while it does not contain the same flexibility as the standard EHR clinical

modules, it performs a subset of the tasks in one or more clinical modules. The simplicity of the module yields a reduced development time.

Further arguments for highly focused functionality include that some tasks performed at bedside consist of several similar operations on different patients, e.g. administering medicine to a group of patients.

Hence, the portable prototype consists is an add-on to the EHR system which accesses the Integration Solution. It is not at the same complexity level as the Clinical Modules, but it mimics part the functionality of one or more of them. It accesses the EHR system through the same interface as do the Clinical Modules. It does not directly rely on the functionality of any of the Clinical Modules.

3.2 Graphical User Interface and Usage

The graphical user interface on a PDA such as the Compaq iPAQ is a touch-sensitive, 320x240 pixel, 4096-color display with a stylus as input device. This means that the device is unfit for free text input; the stylus is best for selection from lists, marking checkboxes etc. The device contains a "virtual keyboard" where the user can enter keystrokes using the stylus.

The design of the interface facilitates for many functions that the user may use finger pressing instead of first extracting the stylus and then tapping the screen.

Figure 1: Login screen

The login screen, Figure 1, identifies the user, the user's organizational unit, and the user's role. Passing the login procedure, the user selects the functionality desired, and the group of patients involved.

The fixed-time medicine administration lists the group's patients for whom medicine has been prescribed for administration at this time. To each patient is attached a list for medicine prescribed, and to each of the drugs is a screen with detailed information, e.g. the physicians who prescribed and approved the medication, instructions of how to take the medication etc.

The drugs are listed with name, strength and the dose prescribed, and further there are two columns for registering when the medicine has been given and if there are any discrepancies between prescribed and actually given medication. Such discrepancies could consist of change in dose or in the time of the actual medication.

Figure 2: Fixed-medication table for a patient

Some patients may have an associated *PN prescription*, that is, a prescription of medicine which can be administered within some limits. The administration of this is handled from a different medication table, which is shown in Figure 3.

Figure 3: PN medication table

The prescribed drugs are listed in the table, and the additional information associate with each drug is a "sliding, 24-hour window" of medication. The information presented in the screen dump (Figure 3) shows that within the last 24 hours, the patient has been given 10 mg of morphine, the latest administration was at 02:00, and the dose administered at that time was 5 mg.

Most of the functionality covered by the PDA has been achieved without the use of free text input. Mostly, the PDA is used for presenting information and providing real-time, on-site documentation that the prescribed medication has been administered. Only in few situations will there be a need for free-text input, and even in case of discrepancies can most cases be handled as selections from a list rather than textual input.

3.3 Limitations and Future Work

The developed prototype is not currently on-line with the EHR system. While the prototype does register the inputs made during operation, the changes are not saved on the server. Nor are the changes saved persistently on the PDA, which means that if the program is terminated, the changes and recordings will disappear.

This, of course, is not acceptable for anything but prototypes of user interface and technology adjustment. In the future work, the PDA is connected to the EHR system via Wireless Local Area Network (WLAN). This network standard provides data rates up to 11Bbit/s, and this enables the PDA to transmit and receive all needed information for the purpose of medicine administration.

Other limitations include the input facilities. Due to security and documentation requirements, there is a mandatory login procedure. This procedure must be executed whenever the user changes. It may also have to be executed after a certain time of inactivity, and it must be executed if the PDA has been off-line with the EHR server. The login procedure can be tedious if it requires typing in a long password, and if it must be done frequently during the day, it will become a source of annoyance. Therefore, a viable solution may be to introduce employee numbers (instead of user names) and PIN codes (as a substitute for passwords).

The screen size of a PDA also requires that information is filtered. This is a task that involves personal preferences, and it can therefore give rise to some discussion. Some preferences do, however, crystallize from the discussions and evaluations, and these inputs will be taken into account for the future work.

3.4 Evaluation

The thin client prototype was the subject of evaluation at the evaluation workshop held at CfPC on October 9th 2001. The prototype was generally accepted as a useful and valuable tool for medicine administration. Among the characteristics immediately appealing to the evaluators were:

- Availability of information. This goes for information on the actual medicine, but also availability of information on the patient (e.g. on PN prescriptions when addressed by a "new" patient).

- Real-time information. Medicine tables are always up-to-date. Many time-consuming checks are rendered obsolete.

- Security. Events are recorded when they occur. Data are consistent across the EHR system.

Some critical remarks were issued too, such as:

- Tedious login procedure. There is a risk that you will act on behalf on someone else ("borrowing their PDA").

- Scalability. How does the PDA perform when the lists of e.g. drugs become huge? Is it possible to ease the way through use of codes?

- Different views. Should be possible to access single patients, not only to perform a predefined task on that patient. Various filtering may also be desirable.

- Information filtering. Some information must be at the top level because it is used every time the medicine is administered.

The final comment must be that the PDA is a sensible and valuable asset in the daily medicine administration where EHR is used.

4 The Pervasive Prototype

4.1 Background

The problem that our choice of vision addresses is that even by the introduction of a mobile EPJ solution that staff are faced with some serious issues in their daily work:

- Due to security issues the EPJ system naturally have to grant access using a traditional login procedure. A login procedure is tedious if it has to be made frequently. The problem is strengthened by the fact that it requires that your hands are free for handling the keyboard – and PDA's do not even have a decent keyboard.

- The EPJ system maintains a vast amount of information. Thus it seems plausible that a rather deep hierarchical structure will be imposed upon it. Thus the staff is faced with a significant navigational task during their daily work, as the they have to select department, unit, patient, type of information needed, date, and so on.

- Activities that are perceived as a unit in minds of the staff may require complex and tedious actions in an EPJ implementation. As an example documenting medicine given to a patient is considered "one activity" but may require the staff to select and mark each type of medicine as given in the EPJ.

- While EPJ supports up-to-date information accessible to multiple users simultaneously, it only partially leads to a higher quality in the administration of medicine. It does e.g. not secure that the right medicine-tray is given to the right patient.

The vision that was pursued was one where knowledge of the location of entities in the environment, context information, and heuristics and knowledge about common occurring work tasks are used actively to reduce staff effort on administrative, navigation, and composite tasks.

The basic premises under which the prototype was designed was

1. **Respect for staff integrity and training**. The prototype system should never modify data without consulting the involved staff. Furthermore the prototype should have a *proposing* attitude and not a *controlling* attitude. There is much difference in perspective and respect in *proposing to perform an action and then execute it if selected* in contrast to *performing the action first and then require the staff to consider canceling it.*

2. **Non-intrusive user interfaces**. The daily life of hospital staff is hectic. Our proposed activities should be discrete on the screen and should not require any interaction what so ever in case the staff is not interested the proposed activity.

3. **Guessed activities may be wrong**. There is uncertainty in our guesses of activities. Therefore the must be formulated as proposals.

4. **Privacy.** Monitoring the location of people (patients and staff) can be turned into a surveillance and monitoring system if not designed with privacy in mind. We therefore constantly in the design process tries to reveals and avoid privacy problems in our system design and architecture. For example no location data is logged – the system cannot answer where people was 5 minutes ago – it simply does not save this kind of data. Furthermore, users indicate themselves whether they want to be monitored or not.

The scenario chosen for further investigation from the design workshop was the one where entities of the hospital domain (specifically staff, patients, medicine trays, beds, and computers) were tagged using RFID-tags (Radio Frequency Identity) and the relevant locations equipped with RFID-scanners. Thereby a computer system may monitor location and movement of entities. Based upon the location of certain entities, context information, and some heuristics the prototype system is able to *guess* at activities occurring in the hospital domain. Activities are *proposed* to the staff, not forced upon them. This is done through the means of an activity bar, much like the taskbar known from the Windows operating system. Buttons representing guessed activities appear and disappear on the activity bar. Staff may acknowledge an activity by clicking/pressing its associated button. The result is usually that the activity stimulates the electronic patient record system (EPJ)

The goal of the prototype was:

- To lower the administrative load of using an EPJ system, specifically log in much faster than by using keyboard. This is even more pronounced on, say, PDA's that have no keyboard but require the user to tap characters using the stylus.

- To lower the navigational load when in the EPJ system, specifically setting up the user interface components of the EPJ with the proper view on medicine schema for the right patient and right time.

- To offer a series of complex tasks to perform in the EPJ system as a single action, specifically by allowing the nurse to document all medicine given to a certain patient by pressing one button instead of documenting it one type of medicine at the time using the EPJ medicine schema (this vision was not supported by the prototype presented at the evaluation workshop).

4.2 Prototype Architecture

The architecture was designed as a **layered architecture** [Bass98] both in the logical/conceptual architectural view as well as in the dynamic/execution view.

4.2.1 Logical View

The logical (run-time) components of the architecture are outlined below. Directed arrows show information flow.
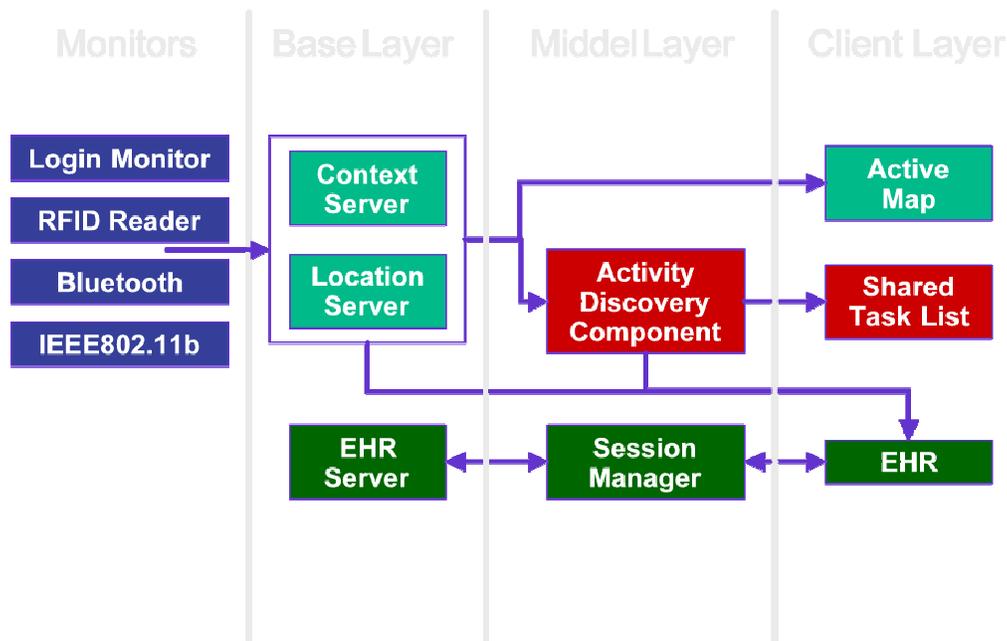
The responsibilities of the components are as follows:

- **Location server**: This component is updated with events from reality through sensors like tag-scanners, WLAN monitors, and other sources. It is the responsibility of the location server to map the hardware specific attributes to location related attributes. For instance at the hardware/reality level a scanner with ID X may trigger a TagEnter event when tag with ID Y enters its area of perception. The location server knows the exact location of scanners and keeps a database of relations between entities (type and identity) and the tag they are carrying. Thus the former TagEnter event may be interpreted as Nurse Helle entering the area of the Medicine Cupboard computer. The relation database of the location server can be updated with information from the other layers. The location server maintains a database that pairs (entity, location). Other components may query this database to obtain location information.

- **Context Server**: The context server knows about entities like persons, places and things. It associates different entities with different hardware in the location server, and then uses location

information provided by the location server to calculate relationships between the entities. Thus the former TagEnter event, from the location server, may be interpreted as Nurse "Helle" entering the area of the Medicine Cupboard computer. Other components may query this database to obtain context information.

- **Activity Discovery Component (ADC)**: This component snoops on the database of context and location information and tries to infer the activities that take place in the environment. When an activity is guessed at it generates an Activity object (a Command pattern) and forwards it to the activity bar. The central nerve system of the ADC is an expert system that uses an inference engine to detect activities based on declarative rules that are match and modify the database and trigger activity object creation. One examples of an activity is for instance that if a medicine tray for patient Hr Hansen is located at the scanner at the bed where Hr Hansen is, and nurse Helle is there also then the ADC guesses at an activity "Show medicine tray contents for Hr Hansen logged in into EPJ as Helle for the current day and time". (It should be noted that some activities are forwarded to EPJ without going through the activity bar – for instance log out activities).

- **Activity Bar**: Most activities discovered by the ADC are presented as buttons on this "taskbar". An activity embodies actions to manipulate EPJ (or another application) but is just stored in the activity bar. When a person clicks the button associated with an activity it is forwarded to the relevant application, typically EPJ.

- **Electronic Patient Record Simulator (EPJ)**: The responsibility of the EPJ is to simulate interesting parts of EPJ as seen from the CfPC project standpoint. This is mainly the graphical user interface. A second responsibility of the EPJ is to accept activity objects and execute them when they are forwarded from the activity bar.

4.2.2   Dynamic View

The logical architectural view also defines the main flow of events. Activities in reality move entities around in the physical environment that are detected by hardware sensors: tag-scanners, WLAN base stations, etc. This causes events to be propagated to the location server that interpret and add semantic information to the events. These are then propagated to the context server that further add semantic information, and so forth all the way to the Activity bar and EPJ. It was this flow of events while adding semantic information along the way that gave the original inspiration for using an expert system.

Information also flows in the opposite direction. The EPJ informs the context server about choices made by staff in the EPJ domain and augments the information held by the context server. A simple example is that EPJ inform a tray entity about which patient it is associated to. Without this information from the EPJ domain the ADC would be severely hindered in detecting activities.

4.2.3   Implementation

The prototype was implemented using J2SE v. 1.3. It uses the Swing framework for User Interfaces, and it uses plain TCP/IP sockets or Java RMI to communicate with the monitors.

The prototype currently runs on (and is tested on) Windows 2000 and Sun Solaris.

4.3 Location Server

The location server keeps track of low-level devises like RFID Tags, WLAN and Bluetooth devices, stationary computers and relevant programs running on the devices that are capable of running programs. It stores these low-level devises and relations between them in a database.

Since the database is shared between the location server, context server and ADC, it is possible for all these components to update the information in the database. The location server receives updates through monitors related to the devises, and stores the new information in the database.

It is possible to access the location server through the database, and though Java classes. Thus it is possible to query the database to get the location of a RFID Tag, but it is also possible to ask the location server for a reference to a Java object modeling the Tag, and then call a method on the object to get the location.

The location server also provides a Listener/Event interface that other components can use, to listen to events from the low-level devices.

4.3.1 Implementation

The database is implemented using the JESS expert system knowledge base [jess]. The location server can be seen as an adapter/wrapper around this database, that provides a Java API that other components can use to communicate with the expert system and the information stored within it.

The location server also has some monitor-listeners that listen for updates from the monitors. This information is send over a plain TCP/IP socket using an agreed protocol. It not only listens for updates from monitors, but it also listens for other components interested in registering event-listeners over TCP/IP in the location server, so that they can be updated when something happens with the low-level devises.

4.4 Context Server

The context server provides context information for entities like persons, places and things. The context server uses the location server, to provide it with information about low-level devises that the context server can associate with its entities.

The context server uses the same database as the location server, and as such acts directly on the location information provided by the location server. This context information is made available to the ADC in the database, but it also provides a Java API that other components can use.

The context server also contains monitors that listen for events from monitors. Currently the only monitor we have, is a monitor that listens for updates about persons logging in at programs. This information is passed as a standard Java method call, but it can either be a local call or a remote call using RMI.

The distinction between the context server and location server is not always that clear. The above-mentioned monitor is a fine example of this. The monitor receives information about people logging into programs. Here, people is an entity that only the context server knows about, but the programs is a low-level device, that runs on hosts like WLAN devices or stationary computers, and the information about which programs are running on which hosts is "kept" in the location server.

In our current implementation, the location server can run as a stand-alone component, but then some of the information that can be stored in the database, like which person has logged into which program, might not make sense. On the other hand, it is not possible for the context server to run without the information provided by location server.

4.5 Activity Discovery Component (ADC)

The responsibility of the Activity Discovery Component (ADC) is to survey the information about people, places, and things that is available in the context server, combine this with other relevant information and heuristics about recurring situations to infer likely activities. As an example of a heuristic rule is: if a patient that has been enrolled in the EPJ system but not yet assigned to a bed is near a hospital bed while a nurse is also around, then the ADC may guess at an activity 'to assign the patient to the bed by the nurse'[1].

From a software engineering point of view, the responsibility of the ADC is to survey a large number of facts about the state of physical objects (both location and context as detected by the respective server as well as medical information stored in the EPJ system) and other parameters (like time of day). Heuristics must then be applied to the survey to infer likely activities.

This problem is complicated. Heuristics are expressed using conditional statements in the procedural/object-oriented paradigm type languages. However, the number of heuristics is potentially large; the order in which the heuristics must be applied may not be obvious; and the heuristics may need to be applied repeatedly. As an example consider the above heuristics in the case that three nurses are near the patient and the bed. Then for each nurse (all) the heuristics must be consulted so see if they apply. To complicate things even more, heuristics may depend on each other. Thus the set of heuristics cannot just be executed once, but needs to be repeatedly executed until no further rules apply.

---

[1] This example rule is however not implemented in the theme one prototype.

This problem is however also solved by *expert systems*, one of the main results from the artificial intelligence community. Here heuristics are expressed by *declarations* (called rules) in a special logic language. These rules infer over and modify a *knowledgebase*. It is outside the scope of this report to describe the details of expert systems. Interested readers may consult standard references, like [Ginberg].

It was therefore decided to try to use an expert system to implement the ADC. This decision turned out to have consequences for the implementation of the location- and context servers as they in the present prototype share the knowledgebase as database of location- and context information.

4.5.1 Design

As the rest of the prototype is written in Java, we decided on the JESS Java Expert System Shell [Jess] that integrates very nicely into Java. It is possible to store Java Beans as entities in the knowledgebase and modify them both from the Jess inference engine as well as from Java. It is also possible to call java methods from Jess rules.

As the ADC has to infer over facts that at present primarily stem from the location- and context servers we simply store location- and context facts in a single Jess knowledgebase as described in the previous sections. This has several advantages. It readily provides a simple database design for the location- and context server instead of having to design one from scratch. Sharing information between the three components is as easy it can get. Finally, the ADC can infer directly on any new information any of the two servers provides without speed- and memory penalty.

4.5.1.1 Facts

The Jess knowledgebase stores *unordered facts* that resemble record instances in Pascal or struct instances in C and C++. The format of an unordered fact is defined by a *template* which is akin the struct declaration in C.

For instance if the context server detects that a person moves then an PersonMove fact is entered into the knowledgebase. The template for a PersonMove fact looks like this:

```
(deftemplate PersonMove
  (slot personId)
  (slot location)
)
```

i.e. it has to instance variables (slots) indicating the person and the location the person has moved into.

There are similar facts for EquipmentMove.

4.5.1.2 Rules

When staff moves to a new location we want to provide them with the ability to access EPJ quickly. Therefore the ADC detects a 'login-activity' and forwards this activity to the Activity Bar. The Activity Bar will then present a button that the staff can press to fast login into the EPJ.

The rule that will trigger a login-activity looks like this

```
(defrule login-staff
  (PersonMove
   (location ?loc)
   (personId ?staffId)
   )
  (staff
   (id ?staffId)
   )
   =>
  (bind ?receiver ?loc)
  (printout t "I want to login " ?staffId
          " at computer in location " ?loc crlf)
  (sendLoginActivity ( ?loc ?staffId ))
)
```

A Jess rule contains two parts: a *left-hand-side pattern* followed by a *right-hand-side action*. The two are separated by the => token. The above rule states that when two facts are seen, a PersonMove fact and a Staff fact, where moreover the value of the 'personId' slot of PersonMove and the value of the 'id' slot of staff are identical, then the action is performed. This rule ensures that the person moving is a staff, not a patient. Note that we do actually not check that there is any computer in the location. This is because computer locations are not modeled in the prototype but must be included in a more advanced prototype.

The action consists of an assignment (bind), a debug print statement, and finally a Jess function call. This function call is actually a Java method invocation that creates a command pattern instance and sends this object to the activity bar for potential execution.

The ADC rules of the theme one prototype are found in appendix 6.

4.5.1.3 *Non location based information*

It became quickly clear that the ADC needs more information than can simply be inferred from the location and movement of physical objects (which is automatically entered into the knowledgebase by the context server). The focus of theme one was medication and one obvious information that the ADC needs access to is the person that a given medication tray is associated with.

Initially a tray is 'unassigned' i.e. it is not assigned to any patient. Thus when an unassigned tray and a nurse is detected near the computer in the medicine cupboard the following rule fires:

```
(defrule unassigned-tray-swiped-in-medicine-cupboard
  "Handle case where unassigned tray is swiped in the medicine cupboard"
  ?tmoveFact <- (EquipmentMove
                (location ?loc & :(eq ?loc "MedicineCupboard"))
                (equipmentId ?eid))
  ?tray <- (tray
          (id ?eid)
          (patientId ?pttId & :(eq ?pttId nil))
          (OBJECT ?ref)
          )
  (PersonMove
   (location ?loc)
   (personId ?staffId)
   )
  (staff
   (id ?staffId)
   )
  =>
  (printout t "I have seen unassigned tray " ?eid " move to MC. " crlf)
  (printout t "  and staff: " ?staffId " is nearby " crlf )
  (bind ?receiverId "MedicineCupboard")
  (sendTrayAssignActivity ( ?receiverId ?eid ?staffId ))
)
```

The left-hand-side pattern states that if some equipment has moved to location "MedicineCupboard" and the equipment is a tray without patient assigned (pttId=nil) and a staff is in the same location – then we send a TrayAssignActivity to the Activity Bar.

The problem is that the EPJ system must inform the ADC of the patient that has been assigned to the tray. The ADC does not itself have any access to the central EPJ database. To grant it that would be one possible solution, however the EPJ database is vast and it seems infeasible that the ADC can infer over the enormous amount of data stored in it. Therefore, we decided to let the EPJ tell the ADC. Currently this is done in an extremely crude and redundant way–in essence by sending the ADC a rather unstructured data package (denoted SemanticInfo in the source code). The ADC interprets the package and updates the knowledgebase accordingly. As the EPJ has its own object that models a tray all sorts of consistency problems may arise between the EPJ and ADC tray objects.

4.5.2 Limitations and Future Work

The prototype has served two purposes:

- Validated the feasibility of the ideas: location- and context awareness, activity discovery, activity bar.

- Pinpointed limitations and weak points in the design.

Here is a short outline of limitations and points of discussion in the present design.

4.5.2.1 *EPJ information provided for ADC*

As mentioned in the previous section the ADC must have access to EPJ information to infer activities with greater accuracy. A simple example is that the ADC must know which patient a given tray is meant for. It is easy to come up with other cases.

The problem is how the ADC accesses this information? Direct access to EPJ is fine from a consistency point of view but unfeasible as the ADC cannot infer over anything other than its own, local, knowledgebase. Thus relevant information must be transferred to the knowledgebase and kept consistent with data in EPJ.

Presently a very crude data package exchange system is used that is infeasible.

An idea that we have worked a bit on is to keep only a single, say, Tray object that is accessed using RMI. Such a RMI enable bean can be stored in the knowledgebase. Thus changes within the tray object will be reflected in both the ADC and EPJ. This solves the consistency problem but potentially adds quite a lot of network traffic overhead.

4.5.2.2 Shared database for location sever, context server, and ADC

The three components share knowledgebase (and inference engine). This has the great advantage of very fast execution, no memory overhead, and no semantic gap between the types of data shared between the components. A downside consequence is high coupling between the components. This is however not quite as bad as it seems as the CLIPS code (the logic language of Jess) is modularized for the three components and for instance the location server works fine without including any of the 'higher-order' components.

Another, and more severe, problem is that the knowledge base becomes a centralized service, as discussed in the section below.

4.5.2.3 Inherently centralized architecture

Presently the Jess knowledgebase is a single-point-of-failure and a centralized resource in the design. All information between location server, context server and ADC passes through it.

This has several consequences:

- Single-point-of-failure: If the Jess engine breaks down, the system stops. Period.

- Distribution: A knowledgebase is a run-time entity running in a single Java virtual machine. In contrast in a mobile/pervasive context activities can and must be detected in many different places on many different devices.

- Scalability: It is unfeasible to have *one* huge knowledgebase that infers over all activities and data within a large hospital. The computational demands are simply too high. Thus in a realistic case many, more localized, knowledge bases must coexist. It is not trivial how to ensure consistency and scalability of a system with distributed knowledge bases.

One of the central points of discussion that the theme one prototype has raised is whether the problems of a centralized knowledgebase actually overshadows the benefits it provides.

4.6 Activity Bar

4.6.1 Software Architecture and Design

The Activity Bar registers itself with the ADC as interested in Activities concerning it. It identifies itself using a unique string. When the ADC discovers activities for a specific Activity Bar it calls the Activity Bar's `activityDiscovered` method.

If the Activity is to be shown on the Activity Bar (the method `showButton()` returns true) it adds a button to the Bar and saves the Activity for later handling. When the button is pressed, the forwards the Activity to the application (e.g. EPJ) in a separate thread. The Activity Bar calls the `activityDiscovered` method of EPJ. This again causes the EPJ to ask the Activity to `execute` with itself as argument (target).

If the Activity is not to be shown on the Activity Bar (the method `showButton()` returns false) the Activity Bar handles it right away, as described above. Examples of activities that is handled without showing a button is a LogoutActivity and a TrayLeaveActivity.

4.6.2   Graphical User Interface and Usage

The Activity Bar is shown below. It contains two activities – a `LoginActivity` and a `TrayAssignActivity`.

Normally one would think that you should press login before you can use the other button. But the `TrayAssignActivity` has "embedded" the `LoginActivity` in a was so that when pressing the second button the login is done first and then the Tray Assign is done.

4.6.3   Limitations and Future Work

The main limitation of the current version is the lack of distribution. The ADC must run on a separate machine (a "server") from the Activity Bars running in a distributed manner (on "clients").

Another limitation is the implementation of Activities using the command pattern. When using the command pattern, the Activity "remote control" the client application. This is not feasible in a distributed environment, because this would cause extensive communication over the network (as remote method invocations) and would place enormous stress on the server running the ADC. Furthermore, the ADC then would need to be changed into a highly parallel architecture, which conflicts with the one thread model used in the Jess engine.

As for the User Interface, the Activity Bar needs to be re-designed. As it is now it is very difficult to see what would happen when pressing the buttons on the bar (see the Evaluation section below).

A basic problem of the current implementation is the duplication of data object in the architecture – for example a Patient and a Tray exist a individual objects in the Context Server, in the ADC, and in the EPJ. Therefore it is not possible to assign a Patient to a Tray in the EPJ, because this would not be distributed to the other components. This again has turned out to be a major source of problems in the current version of the system.

4.7   Electronic Patient Record (EPJ) Simulator

We decided to provide our own simplistic EPJ system instead of relying on the product by SSE. This allowed a greater deal of experimentation, much simpler interfacing to the rest of the prototype system, as well as insight into the EPJ domain problems.

The current prototype focuses on medicine qua the focus of this project. Hence, the EPJ only implements functionality and views connected to medicine, like a medicine schema, ordination wizard, observations schema, etc.

4.7.1   Software Architecture and Design

The EPJ is created as a basic Swing frame that displays and control different data view components. The main classes are illustrated below.
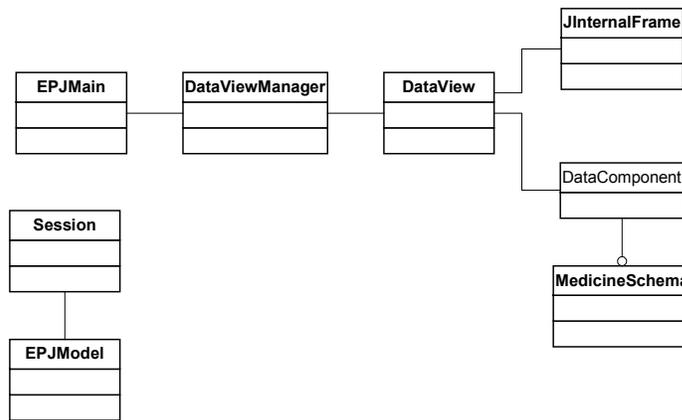
Components that want to be displayed must implement the `DataComponent` Interface, as listed below.

```java
/*
 * DataComponent.java
 *
 * Created on 12. september 2001, 20:54
 */

package dk.pervasive.phc.epj.gui;

/**
 * This Interface is used to show data components in EPJ. All data views should implement this
 interface.
 *
 * @author  Jakob Bardram
 * @version
 */
public interface DataComponent extends java.util.Observer {

    /**
     * Is called to set the current session.
     *
     * @param session
     * @see dk.pervasive.phc.epj.model.Session
     */
    public void setSession(dk.pervasive.phc.epj.model.Session session);

    /**
     * Returns the type of this data component.
     *
     * @return String - a unique string for this data component
     */
    public String getType();

    /**
     * Sets the unique type for this data component.
     */
    public void setType(String type);

    /**
     * Returns the view to be displayed.
     */
    public java.awt.Container getView();

    /**
     * Returns the menu to be added to the Applications main menu
     * If null is returned, no menu is added.
     */
    public javax.swing.JMenu getMenu();

    /**
     * Return the toolbar to be added to the Application toolbar.
     * If null is retuned no toolbar is added.
     */
```

```
    public javax.swing.JToolBar getToolBar();

    /**
     * Updates the data component's view
     */
    public void refresh();

    /**
     * Is called to notify the data component about changes. Is called when the session is changed
     */
    public void update(java.util.Observable observable, java.lang.Object obj);

    /**
     * Set the state of this data component.
     */
    public void setState(Object state);

    /**
     * Get the state of the data component as it is now.
     */
    public Object getState();
}
```

The EPJ Main application loads DataComponent classes at runtime. The properties file (epj.properties) list the DataComponent to load. An example from epj.properties is listed below:

```
#
# The Center for Pervaisve Computing EPJ Prototype properties file
#
# Jakob E. Bardram, 2001
# bardram@daimi.au.dk
#

# Path to images and other resources
resourcepath = resources/

# Path to XML file containing datamodel
modelpath = data/model.xml

#Medicine views
view.medicine.count = 3
view.medicine.menutext=Medicin

view.medicine.1.id=medicine_schema
view.medicine.1.classname=dk.pervasive.phc.epj.gui.medicineschema.MedicineSchema
view.medicine.1.name = Medicin Skema
view.medicine.1.description = Medicin skema for både Fast og PN Medicin
view.medicine.1.icon=properties.gif

view.medicine.2.id=observation_schema
view.medicine.2.classname=dk.pervasive.phc.epj.gui.medicineschema.ObservationSchema
view.medicine.2.name = Observationsskema
view.medicine.2.description = Observationsskema blodtryk, vædske, etc.
view.medicine.2.icon=PrintPreview16.gif

view.medicine.3.id=ordination_wizard
view.medicine.3.classname=dk.pervasive.phc.epj.gui.ordinationwizard.OrdinationWizard
view.medicine.3.name = Ordinationsguide
view.medicine.3.description = Guide til at lave/ændre ordinationer med
view.medicine.3.icon=OrdinationWizard.gif
view.medicine.3.palette=true
```
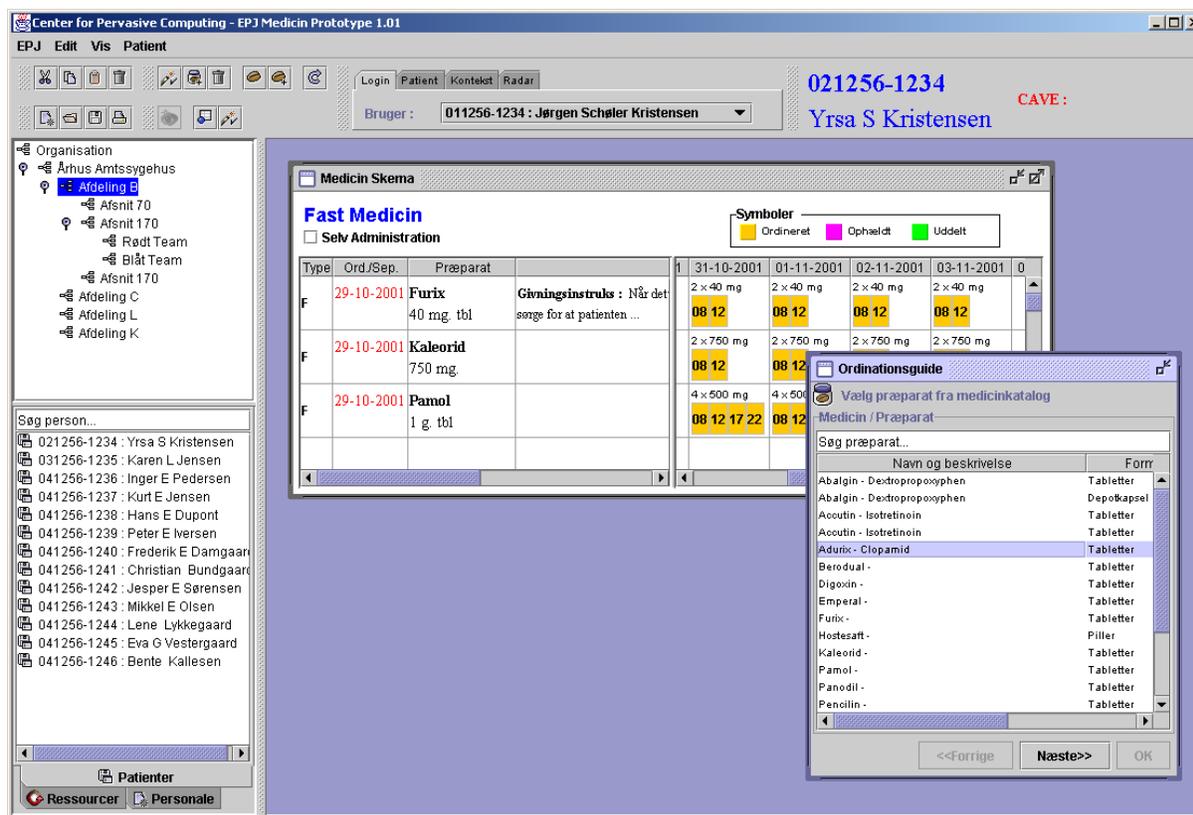
4.7.2 Graphical User Interface and Usage

As described above the EPJ provides a frame for displaying DataComponent views in the Desktop Pane of the EPJ (see below). The main frame displays an organizational overview and patient, employee, and resource overviews.

The following DataComponents exits at the time of writing:

- **MedicineSchema**. This is the main data view used in the medication theme of this project. It displays medicine ordination for a patient and keeps track on which dates this medicine should be or has been given.

- **OrdinationWizard**. This is a 3-step wizard used to prescribe (ordinate) new medicine and to extend an existing prescriptions

- **ObservationsSchema**. This view is intended to display observation data for a patient, i.e. temperature, blood pressure, and pulse. Currently this view is only implemented as a static view.

The following are the central usage principles in the prototype:

1. Only one user can be logged in at the time.

2. You can display or hide Data Views (corresponding to the DataComponents).

3. You always work on the "Active Patient". Hence you cannot see two patients at the same time, for example see the medicine schema for Patient A and B at the same time. You can open the medicine schema, and then toggle between patient A and B. There are several ways to select the active patient – using the patient overview(s) or the toolbar.

4.7.3   Limitations and Future Work

The current EPJ prototype has three large limitations in its current state:

1. **No standards are used**. We have implemented our own data model for an EPJ, including patients, staff, medicine, etc. We do not use any of the standards available like the Domain Object Model (DOM) used in the Aarhus Amt EPJ project, the EU ECHRA standard or the American HL7 standard. If this EPJ is to be more than a simulator / demonstrator we need to consider these health standards for IT.

2. **No distribution**. The EPJ runs on a single machine in a single Java VM. Hence there is no distributed setup of this EPJ. Clearly, this is to be made differently. A 3-tier architecture is to be implemented using the standard J2EE architecture pattern. This, however, can hardly be characterized as research and until we have the manpower to do this we cannot and will not focus on creating a full scale, distributed, transaction-safe, application server based EPJ architecture.

3. **No persistence**. The current persistence of the EPJ is by using an XML data exchange format. In line with the above, there is a need for creating a transaction-safe, distributed data management layer in the bottom of the EPJ. At the time of writing is has not be decided whether we want to go for a J2EE setup.

4. **No session management**. A direct consequence of the lack of distribution is the lack of distributed session management. The EPJ is intended to provide the same setup of the EPJ's "look-and-feel" no matter where or when a user logs in to the system. Hence, a nurse can use the computer at the medicine cupboard, leave it at when she uses the computer at the bedside or any other place, she just continues her session exactly as she left it. This is an essential part of what we want to demonstrate in this project, and this limitation will be looked at in theme 2.

In addition to this there are some small limitations, which might be address later:

1. No detailed properties view on data objects

2. No validation of data input, e.g. in the ordination wizard

3. No version control of data changes

4. No undo, cut, copy, paste, printing

4.8 Limitations

The current prototype comes with a number of limitations

- The full prototype runs in one JVM meaning we cannot show real distributed / pervasive behavior, only simulated it.

- As a consequence of the above limitation the prototype setup only runs on a lab top computer. No PDA's were (directly) involved.

4.9 Evaluation

Based on the evaluation workshop the following issues were discussed.

- **The usability of RFID.** The basic idea of using tags or chips to identify and locate people and things appeared appealing to the hospital staff. Especially, rapid login & -out and navigational help was perceived as highly positive features, as were the possibility of warning against the placing of wrong medicine trays at a patient's bedside (generated by non-congruency of patient-id of tray and bed). However, there is a great deal of practical problems associated with its use, especially during the prototype setup we used.

  o The reading range of passive RFID tags seems to be too short. The users had to physically place their id badge on the reader to ensure that it could see it. It was not possible to have the id badge in the pocket or attached on the chest. On the other hand, long-range reading would generate many bottoms on the activity bar, e.g. through people going by.

  o There was a tendency to increasing "hysteresis" in the system, because a lot of tags constantly entered and left the reader. For example, the user was often logged out just by turning around to pick up another medicine tray, or reaching for something. It seems like there is a need for some "intelligent" delay in the effect of a tag scanning, e.g. waiting a few seconds before logging people in or out.

  o The need for placing readers a lot of places seemed necessary if we only count on RFID to provide location information. Hence, there is a lot of work involved in finding out where to place readers in a hospital, and how to interconnect these to the Context Server. Hence, enabling the infrastructure necessary for this to work is an enormous task.

- **The idea of Activity Guessing.** The notion of guessing activities and providing the user with relevant information and tools for this activity was also an appealing idea for the hospital staff. However, a discussion of the type of activities to be guessed arose. Everybody could agree that providing help for "low-level" activities, like logging in, finding patients, showing the right data views, etc. was very helpful – indeed it seems like it would enhance the usability of an EPJ in a real work-setting significantly. However, it became more difficult to make proper guesses for more "high-level" activities, like medicine documentation, medicine handout, or medicine ordination. The basic problem is that it is difficult to guess such activities merely based on location of things and people. It was nevertheless decided that we should try to investigate this further in the rest of the project, because the usage of guessing the more "high-level" activities is considerably significant.

- **The idea of the Activity Bar.** The idea of having an Activity Bar was judged to be essential and a good way of providing assistance without imposing things on the users, which may or may not be relevant in the current use context. However, there are some minor problems with the Activity Bar as it is right now.

    o The text on the buttons is not sufficient to see what would happen if one presses the button

    o It is sometime confusing that you need to press "outside" the EPJ to make something in EPJ happen.

    o The Bar disappears behind other windows.

    o There is only room for 3-5 buttons – one might expect more activities to pop up.

    o A central question is what would happen if a staff person approaches a computer that is used by e.g. a patient. Then a Login Activity Button for the staff appears on the computer's Activity Bar. But that would enable the patient to log in! We need to consider what to do in this situation.

- **The UI of the EPJ.** There were several remarks concerning the user interface of the EPJ. In general we must say that because we do not use the "real" EPJ as developed concurrently with our project, involving the same hospital staff as involved in this research project, there is a difference in the look-and-feel of our EPJ prototype and the EPJ as developed by SSE. This leads to a great deal of confusion among the hospital staff. Furthermore, many of the usability issues surrounding the look and use of e.g. the medicine schema has already been discussed and settled upon during workshops between the hospital staff and SSE. Since it is not the aim of this project to develop an EPJ, we need to avoid these kinds of misunderstanding and confusion concerning EPJ GUI. And because the SSE EPJ is not available to us in source code, we need to make the EPJ prototype of this research project to look and function more like the SSE EPJ. Of concrete thing mentioned during the workshop was:

    o Clearly mark today in the medicine schema

    o Scroll to today when showing the medicine schema

    o Showing observation schema and medicine schema together in a shared view according to date

    o Do not support showing medicine schemas and other views for more than one patient at the time

    o Clearly mark and show what patient is in focus right now (the Active Patient concept)

    o It was very difficult to make the association between a patient's medicine (schema) and the medicine tray using the "eye" icon. Ideally this association ought to be made automatically.

    o Make it possible to mark all medicine for a specific date and time of day. For example to mark that all medicine at 12:00 has be handed out. As it work now, the nurse needs to click all the 12:00 rectangles.

- **Work-processes**: While the login & -on facility and navigation help was appreciated by hospital staff in the workshop, the playing through of scenarios also pointed towards a need to find new routines of work if the concept was to work efficiently.

- o Should the nurse place one or more medicine trays within the portal of the computer and the medicine cupboard? The latter generates many buttons and potentially leads to confusion: does the medicine schema shown and the tray into which medicine is poured belong to the same patient?

- o Should a medicine tray that is not yet connected to a patient, be connected before or after medicine is poured into it? Should there be a warning if a non-connected tray is removed after documentation of pouring medicine, but before a connecting to a patient is made?

- o At present two nurses are often found pouring medicine at the same time. This will not be possible with the EPR unless two monitors are provided for the medicine cupboard. If so, can the RFID-system be formed so as not to make two nurses interfere with each other's monitor and work process? This would seem to call for a very limited range of sensitivity that would seem to go against the interest in a more wide range for the logon-process.

- o How should warnings in cases of incongruence be formed if at all? A new, bright bottom on the screen or a warning-sound?

5   Reference

[Bass98]   Bass, L., Clements, P., Kazman, R. – *Software Architecture in Practice*. Addison Wesley. 1998.

[Jess]   JESS - Java Expert System Shell. Available from http://herzberg.ca.sandia.gov/jess/

[Ginberg]

[TR1]   Teknisk Rapport 1 – Dokumentation af Workshop på AAS d. 27. Juni 2001. Available from
http://wiki.daimi.au.dk:8000/phc/documentation.wiki

[TR1]   Teknisk Rapport 2 – Pervasive Computing in Health Care. Theme One: Administration and
Documentation of Medicine. Analysis and Design. Available from
http://wiki.daimi.au.dk:8000/phc/documentation.wiki

6 Appendix: ADC Rules

```
(defrule login-staff
  (PersonMove
   (location ?loc)
   (personId ?staffId)
   )
  (staff
   (id ?staffId)
;   (role ?role & :(neq ?role "Patient"))
   )
   =>
  (bind ?receiver ?loc)
  (printout t "I want to login " ?staffId
           " at computer in location " ?loc crlf)
  (sendLoginActivity ( ?loc ?staffId ))
)

(defrule logout-staff
  (PersonLeave
   (location ?loc)
   (personId ?staffId)
   )
  (staff
   (id ?staffId)
;   (role ?role & :(neq ?role "Patient"))
   )
  =>
  (bind ?receiver ?loc)
  (printout t "I want to logout " ?staffId
           " at computer in location " ?loc crlf)
  (sendLogoutActivity ( ?loc ?staffId ))
)

;; *** TRAY HANDLING
(defrule tray-leave
  (EquipmentLeave
   (location ?loc)
   (equipmentId ?eid))
   (tray (id ?eid))
  =>
  (bind ?receiver ?loc)
  (printout t "I want to inform about tray " ?eid crlf)
  (sendTrayLeaveActivity ( ?loc ?eid ))
)

;(defrule new-tray-object-on-tray
;  ?tmoveFact <- (EquipmentMove
;                (location ?loc & :(eq ?loc "MedicineCupboard"))
;                (equipmentId ?eid)
;                )
;  (equipment
;   (id ?eid)
;   (type ?t & :(eq ?t "tray"))
;   )
;  (not (Tray (id ?eid)))
;  (PersonMove
;   (location ?loc)
;   (personId ?staffId)
;   )
;  (staff
;   (id ?staffId)
;   (role ?role & :(neq ?role "Patient"))
;   )
;   =>
;   (printout t "No bean object for tray with id " ?eid crlf)
;   (definstance Tray (new dk.pervasive.phc.activity.Tray
;                        ?eid nil nil nil nil ))
;)

(defrule unassigned-tray-swiped-in-medicine-cupboard
  "Handle case where unassigned tray is swiped in the medicine cupboard"
```

```
  ?tmoveFact <- (EquipmentMove
                 (location ?loc & :(eq ?loc "MedicineCupboard"))
                 (equipmentId ?eid))
  ?tray <- (tray
             (id ?eid)
             (patientId ?pttId & :(eq ?pttId nil))
             (OBJECT ?ref)
             )
  (PersonMove
   (location ?loc)
   (personId ?staffId)
   )
  (staff
   (id ?staffId)
;    (role ?role & :(neq ?role "Patient"))
   )
  =>
  (printout t "I have seen unassigned tray " ?eid " move to MC. " crlf)
  (printout t "  and staff: " ?staffId " is nearby " crlf )
  (bind ?receiverId "MedicineCupboard")
;  (retract ?tmoveFact)
  (sendTrayAssignActivity ( ?receiverId ?eid ?staffId ))
)

(defrule assigned-tray-swiped
  "handle case where assigned tray is swiped at any location"
  ?tmoveFact <- (EquipmentMove
                 (location ?loc)
                 (equipmentId ?eid))
  ; tray with same id as event above indicate
  ?tray <- (tray
             (id ?eid)
             (patientId ?pttId & :(neq ?pttId nil))
             (OBJECT ?ref)
             )
  ; staff in same location HOW DO WE TEST THAT the person is staff???
  (PersonMove (location ?loc)
              (personId ?staffId)
   )
  (staff
   (id ?staffId)
;    (role ?role & :(neq ?role "Patient"))
   )
  =>
  (printout t "Assigned Tray swiped in location " ?loc crlf)
  ; find the proper computer at the given location!!!
  (bind ?receiver ?loc)
  (sendTrayShowActivity ( ?receiver ?staffId ?ref ))
)

(defrule document-medicine-given
  "handle case where assigned tray is swiped at a location
   and a patient and staff is present at the same time"
  ?tmoveFact <- (EquipmentMove
                 (location ?loc)
                 (equipmentId ?eid))
  ; tray with same id as event above indicate
  ?tray <- (tray
             (id ?eid)
             (patientId ?pttId & :(neq ?pttId nil))
             (OBJECT ?ref)
             )
  ; staff in same location HOW DO WE TEST THAT the person is staff???
  (PersonMove (location ?loc)
              (personId ?staffId)
   )
  (staff
   (id ?staffId)
;    (role ?role & :(neq ?role "Patient"))
   )
  ; patient in the same location, and the tray must belong to this patient.
  (PersonMove (location ?loc)
              (personId ?pttId)
   )
```

```
  (patient
   (id ?pttId)
;    (role ?role & :(eq ?role "Patient"))
   )
  =>
  (printout t "Document medicine given location " ?loc " by:" crlf)
  (printout t "  Staff: " ?staffId crlf)
  (printout t "  Patient: " ?pttId crlf)
  ; find the proper computer at the given location!!!
  (bind ?receiver ?loc)
  (sendMedDocActivity ( ?receiver ?staffId ?pttId ?ref ))
)
```