4 Specifications Based on the Selected Notations

By applying the four selected notations, we present the resulting specifications of the collaborative system for hospital physicians described in Section 2. To do so, we assigned one notation per author. None of the authors was an expert in one of the notations. Moreover we did not communicate with each other in order to avoid mutual influences. However, the specifications presented below are normalized to make them more readable and to provide the means to compare the notations.

4.1 Applying CTT

Collaborative activities may be described with CTT using multiple task trees : a collaborative task tree and an individual task tree per role. A collaborative task tree contains collaborative tasks and high-level individual tasks, desbribed by one of the multiple individual task trees. Figure 1 shows how we model our case study with CTT with a collaborative task tree and two individual task trees associated to the *PDA user* and *Public Screen user* roles.

The CTT notation considers three kinds of tasks : individual, abstract, which must be refined into a concrete task, system, mental and collaborative. A collaborative task is always considered as an abstract task that must be composed of individual tasks. Relations between tasks are defined with operators inherited from LOTOS. For example, as shown in Figure CTT1, we use the following operators : *concurrency* (|||), *concurrency with information passing* (|[]|), *enabling* (>>), *enabling with information passing* ([]>>), *deactivation* ([>), and an unary operator for iteration (*).

Hence, the task *have meeting* is decomposed into three sequential sub tasks: *start shared session*, *interact in a shared session*, and *stop shared session*. The two first sub-tasks are associated by the *enabling* operator which means that the two users are able to collaborate because the shared session is started. The *deactivation* operator associating the two last sub-tasks means that the *PDA user* may end the shared session.

The task *interaction in shared session* is decomposed into three concurrent sub tasks : *move telepointer*, *interact with shared document* and *show*. These tasks are linked with the iteration operator. In addition, the two first sub-tasks are linked with the *concurrency operator with information exchange* which means that the telepointer moving on the public screen is controlled by the PDA

Finally, the task *interact with shared document* is decomposed into three sequential sub-tasks: *share document*, *annotate document*, and *close document*. It means that the PDA user may share a document with the PS user. Then, the PS user is able to annotate it. The *enabling operator with information passing* explicits document sharing.

The operators used to describe the collaborative task tree can be used to describe any individual task tree but a collaborative task can not appear in an individual task tree. However, a link is maintained between the collaborative task tree and individual task trees using abstract tasks.

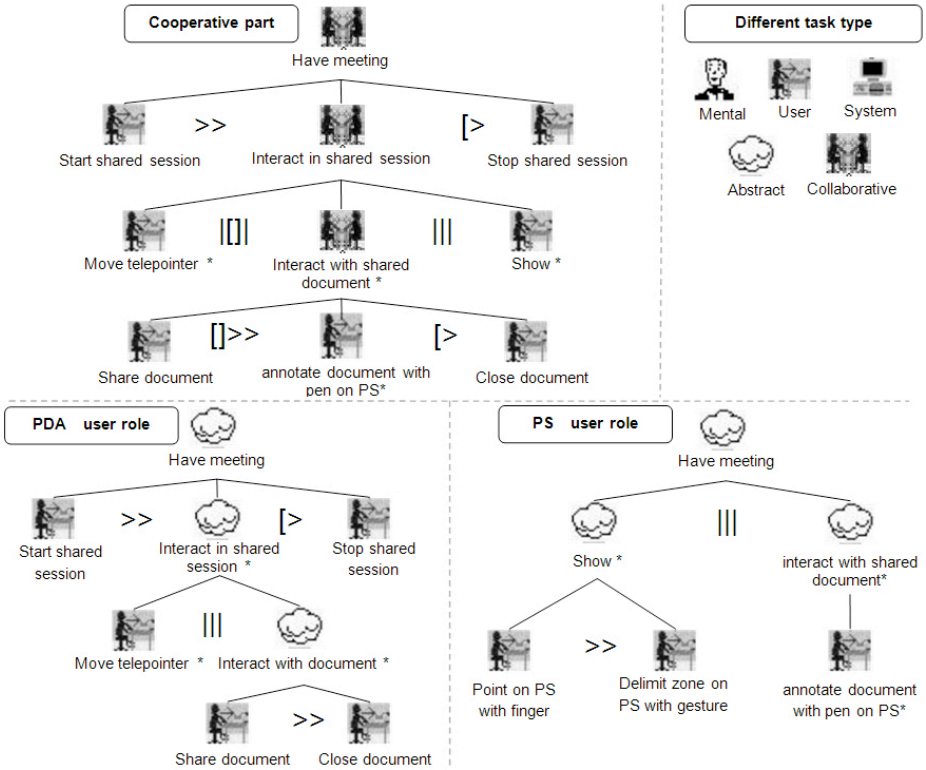


Fig. CTT1. Collaborative and two individual task tree.

The CTT notation is originally a single-user centered task tree description notation. Then, this notation has been extended to support collaborative activities. The description of collaborative activities is made at a high-level of abstraction while an individual task tree can describe concrete tasks. But, a collaborative task is no more no less than an abstract task. In addition, we observe that an individual sub-task of a collaborative task tree is not necessary associated to a role which is a bit confusing for the designer. In addition, reading or achieving a description of a collaborative task tree using the CTT notation results in a cognitive overload because we have to go to and fro repeatedly between individual and the collaborative task trees.

This notation can be used to describe concrete tasks but does not provide support to define which interaction resource or modality are available or used to accomplish a user interaction task. The only means is to use a task identifier to specify it. For example, we do so for the task *Annotate document with pen on PS*. Then, it is not possible to formally specify and describe the heterogeneity of the interaction resources.

In addition, the notation does not provide any means to represent shared objects and to specify a policy for the sharing. For example, we are able to specify that the telepointer is controlled by the PDA but we are not able to specify that the telepointer can be observed by both roles.

To conclude, the CTT notation is accurate enough to describe the interactions between a role and the system but not enough between roles. In addition, this notation suffers from a lack of complementary models to describe, for example, shared objets or interaction resources.

4.2 GTA

GTA (*Groupware Task Analysis*) is a method and a notation that can be used for the task analysis of collaborative activities, based on observations *in situ*, and that can be used as a specification tool for the design of groupware. GTA provides four models to describe collaborative activities:

- A task tree model,
- A collaborative activity workflow model,
- A class diagram of objects or artifacts,
- A model of the cultural and physical environment.

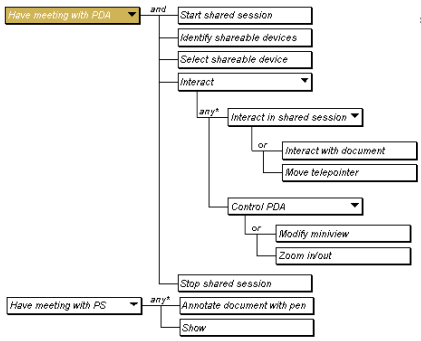
GTA is based on a simple ontology, which includes the following concepts: task, role, object, agent and goal.

In order to model our example, as shown in figure GTA1, we consider three roles : *User*, *Physician* and *System* ; the role *PDA User* is just a specialization of the role *User*. Based on these roles, we consider three agents: the Physician1 agent which is a *PDA User* and a *Physician* ; the Physician2 agent which is also a *Physician* ; the system agent which plays the *System* part. We model objects with a class diagram (not represented here) which includes few artifacts: a PDA, a stylus and shareable devices such as the public screen.

Figure GTA1 shows the task tree model. We consider one task tree per interaction surface : one for the PDA and one the public screen. The first one is the *Have Meeting using the PDA* task, associated with the *PDA User* role. This task allows a PDA user to interact with the PDA and to control a shareable device. The main sub-tasks are : (i) select a device supporting collaborative sharing ; (ii) control and interact with the public screen ; (iii) stop a shared session. The second task tree is the *Have Meeting using the PS* task, associated with the *Physician* role. This task tree contains few sub-tasks dedicated to the interaction with the public screen such as the annotation task. As shown in Figure GTA1, sub-tasks are connected to a parent task with only one operator. This lead us to introduce empty sub-tasks in the task tree in order to model a more complex task planification that corresponds to the real context of our case study.

For each task and sub-tasks, we specify what are the manipulated objects, the associated roles, the pre and post conditions and the triggered tasks. We focus on the trigerred tasks because this is a means to describe the dynamic part of the group activity. Indeed, based on this description, we are able to produce an activity diagram as shown in figure GTA2. This diagram highlights the orchestration of the activity according to each role and how the tasks are executed over time. For example, this shows the differences between the three roles and explains why we introduce the

System role. In addition, we may observe that a sharing session is always started by a *PDA User* role. It becomes a pending task during the execution of the *Identify Shareable Devices* sub-task and is resumed when the latter is over. Then, the *Have Meeting with PS* task is triggered : this and the *Interaction in shared session* task are executed simultaneously.



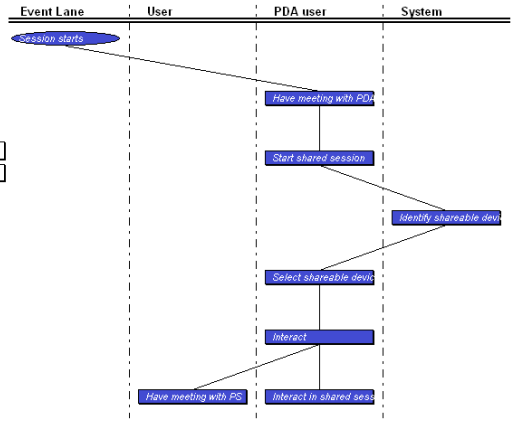


Fig. GTA1. GTA task model of our case study. **Fig. GTA2.** Activity diagram showing start of shared session.

Finally, GTA is based on NUAN to describe a concrete task. The figure GTA3 shows the modelling of the concrete task *Annotate with pen on PS*. Hence, we have to introduce the concept of pen and we have to explicit that the annotation is produced on every shared interaction surface. In particular, we have to explicit the tight coupling between both views. From a WIMP interface point of view, the NUAN description given in figure GTA3 would be interpreted as follow : a MOVETO operation means that the public screen user is moving the pen without touching the screen while the pointer is following the pen ; then the user is taping on the screen which is semantically equivalent to a click (CLICK operator) ; to annotate, the user have to hold and move the pen on the surface : a stroke is drawn. When annotating, the drawn stroke is echoed on the PDA until the user releases the pen (RELEASE operator).

User actions	Interface actions	Computation
MOVETO(<pen>,@(x,y))		
CLICK(<pen>)	SHOW(<pen(x, y)>)	Draw Point(x, y) on PDA
HOLD(<pen>)		
(MOVETO(<pen>,@(x',y'))*	SHOW(<Point(x', y')>)	Draw Point(x',y') on PDA
RELEASE(<pen>)		

Fig. GTA3. Task annotate document in NUAN.

Individual and collaborative activities are described with a typical task tree. However, rules imposed by GTA on the design of a task tree, as explained above, make it more difficult to describe. At last, we are able to do it but it requires an important cognitive effort and it produces a bigger task tree because of the dummy

tasks. However, it is convenient to describe the dynamic part of the activity with the concept of triggered task. Another difficulty is to define the best strategy in order to design the task tree because our application is working with multiple interaction surfaces. Then, we decide to design one task tree per surface in order to make explicit the multiplicity of interaction surfaces and the multiplicity of devices and modalities of interaction. We do it because GTA offers no means to model this. We think that the authors had considered that every user of a groupware application interacts with the same User Interface running on desktop computers. Maybe it is due to the fact that the notation was published back in 1996. Once again, we are able to do it but it requires an important cognitive effort.

In addition, GTA supports NUAN, which enables the design of a concrete task. Again, we have difficulties to describe the concrete tasks such as *Annotate document with pen on PS*. Indeed, the NUAN language is useful to describe interactions for WIMP interfaces but fits not very well in the case of post-WIMP interfaces, which introduce new kinds of interaction device and modality. As shown in figure GTA3, we introduce the concept of pen and we try to model the mechanic of the interaction using the given NUAN operators. We can observe that the notation consider that any input device is behaving like a mouse and that the user is manipulating a mouse pointer or cursor. The operators CLICK, HOLD and RELEASE are explicitly associated with a mouse- like device; these operators are no more usable with new interaction devices such as a tactile surface or a finger tracking. In addition, no operator is available to describe the coupling between distributed interaction surfaces. Furthermore, this notation provides no means to describe how feedback is made between devices. To do this, we have to write it explicitly as show in the third column of the table in figure GTA3.

Finally, about role distribution, we do not know how roles are managed within a task tree description : is a role, associated to a sub-task, inherited from a parent task or not ? In addition, we have to introduce the *System* role in order to explicit the detection of shareable devices as shown in figure GTA2.

To conclude, GTA does not fit well for post-WIMP interfaces, for heterogeneous technology and distributed interactive surfaces. However, we find that the concept of triggered task is a good approach to associate a static and a dynamic representation of the activities.

4.3 MABTA

MABTA is a framework based on the concepts of role, task, goal and object. A groupware application may be defined using four models : (1) a collaborative task model that describes collaboration and communication between users ; (2) a hierarchical task model, which refines the group task model, including individual tasks ; (3) a hierarchical model of roles and users involved in activities ; (4) a sketch of the user interface.

First, this notation uses labeled rectangles to represent roles and users. A line between two rectangles symbolises a relation. A vertical relation between two roles or two users means *is hierarchically over of*. A relation between a role and a user means *play role of*. As shown in figure MABTA1, we define two roles : *PDA user* and *PS*

user. The relation between these roles means that a *PDA user* is manipulating the PDA and is able to start or end a sharing session ; a *PS user* uses a public screen. We also define two users : *U1:PDA user* and *U2:PS user*. The user U1 plays the role of *PDA User* and the user U2, the role of *PS User*. In addition, U1 is *hierarchically over* U2.

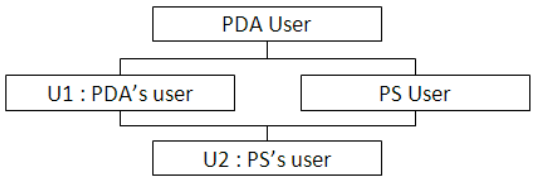


Fig. MABTA1. People relationships model our case study.

A group task model describes how collaborative tasks are organised. MABTA identifies three kinds of task: Coordination task (C), Single user task (S) and Group decision making task (D). Tasks are organised by column and each column is associated with a user. Arrows between tasks show how tasks are triggered. The relation means *influence*. As shown in figure MABTA2, a sharing sesssion activity is started by the user U1 with the *Interact with share documents* task. Then, U1 asks U2 to give his/her opinion. We can notice that this task is a cognitive and collaborative action and the associated task, *give opinion to U1*, is a group decision making task. Then, the user U2 may annotate or show a document on the public screen. Alternative tasks are simply stacked inside columns.

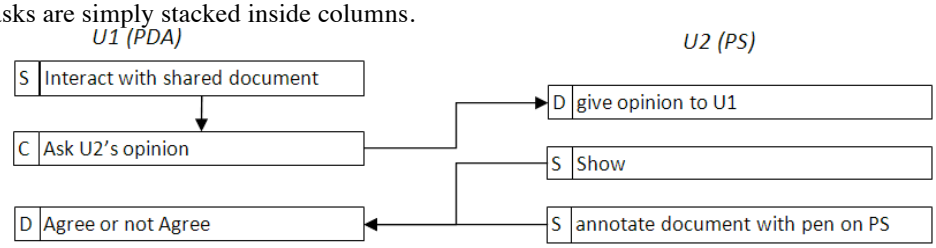


Fig. MABTA2. Group work task model.

As shown in figure MABTA3, the previous is refined into a hierachical task model that takes in account single-user tasks. Each group task is refined as a set of single-user tasks. An action plan defines the execution order of the sub-tasks. For example, the *Show* group task is decomposed into two substasks and a user may execute either because the execution plan is specified by the label *Plan: 1 or 2*.

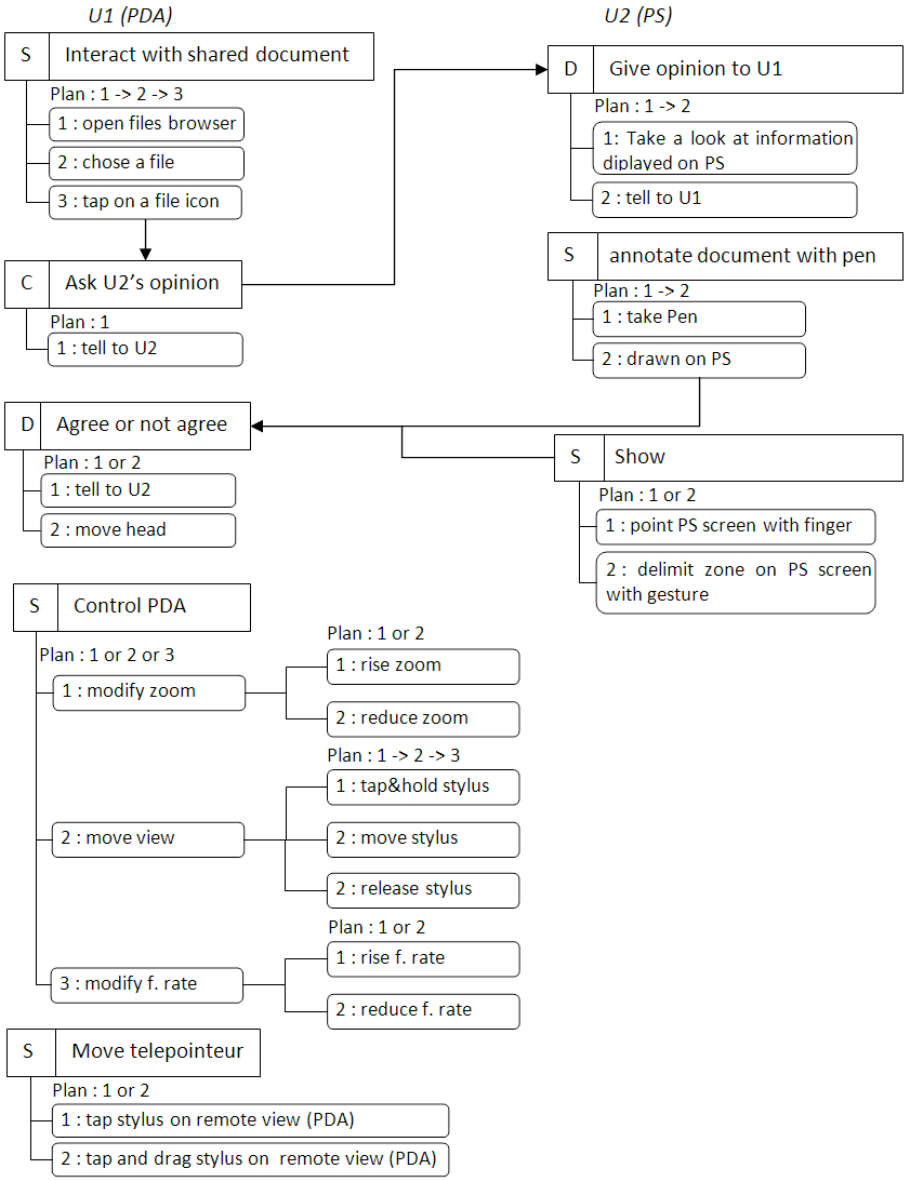


Fig. MABTA3. Hierarchical Task model.

The last model, as shown in figure MABTA4, represents a sketch of the user interface which is available on the PDA and on the Public Screen. The labels are used to identify interface elements and to link the actions done through the interface with the tasks defined by the previous model.

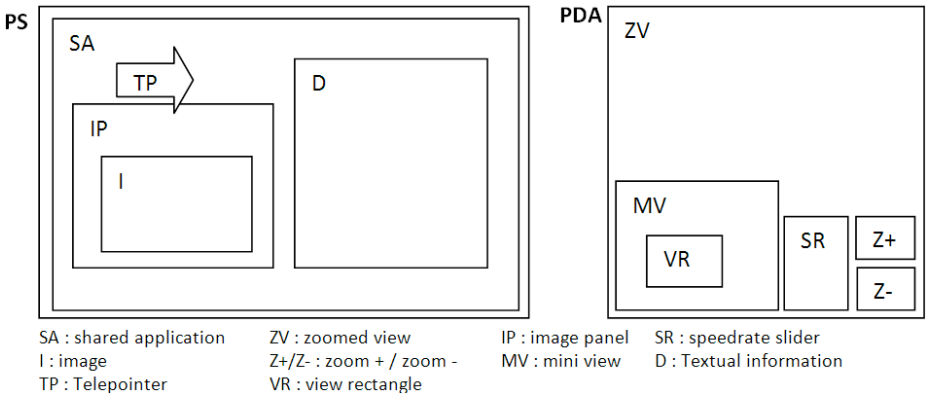


Fig. MABTA4. Interface layout model.

The user model is not a convenient model to specify relations between roles and users. Indeed, this representation does not help the designer to make the difference between inheritance and association. In addition, we have to define users statically which is supposed to be known only at runtime. We find that the frontier between roles and users is not all clear.

We find that both group and task models are very convenient to use. Indeed, it is easier to make the difference between task types and to visualise at a glance which user is associated to a task. However, the refinement into single user subtasks is very similar to the HTA notation for the representation of hierarchical task trees. Furthermore, the relation between tasks has the same meaning, *influence*, as in HTA.

A sketch model is commonly used to design user interfaces at specification time. MABTA's contribution is about the explicit link between the sketch and the task models. Indeed, each part of the sketch is labeled. Then, a label may be used in the task tree description to link interactive tasks with a part of the user interfaces. This is a means to explicit which are widgets are collaborative and which interaction resources are distributed. However, this approach does not consider multiple input or output modalities (other than visual).

To conclude, MABTA is based on four complementary models that are mainly useful for a requirement analysis. However, these models are not enough precise to fully describe and specify the behaviour of a groupware application, in particular in terms of relation between tasks (i.e. limitation of the *influence* relation) and in terms of multi-modality.

4.4 UML-G

UML is a formalism used to design a system using multiple views : the use case view describes the system from the user's point of view ; the logic view describes it from the system point of view ; the implementation view describes the dependencies between software modules ; the process view describes scheduled and the concurrent tasks ; and, finally, the deployment view describes the topography of the system. A

system can be described using UML by 13 different kinds of diagram. The most well-known are the class and sequence diagrams.

UML-G adapts UML in order to describe a groupware applicagtion using several stereotypes. The first one is the <<*shared*>> stereotype. If used along with an object or a relation, it means that they may be shared during a collaborative session. The properties of a such object or relation are :

- *lockable*, which allows a unique user to manipulate the object,
- *access-controllable*, which limits an access to a restricted set of users,
- *observable*, which means that an object can be visualized by several users,
- *time-persistant*, which means that an object can be retrieved over time,
- *distribution*, which defines how data are distributed over the network.

Stereotypes <<*sharedRole*>>, <<*sharedActor*>> and <<*sharedActivities*>> are inherited from the <<*shared*>> stereotype ; it defines concepts of role, actor or collaborative activity.

As shown in figure UML1, we consider three generic objects : *Telepointer*, *Document* and *Annocation* classes. The *Telepointer* object is a graphical and shared telepointer controlled by the PDA and visible on the public screen. The *Document* object covers the medical records shared by the physicians. The *Annotation* object represents the stroke drawn by a physician on the white board using a pen. We can observe that these objects are collaborative objects because they are tagged with the <<*shared*>> stereotype. We consider that these objects are observable and replicated on available shareable devices (PDA and public screen). Finally, we consider two roles represented by classes *PDA User* and *Public Screen User*. These classes are tagged with the <<*sharedRole*>> stereotype because these users are involved in a collaborative sharing session. Now, let us have a closer look on relations *move*, *share* or *close*, and *annotate*. These relations are also tagged with the <<*shared*>> stereotype. It means that only one *PDA User* can manipulate the telepointer (1 to 1 cardinality) and can manipulate remote electronic documents (1 to n cardinality). As shown in figure UML1, the *PS User* can also manipulate electronic documents. Finally, only the *PS User* is able to annotate documents as defined by the *annotate* relation between the *Document* and *PS User* classes. However, the class diagram shown in figure UML1 is simplified for this paper. Indeed, the final class diagram includes the classes related to the graphical elements of the user interface, such as widgets, and the classes related to the data manipulated by the system. We choose to focus mainly on shared objects used during a collaborative sharing session.

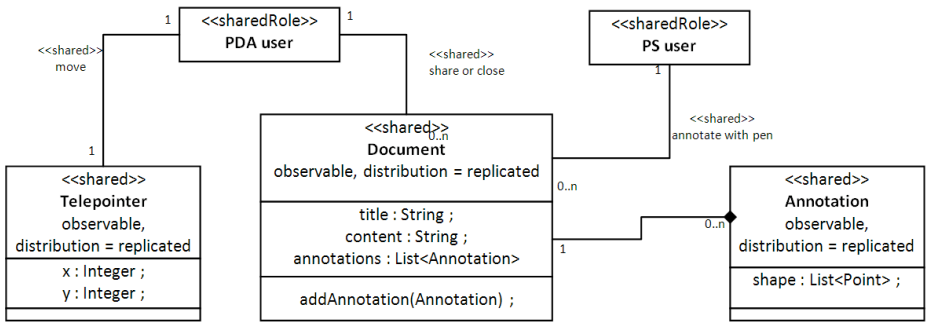


Fig. UML1. Class diagram.

A class diagram provides only a static view on the system. As shown in figure UML2, we consider an activity diagram which describes how the user interactions are planned. This diagram is organised by roles, one per column. An activity is symbolised by a rounded rectangle and a name. The planification is represented by lines connecting boxes. A sharing session is started by the *PDA User* with the *Interact in shared session* activity. Then, the *PDA User* can choose to open a file browser or to stop the sharing session. If the *PDA User* choose the first one, a medical record is open which can be manipulated concurrently between both users. The *PS User* can annotate the document while the PDA User can choose to close the file. This can be done several times again as needed.

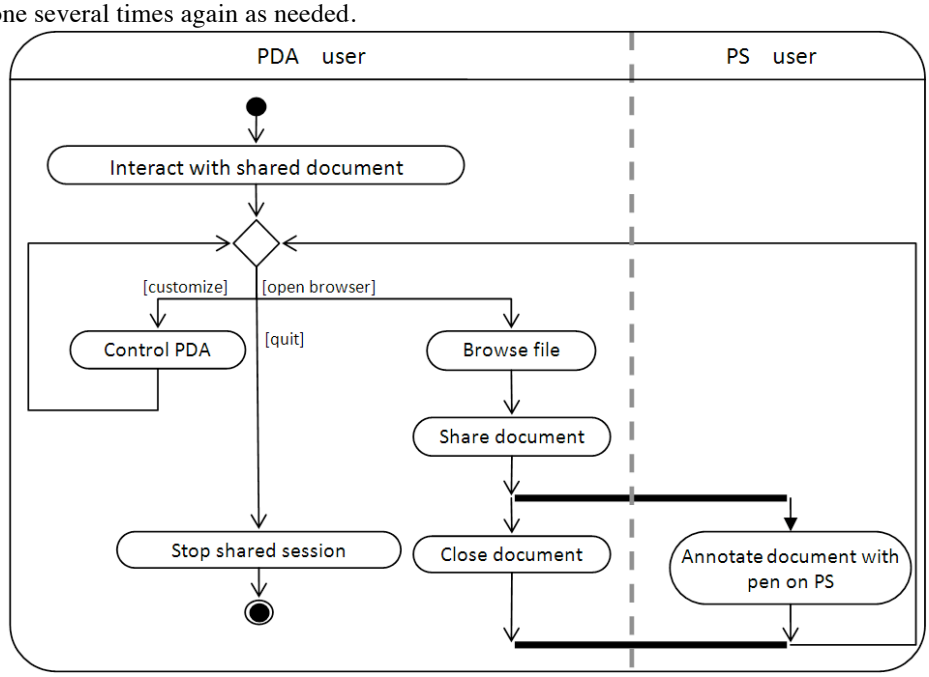


Fig. UML2. Activity diagram.

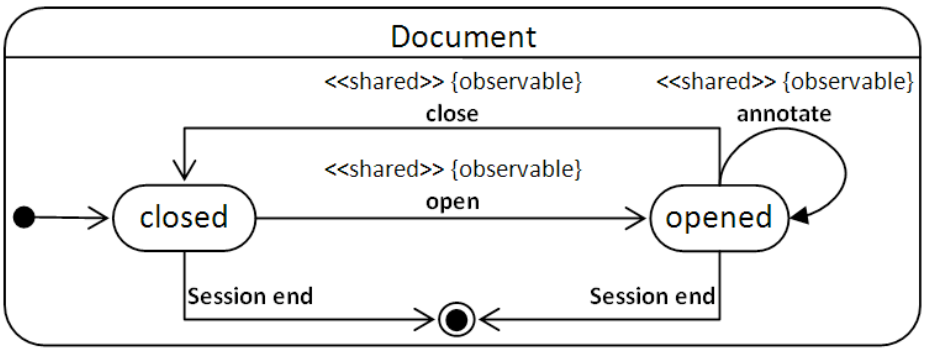


Fig. UML3. State transition diagram for document.

Both activity and transition diagrams enabled us to describe user interaction with the system and with artifacts such as the medical records. In addition, the sequence diagram is helpful to specify how a user event triggers a software component. An example is given by figure UML4 in the case of the manipulation of the telepointer. In this example, we consider two software components that compose the overall application : one is running on the PDA while the other one manages the Public Screen. Each arrows correponds to a user action or a software function call. The figure shows a mouse move-like event on the PDA using the stylus. The new coordinates are calculated by the PDA and transmitted to the application part managing the public screen. The latter updates the screen with the new location of the telepointer. The PDA also updates its screen which provides feedback to the *PDA User*.

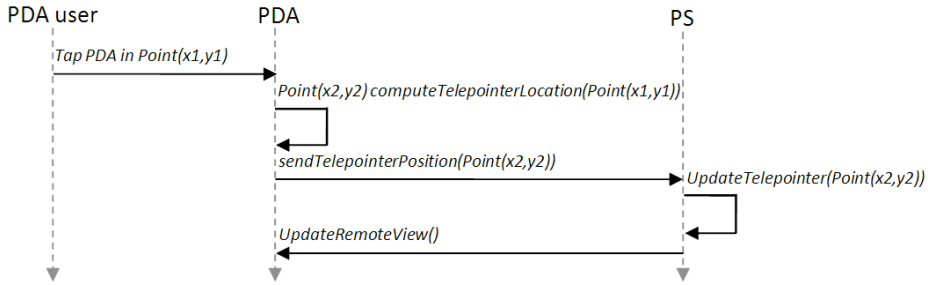


Fig. UML4. Collaborative and two individual task tree.

It would be possible to deeper describe the system but it has no interest in terms of user interaction design. Indeed, we prefer to focus on the design of the collaborative aspects instead of describing very common pieces of code.

In this part, we focus on the value bringed by UMLG compared to UML. We do not aim at discussing what UML brings or not to design groupware.

Firstly, the new stereotypes provided by UMLG are helpful to explicit the shared and observable objects used in our case study and the different roles involved in a sharing activity. In addition, we are able to explicit the observability on manipulated objects : this is made possible using the stereotypes on relations. Furthermore, the stereotypes are useful to highlight the collaborative aspects of the interactive system. We only use two properties: *{observable}* and *{distribution}*. The other properties are not needed for our case study.

However, we find that UMLG is hard to use in order to describe the interaction dialog. Indeed, relations in a class diagram are not a very convenient way, from a human and cognitive point of view, to describe the user interactions with a shared object and between users. A real diagram would become overloaded if we had to represent every relations and, then, not usable for the design.

Secondly, the state transition diagram is helpful to describe at finer grain the user interactions with the system and the state transitions that are observable by users. Maybe this is due to the *{observable}* property which can help the designer to explicit feedback and group awareness.

To conclude, UMLG adds value to UML for the design of groupware with the new stereotypes which introduce the concepts of shared object, roles, distribution or observability.