Initial draft preliminary on the fly PAC analysis for the ECOM Interface

Nigay L., Coutaz J. & Salber D.
Laboratoire de Génie Informatique, University of Grenoble

19th March 1994

Amodeus Project Document: System Modelling/IR4

Abstract

This document describes a preliminary analysis of the ECOM exemplar from the PAC perspective. It summarises the process used to address the design issues and a list of lessons drawn from the exercise. We describe our analysis for issues 1 and 2 about making AV connections. We then propose a QOC representation of this analysis Finally, we present a PAC-Amodeus architecture for the current ECOM interface along with a QOC description that justifies the software design solution.

1. Introduction

This document describes a preliminary analysis of the ECOM exemplar from the PAC perspective. The following issues are addressed:

1) How well does the information displayed by the ECOM interface conform to the underlying system states and potential actions?

2) How should the display and dialogue be designed to ensure that the user selects the correct connection and always gets the connection they expect?

3) How should the users specify their availability level?

4) What kind of control should be offered over bandwidth and when?

We consider that questions 1 and 2 cover the core design issue "Making and Breaking AV connections" as described in Document "EuroCODE AV Exemplar Round 2, RP3-ID-IR6. Questions 3 and 4, on the other hand, are identical to the original core design issues stated in RP3-ID-IR6.

This document is structured in the following way: Section 2 is a summary of the process used to address the design issues and a list of lessons drawn from the exercise. In sections 3 and 4 we describe our analysis for issues 1 and 2. Issues 3 and 4 will be addressed in a revised version of the document. Section 5 is a QOC representation of our analysis of the problem. In section 6, we propose a PAC-Amodeus architecture for the current ECOM interface along with a QOC description that justifies the software design solution.

In the following discussion, expressions in italic denote entries in the system modelling glossary SM/WP26.

2. Process and Lessons

How did we proceed to address the design issues?

1. Our first reaction was to express design issues in terms of properties and principles developed by the system modellers.

2. Secondly, concepts such as the notions of connection and accessibility level, have been analysed thoroughly leading to a better understanding of the domain and a better coverage of the services that can be provided.

3. Thirdly, based on a formal description of the key concepts, we have been able to extend or modify the current QOC analysis.

Lessons drawn from this analysis include:

1. It is very hard to stick to a PAC role. Many of our suggestions go beyond the scope of PAC software engineering. PAC is primarily applicable from precise external specifications and requires knowledge about the underlying implementation tools. The EuroCODE questions do not emphasize this perspective. As a result, some of our suggestions are proposals for new external specifications along with their implications onto the software architecture.

2. We would like to extend QOC into a QORC notation. In QOC, options are linked to criteria only. We have found it useful to link options to requirements as well. For example, in the particular case of EuroCODE, option "proactive feedback" requires (or at least suggests) that the data base for access control be active. If the data base at hand is passive, then honesty may not be achieved satisfactorily. If so, proactive feedback cannot be considered as a realistic design option.

3. The meaning of some criteria is context dependent. For example, when user-centered, flexibility expresses the scope of freedom among a set of possible choices. In software engineering, flexibility is related to product revision. It denotes the capacity of the software to accept modifications. When ambiguity occurs, we suggest to prefix the names of software engineering criteria with "SE" (stands for "Software Engineering").

4. We have derived some heuristics about classes of questions that designers should formulate when using QOC. These are based on the quintilian[1] model:
   - What ? e.g., what information is relevant to the task? (this question is sometimes addressed in theQOC examples we have seen. Should be addressed more systematically)
   - Who? e.g., who is concerned, who is in charge of the task? (this question is rarely addressed in the QOC examples we have seen. Q5 in Issue 2 (How are accessibility levels defined?) is actually an example of a "who" question. It should be rephrased accordingly
   - Where? e.g., where, on the screen, to present information (this question is frequently addressed in the QOC examples we have seen)
   - When ? e.g., when to present information (this question is rarely addressed in the QOC examples we have seen)
   - How? widely covered in most QOC examples we have seen
   - Why? rationale and causecovered by QOC criteria.

   To emphasize the "what, who, how, etc." nature of questions, we have suffixed question names with their rhetorical type.

There are three levels of sophistication in the EuroCODE project technology. These are referred to as the Low Road, Middle Road and High Road Demonstrators.

[1] From Marcus Fabius Quintilianus: Roman rhetorician of the first century A.D.

5. We suggest that the QOC notation be extended with an encapsulation mechanism just like procedures are used in programming languages. For example, the subtrees under Q5 and Q8 are similar (except for one option). If they were strictly similar, then one could replace the two subtrees by the same name and that name would denote a QOC subtree. The encapsulation facility would increase lisibility, would point out similar lines of reasoning, and would augment consistency across the user interface. They would eventually serve as reusable rationale applicable by other designers to other design problems (just like libraries are ready for use pieces of code that are reusable by different programmers in different contexts).

3. Design issues as properties

Questions 1 and 2 are concerned with *observability* and *conformance*. In turn, observability and conformance can be constrained in terms of *honesty* and *affordance*.

As defined in the system glossary (SM/WP26),

- Observability is the property that the presentation of a system contains sufficient information to allow the user to determine the functional state of the system. In the case at hand, we need to identify what information is relevant to the user for making AV connections. For doing so, we need to model the notion of connection in a precise way. If some information is missing in the presentation, then the functional state is not observable or unsufficiently observable.

- Conformance is the property that the presentation of the system mirrors the underlying behaviour of the system. If we agree that observability is reached in a reasonable way by the ECOM presentation, the next step is to check how well the presentation matches the behaviour of the functional state.

- Honesty is the property that the presentation of the system renders its functional state appropriately (e.g., does not distort the functional state). Honesty constrains conformance a little bit further: not only there should be a correspondence between the presentation and the functional state that is relevant to the task, but also features of the presentation should lead the user to interpret the functional state correctly. In particular, the presentation cannot lie. In the case of EuroCODE, it may not be always possible for the system to present the relevant system state in an honest way. In particular, changes in the system state that are relevant to multiple users cannot be updated instantaneously and simultaneously on every workstation. In such circumstances, the system should show the users that the current rendering is possibly wrong or unstable. Honesty is a necessary but not always sufficient condition for the user to be able to elaborate a correct mental representation of the functional state. Affordance goes one step further.

- Affordance is the property that the presentation of a system conveys information about the actions that can be performed by the user on that system. This property relates to the "real world" understandings that the user has. In EuroCODE, the door metaphor for expressing levels of availability favors affordance.

Properties such as observability, conformance, honesty and affordance can be used in two ways: as evaluation criteria of the current ECOM design or as new criteria in a QOC description.

4. Analysis of the key concepts

The key concepts we have considered include the notions of connection, level of availability, system state relevant for the caller, and system state relevant for the target user.

4.1. Connection

A connection is a *task domain concept* (i.e., a concept identified by task analysis as relevant to the user to accomplish the tasks in that domain). It can be characterized by a set of attributes and operations. Operations are used to modify the attributes.

For the purpose of EuroCODE, we have identified 7 attributes that are relevant to the user. (In the following description, operations are not described. They should result from a task analysis.):

- **target**: denotes the end point of the connection. Its domain is: <myself, another user, another site, another window>.

- **source**: denotes the source point of the connection. Its domain is: <myself, another user>.

- **orientation**: defines whether the connection is one way or two-way. The domain is: <one-way-in, one-way-out, two-way>.

- **duration**: provides an indication of the duration of the connection. It may be <finite, infinite or instantaneous>. When finite, the connection is closed after a certain amount of time determined by a system parameter. The value of this parameter may or may not be modified by the user through preferences. At the opposite, a connection whose duration is infinite will be broken on user's will. The system will close an infinite connection on the occurrence of exceptions (resource availability, breakdowns, time out, etc.). An instantaneous connection is a special case of finite connection whose duration is short and set up by the system.

- **occurrence**: provides an indication of when a specified connection will occur. It may be <now, as soon as possible, deferred>. A "now" connection means that the system should try to connect the two end points right on the spot without any delay. If the connection does not succeed, then there is no retry. When "asap", the specified connection will be attempted as soon as the preconditions for the connection are satisfied. For example, the camera on the target site may not be turned on. As a result, a video connection cannot occur. If specified as "asap", the connection will automatically be attempted by the system when the camera becomes available. When "deferred", the connection will be attempted later at some specified point in time. The scenario provides a good illustration of the utility of deferred connections: the supervisor leaves a message to Lisette saying he will await in the next two hours. Because he is eager to talk to Lisette, he may not want to rely on Lisette's will to call him back. In order to be aware of Lisette's presence as soon as possible, he may defer a glance to Lisette for the next half hour and delegate the glance task to the system.

- **media**: specifies the type of media involved in the connection. It may be <video, audio, text, etc.>.

- **speed desired**: characterizes the speed desired for transferring information <high, low, medium>. Note that the desired speed may be distinct from the speed that will be obtained effectively.

- **quality desired**: characterizes the quality desired for transferring information. It may be <high, medium, low, etc.>. Note that the desired quality may be distinct from the quality that will be obtained effectively.

For example, a glance connection is defined as the 7-uple: (target = another user, source = myself, orientation = one-way-in, duration = instantaneous, occurrence = now, media = video, desired speed = high, desired quality = medium,).

Similarly, predefined task-oriented connections described in the EuroCODE document can be expressed using 7-uples.

One first benefit of the formal description of the concept of connection opens the way to the definition of new connection types. For example, users may be able to define their own glance connection using an audio port: (target = another user, source = myself, orientation = one-way-in, duration = finite, occurrence = asap, media = audio, speed = high, quality = high).

The ability for users to define new connection types by combining attribute values provides more flexibility but has a direct impact on the definition of access control. Access control will have to be defined in terms of attribute combinations. It will no longer be based on predefined task names only.

Our description of the concept of connection spawns news options and questions that will be developed in section 5.

4.2. Level of availability

According to the original motivation of the EuroCODE designers, the level of availability of a particular user provides other users with a general sense of his current willingness to accept intrusion.

We have identified seven levels of availability:

- fully available (door is opened)
- half available (door is ajar)
- busy but willing to be interrupted (door is pushed)
- busy (door is closed)
- very busy (door is locked)
- out of the office (user should be back later in the day)
- out of town (there is no hope to contact the user)
- not logged in (user is either in or out of town)

In Cave, the notion of availability is confused with that of access rights. For instance, if the door is closed, glance operations are forbidden. We think that the level of availability and access rights are two orthogonal concepts. A user may be busy for the day but is willing to accept any type of connection with a particular person. Thus, if a caller has the right to vphone a busy user, it should be up to the caller to decide whether the target user should be interrupted or not (social control as opposed to technical control). The only problem we envision with our approach is the case where a target user shows a fully open door but denies a particular user any form of access. In such a case, should the system detect this type of social problem and propose the target user to show, for example, a closed door? Or should the system be dishonest and take the decision to show a closed door anyway?

The level of availability has two complementary roles: it shows other users the willingness of the user to accept intrusion and, in case of intrusion, it indicates the system the type of warning messages to use. For example, if you are busy brainstroming with other collegues, although you accept to be glanced, you may not want to hear audio warnings about glance connections. From this analysis, we may spawn new questions: should “before” and “after” messages be attached to level of availability as well?

See section 5 for an expansion of the QOC skeleton about availability.

4.3. System state for callers

A number of information hold by the system is of interest to the caller. It includes information about uncertainty, information about the target, and information about the local workstation.

- information system state is uncertain (or transient) while the system state is being consolidated across the network. This corresponds to the concept of “grey area” as used in operating systems and DBMS. In order to ensure system honesty, users must be aware of grey areas.

- information about the target state. These include:

  - status and properties of the physical target resources. For every resource used in a connection, the caller should know whether the resource is busy or available, whether it is turned on or off, etc. Properties include quality level, transmission rate, etc. Knowledge of physical properties will have an impact on the caller’s expectations. If, for example, a high quality transfer is desired and "real time" cannot be guaranteed, then a deferred offline transfer through a file should be made available;

- level of availability of the target user;

- access rights. For every type of connection, the caller should know whether the target user will accept, will reject, may accept or reject on the fly (dynamic control);

- location of the target user. If the target user is next door, the caller will expect the system to perform fast and high quality connections. Knowledge about the target's location in terms of distance, city name, etc., will have an impact on the caller's expectations in terms of response time conformance.

• cost of communication (in terms of charging money)

• information about the local workstation. These include primarily:

  - the availability level of oneself;

  - status and properties of the physical local resources. For every resource used in a connection, the caller should know whether the resource is busy or available, whether it is turned on or off, etc. In particular, the user should know the number and types of connections that are engaged and that can be engaged in the current state.

We have identified <u>what</u> system state variables are of interest to the caller. <u>When</u> should these state variables be disclosed to the caller? Should all of them or some of them be provided proactively? if so, the caller is able to predict the outcome of the attempts. Should they be provided after the fact? If so, the user may be informed about the exact reason of the failure or success. Should they be provided proactively with the possibility of knowing why a connection would fail? Such questions are refined in 2.4. <u>How</u> and <u>where</u> questions will not be addressed.

4.4. System state for target users

The target user should be aware of which connections are being performed or has been attempted while he was away. Depending on the connection type, we need to make a distinction between "before", "after" and "attempts" warnings.

Before and attempt warnings should contain:
• caller's identification,
• connection type.

Should the form of warnings depend
• on caller's id?
• on target's level of availability?
• on connection type?

Should warnings be

- Unvoidable? If so, audio and/or visual
- Transient?

5. Extension and modification of the initial QOC skeleton

Options in Q1 must be rephrased to mirror our analysis for the concept of "connection". In addition, as discussed above, we propose that users be able to define their own task names by combining connection attribute values. Thus Q1 becomes:

Q1-what: What types of connection to define within the architecture?

O1.1: Defined in terms of connection attributes
C: completeness (coverage of all of the possibilities)

O1.2: Distinguished in terms of predefined user's task names; E, T

O1.3: User defined task names by combining connection attributes
C: flexibility
C: efficiency (adaptability to users' preferences= shortcut)
C: increase software development cost (extra coding)

In option O1.2, user's task names are those that are predefined in the system whereas in O1.3 task names are those defined by the user by combining connection attributes.

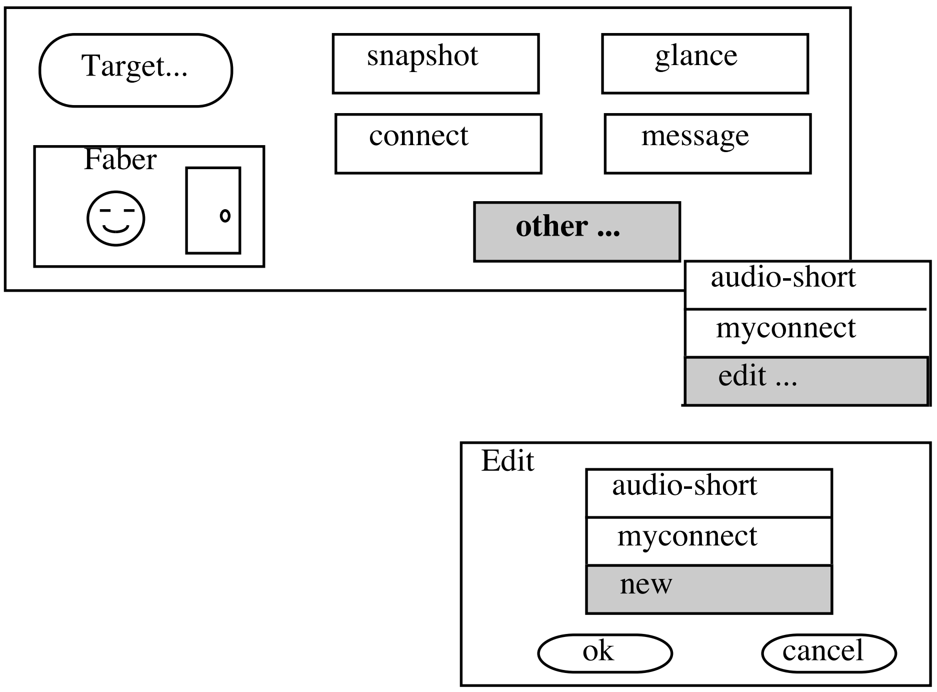
Figure 1 shows one possible way of making O1.3 available to the user. The "edit" and "connection" windows could be grouped together as in the Define Styles command in WORD. The ability for the user to define new connection task names spawns a new question (Q1.3) about the layout and accessibility of the task names.

Q1.3 spawns from O1.3. Q1.3 should be linked to Q5, Q6, Q7 and Q8, Q9, Q10 of the skeleton provided in RP3-ID-IR6.

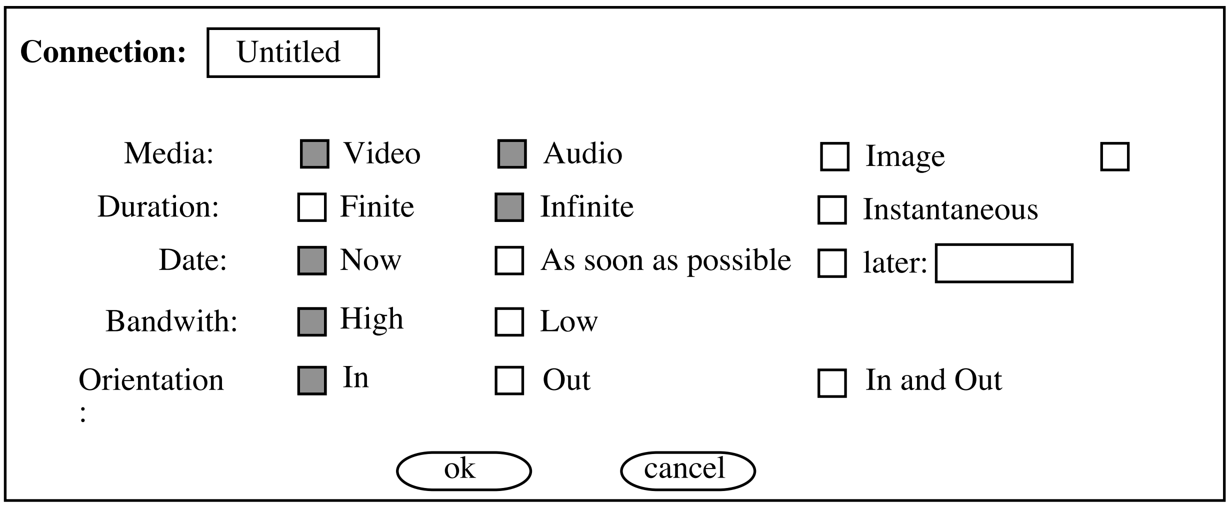
Q1.3.how: How AV connection task names be rendered?

O1.3.1: as buttons
O1.3.2: as pop-down
O1.3.3: as a combination of both (frequent tasks as buttons, unfrequent as pull-down)

O1.3.3 spawns new question: shall the combination be decided by the system or dynamically controllable by the user (preferences)?



(a)



Connection: Untitled

Media:	[✓ Video]	[✓ Audio]	☐ Image	☐
Duration:	☐ Finite	[✓ Infinite]	☐ Instantaneous	
Date:	[✓ Now]	☐ As soon as possible	☐ later:	
Bandwith:	[✓ High]	☐ Low		
Orientation :	[✓ In]	☐ Out	☐ In and Out	

(b)

Figure 1: How to define a new type of connection. A sketchy layout.
(a) Create a new type of connection.
(b) Parameters to specifiy a type of connection.

In the current QOC skeleton, we have not seen any explicit design rationale that justifies the structure of the ECOM connection window. In the informal description it is said that this window is composed of three parts: connections to people, connection to oneself and to non personnal nodes, and the level of availability of oneself. We suggest a different organization for the connection window and propose two new question Q’1 and Q’2.

Q1-how: How should connection window be structured?

O1.1: Structure based on non personal /personal target; EC
C: Hittability
O1..2: No structure (see figure 2 for a proposal inspired from current ECOM)
C: Low screen clutter

Q2-how: How should availability level be displayed?

O2.1: as a palette of possibilities with curent selected level highlighted; EC
C: Hittability
O2.2: as a pull-down menu with current selected level as header of the menu
C: Low screen clutter

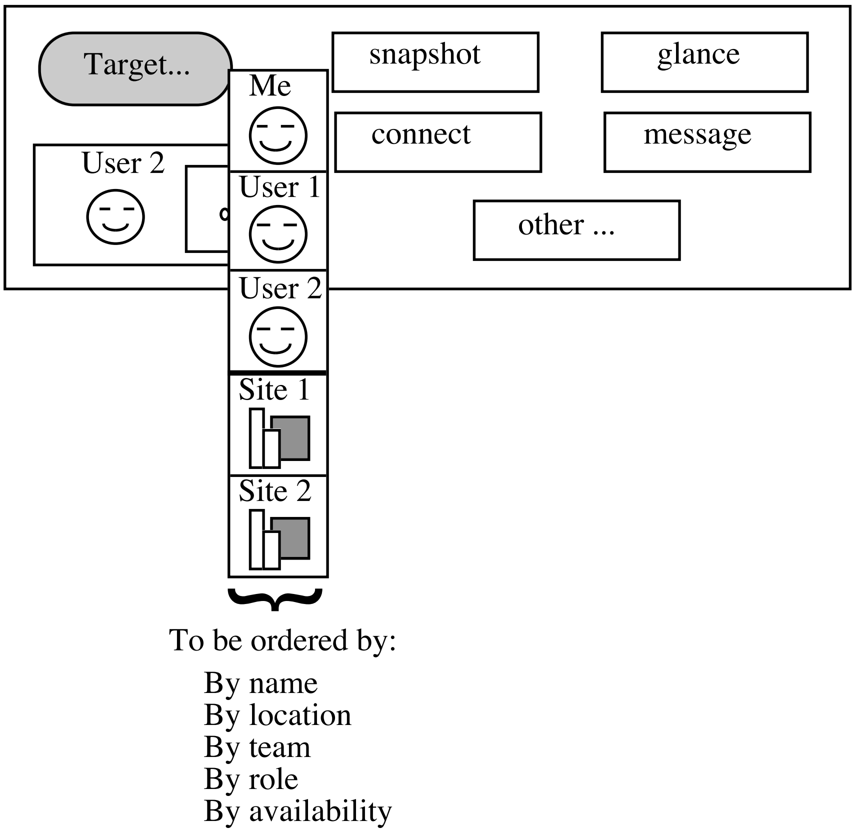


Figure 2: Define the target of a connection. Targets could be organize as a two level menu: one level for category (windows, sites, users) and the second level for the instances in the category.

Analysis in 4.3 (system state for callers) spawns the following “what” questions not addressed in the QOC original skeleton:

Q2-what: what system variables should be observable to the caller?

Q3-what: what system variables should be observable to the callee (target)?

We develop Q2 first.

Q2-what: what system variables should be observable to the caller?

- O2.1: about target
- O2.2: about oneself
- O2.2: about uncertainty (software grey areas)
  - C: Support honesty
  - R: openness of commitment algorithms

To be realistic, O2.2 requires that the algorithms that are in charge of managing grey areas be able to dynamically communicate their state to other software components (e.g., the user interface).

O2.1 and O2.2 spawn questions Q2.1 and Q2.2 respectively.

Q2.1-what: What variables about target should be observable to the caller?

- O2.1.1: Status and properties about physical resources (see 2.3.3 for more details)
  - C: Support expectation about system response time and quality of transmission
  - C: Increase confidence (e.g., feedback about the status of the connection: "trying to contact target", "target notified", "waiting for target to answer")

- O2.1.2: Level of availability (see 2.3.2 for detailed analysis of what type of availability)
  - C: Support social control

- O2.1.3: Access rights (spawns Q2.1.3 below)

- O2.1.4: Location (see 2.3.3)
  - C: Support expectation about system response time

Q2.2-what: What variables about oneself should be observable ?

- O2.2.1: Status and properties about physical resources
  - C: Low human memory load

- O2.2.2: Level of availability
  - C: Low human memory load

- O2.2.3: Ongoing connections
  - C: Low human memory load

Option O2.1.3 (access rights about the target) spawns the following question:

Q2.1.3-what: What information about access rights

O2.1.3.1: Connection type
O2.1.3.2: Target will reject (i.e., access denied)
O2.1.3.3: Target will accept (i.e., access permitted)
O2.1.3.4: Target may accept (i.e., on the fly dynamic control by target)

In turn, option O2.1.3.2 raises a new question:

Q2.1.3.2-who: Who is in charge of elaborating messages for notifying rejection?

O2.1.3.2.1: By target
C: Supports customization (interpersonal relation)
C: Supports informativeness
C: Additional task (spawns new questions about how and when to specify message)

O2.1.3.2.2: By system
C: Supports anonymity
C: Ease of implementation

So far, we have developed question Q2-what (what system variables should be observable to the caller?). The How questions will not be developed. We need now to look at temporal issues.

Q2-when: When system variables be observable to the caller?

O2.1: Proactively (i.e., as value changes)
C: Low errors due to proactive feedback
R: Browsability of system state to reduce screen cluttering
R: Active data base to support conformance and honesty
R: Efficiency of the underlying system to support response time conformance

O2.2: A posteriori (e.g., after a connection has been attempted)
C: Support explanation about failures

O2.2: Proactively with facility to get explanation
C: Low errors
C: Support explanation about failures
R: Browsability of system state to reduce screen cluttering
R: Active data base to increase conformance and honesty
R: Efficiency of the underlying system to support response time conformance

As demonstrated by the above when questions, we have considered the state variables globally. It may be useful to develop “when” questions at a finer grain.

Having QOC’ed about the caller, we need to switch to the target:

Q3-what: What system variables should be observable to the target?

O3.1: Events about “connection attempt” (type, date, caller id)
O3.2: Event “begin of connection” (type, caller id)
O3.3: Event “end of connection”

Q3’-what: On which ingredient notification rendering should depend?

O3.1: Level of availability of the target
O3.2: Caller
O3.3: Type of connection
O3.4: Combination of the above

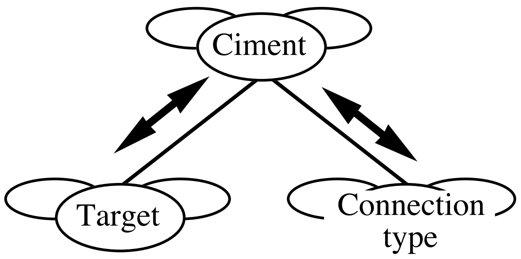
6. PAC-Comments

Issue 1: Making and Breaking AV

Q11: In what order should user specify connection type and target?

O11.1: Either; when both selected hit confirm button to fire up message window or connection
O11.2: Connection type then target
O11.3: Target then connection type; E, EC

Same agents whatever the option is:



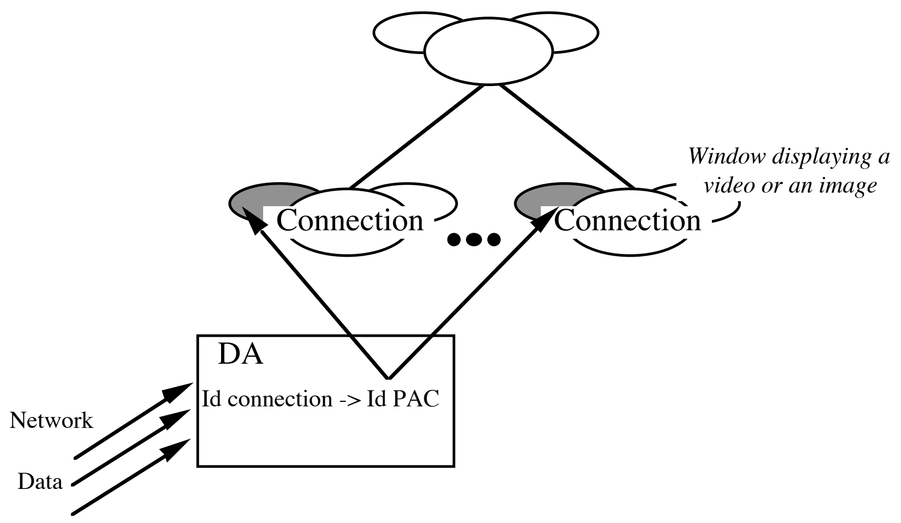
Q12: How to exit communication mode?

O12.1: On termination of active use; E, T, EC ->
O12.2: On selection of new mode

Q13: How to deselect communication target?

O13.1: On termination of the communication; T
O13.2: On selection of a new target; E, EC

Many connections at a given time (if technically feasible):
Benefits of the agent approach: one agent per opened connection


Window displaying a video or an image
Connection
Connection
DA
Id connection -> Id PAC
Network
Data

7. PAC-Amodeus architecture for current ECOM interface and software design rationale

In this section, we present the software architecture of the ECOM system according to its current design state. For doing so, we have used Figures 5 and 6 in Document RP3-ID-IR6. In paragraph 7.2, we show how QOC has been used to trace our software design decisions.

7.1. PAC-Amodeus architecture

Figure 4 shows one possible PAC-Amodeus software architecture for implementing the current ECOM system. Figure 5 presents the software agents hierarchy refining the Dialogue Controller.

On one side of the arch, the Low Level Interaction Component (LLIC) denotes the underlying software and hardware platform. It receives mouse and keyboard events from the user. It also manages the presentation and contains functions to play video inside a window, functions to produce sounds and functions to display graphics on screen. The Presentation Techniques Component (PTC) bridges the gap between the Dialogue Controller and the LLIC. The Dialogue Controller (DC) is then independent of the functions playing video for example.

On the other side of the arch, the Functional Core (FC) corresponds to the functions to send and receive messages along the network. The Functional Core Adaptor (FCA) allows communication between its two surrounding components by implementing a communication protocol. It is therefore possible to receive information through the network and to handle user events: This is managed within the DC organized as a hierarchy of PAC agents.

Users' states (e.g., their access rights and availability) give rise to the question: Where to maintain this information? Two options are possible:

- One option is to duplicate the data base about access rights on each workstation. The data base will therefore be located inside the FCA. This option will guarantee the stability of the response time as far as no request is sent through the network. On the other hand, it will increase the number of messages through the network to update the duplicated data bases.

- Another option is a centralized or distributed data base that is not local to the workstations. in this case, the data base is maintained by the FC. It is a good solution to reduce the number of messages. But the response time stability for displaying the status of a particular user (necessary to provide pro-active feedback) is no more verified since response time depend on network load.

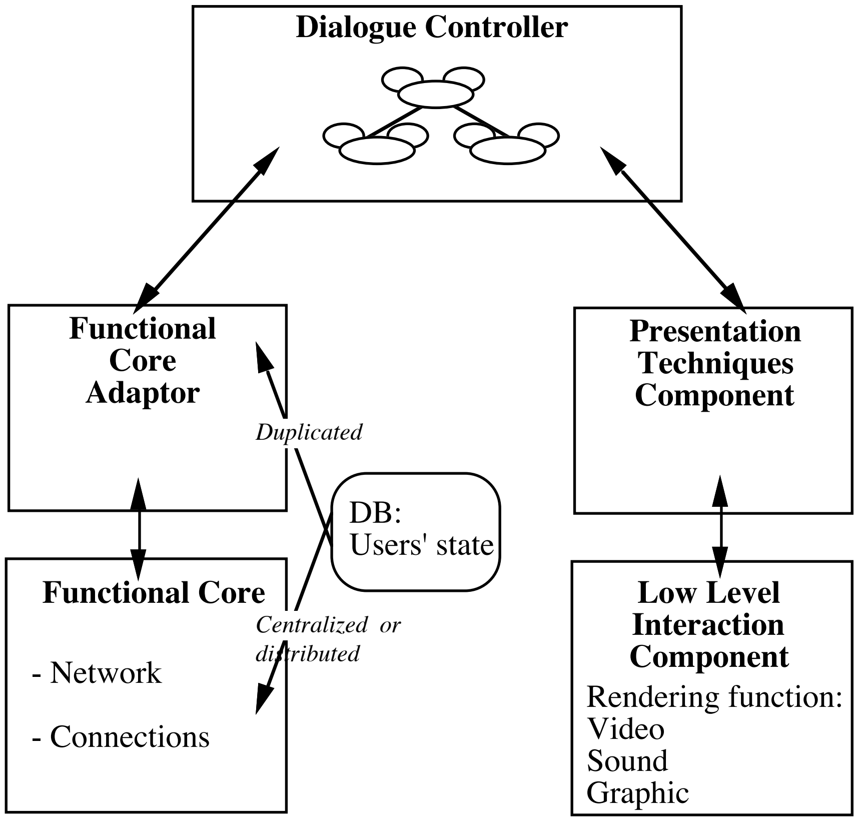

Dialogue Controller
Functional Core Adaptor
Presentation Techniques Component
Duplicated
DB: Users' state
Functional Core
- Network
- Connections
Centralized or distributed
Low Level Interaction Component
Rendering function: Video Sound Graphic

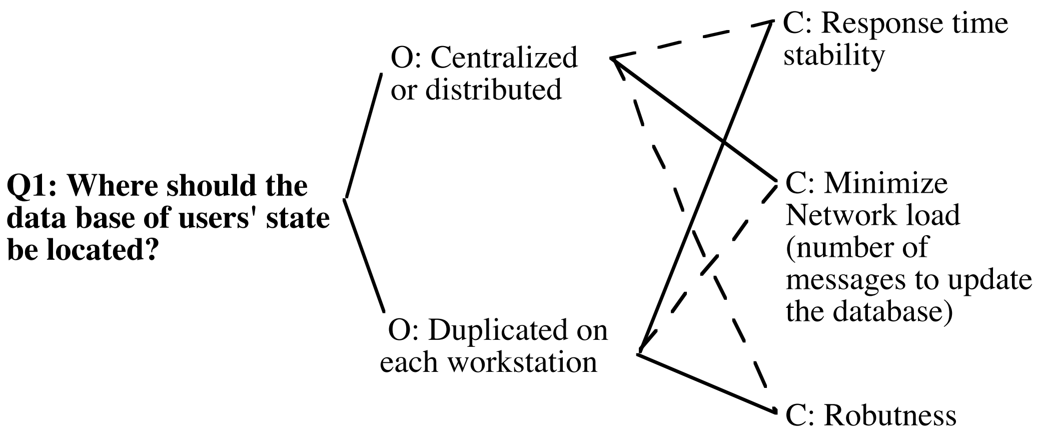
Figure 4: Software components implementing the ECOM system (applying PAC-Amodeus model).

We synthesize this design issue "Where to locate the data base?" within a QOC space:

Q1: Where should the data base of users' state be located?

O1: Centralized or distributed

O2: Duplicated on each workstation


Q1: Where should the data base of users' state be located?
O: Centralized or distributed
O: Duplicated on each workstation
C: Response time stability
C: Minimize Network load (number of messages to update the database)
C: Robutness

At the top of the arch, the Dialogue Controller is comprised of PAC agents: Figure 5 shows the two level hierarchy.

• The root agent "Make a connection" is in charge of the global control of the interaction with the user. The Presentation facet corresponds to the buttons to establish a connection as shown in figure 5. The Abstraction part maintains the current connection to be established. At the creation of a connection, the Abstraction facet sends the request through the FCA to the network. To establish the connection, the "make a connection" agent dynamically creates an instance of a Connection agent.

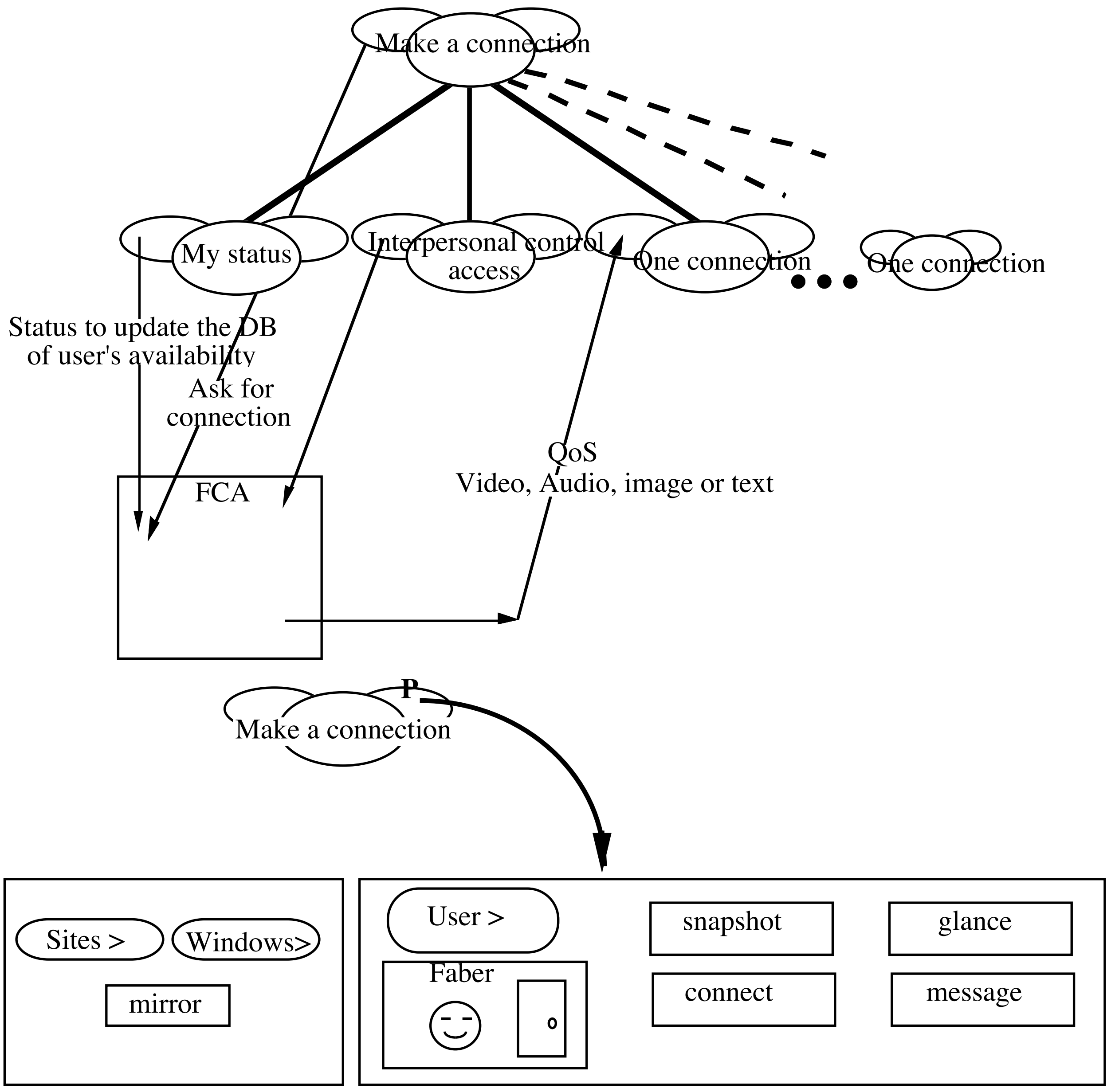


Figure 5: PAC agents organizing the Dialogue Controller component.

• The "Connection" agent corresponds to a single connection. Its Abstraction part receives information to be displayed such as a movie from the FCA (FCA corresponds to the interface with the network). This software design allows the user to establish multiple connections at a given time. Each connection is managed by a dedicated agent.

• The "Status" agent models the window where the availability of the user is displayed. Its Abstraction facet maintains the current availability of the user. When the user selects the Exceptions button, the corresponding event is received by the Presentation facet. The "Status" agent sends a message to the "Make a connection" agent which will in turn create or make active the "Interpersonal control access" agent.

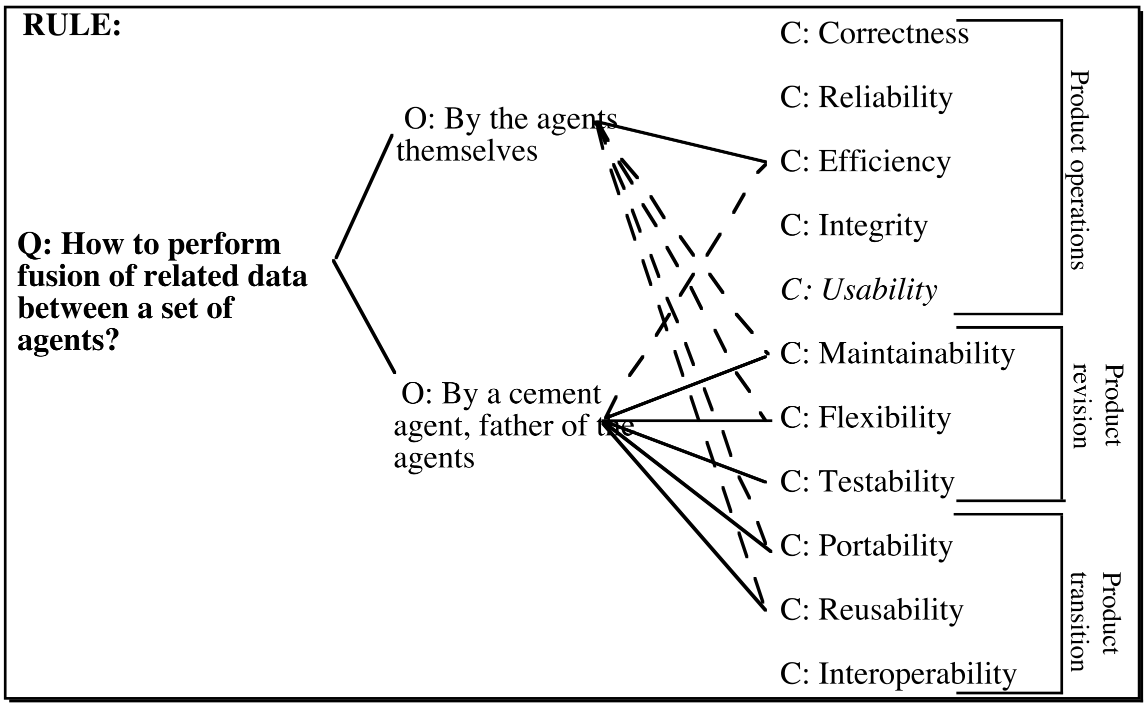
• The "Interpersonal control access" agent corresponds to the form in figure 5 in Document RP3-ID-IR6. Its Abstraction facet maintains the exceptions defined by the user.

7.2. QOC Space to justify the hierarchy of PAC agents

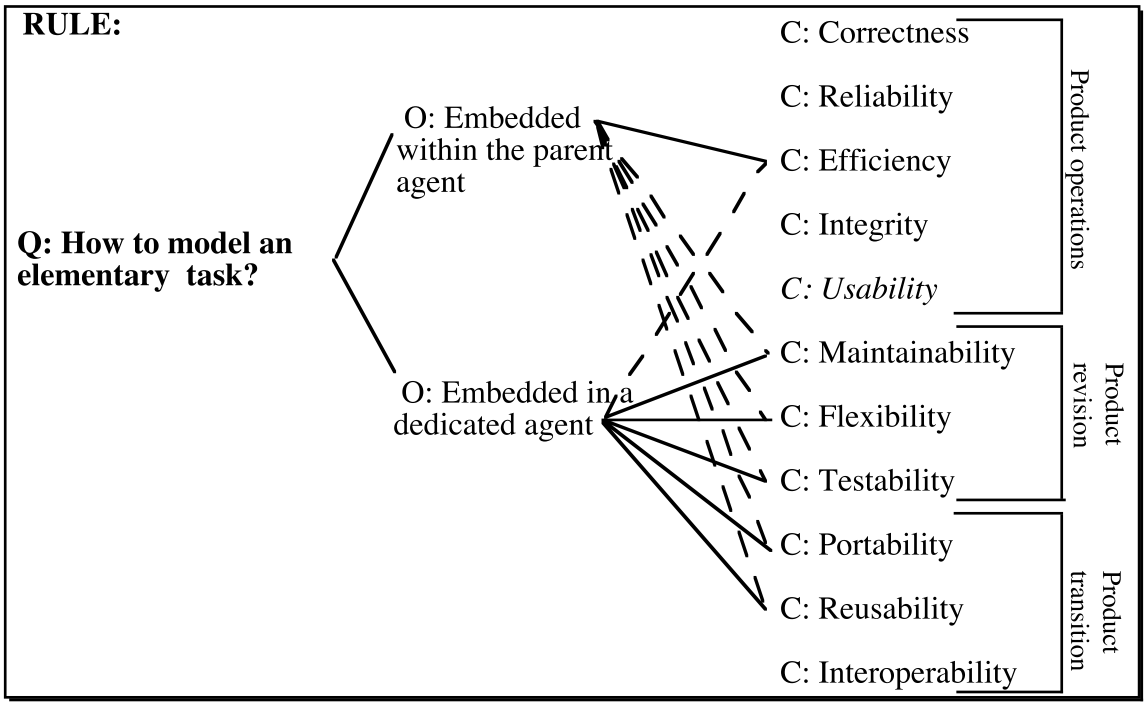
7.2.1. A generic QOC space

One can use the set of heuristic rules presented in [Nigay&Coutaz 92, Nigay 94] to derive the hierarchy of PAC agents of the Dialogue Controller. These rules help defining the levels of abstraction that are necessary to the interpretation of user's inputs and to system state rendering.

For example, one of these rules proposes that a cement agent be used to perform the fusion of user's inputs when these inputs are distributed over multiple agents and when their combination "makes sense" to the system. Another option is that the combination of user inputs be performed by the agents themselves. For doing so, they would need to exchange messages directly. It may be more efficient but they would not be reusable independently. Clearly, our analysis draws upon software engineering criterias such as those devised by McCall (see Annexe A and B for a complete list). (Remark: McCall refines each factor in terms of criteria. In this paper we use the first level only, i.e., the factors.)



Another general interesting question applies to the leaf agents of the hierarchy: In general, leaf agents are introduced to support elementary tasks. Do we need a dedicated agent to manage this task or do we delegate this competence to the root agent? We address this question using the following QOC space.



The above QOC space is generic and does not address any particular application. The QOC space expresses two general rules:

- RULE 1: Minimizing the number of levels within the hierarchy as well as the number of agents is good for efficiency: it reduces the message communication between the agents.
- RULE 2: Adding agents increases modularity. Modularity is directly related to the criteria corresponding to product revision. Adding agents is therefore good for maintainability, flexibility and testability. Modularity is also related to the criteria "product transition" and in particular portability and reusability.

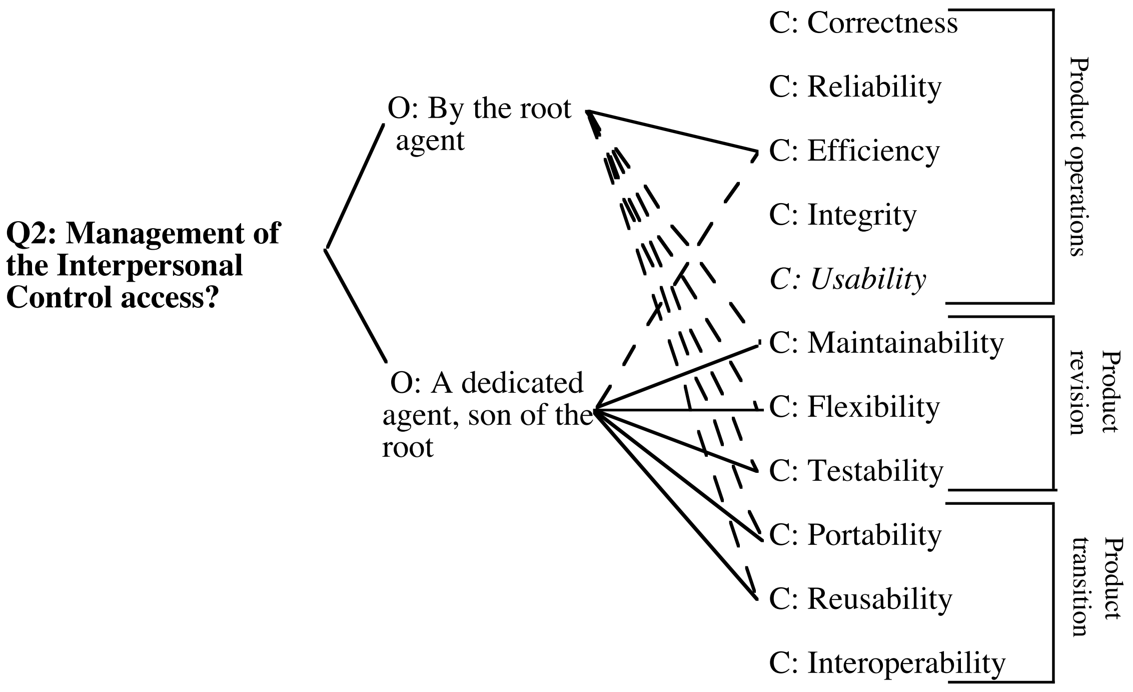
7.2.2. The case study ECOM

We have applied the above generic QOC space to the case of ECOM. For example, while designing the hierarchy of figure 5, we addressed the question about the management of the Interpersonal Control access. Do we need an agent to manage the Interpersonal Control access. If not, it is then managed by the root agent. The corresponding QOC space is described below:

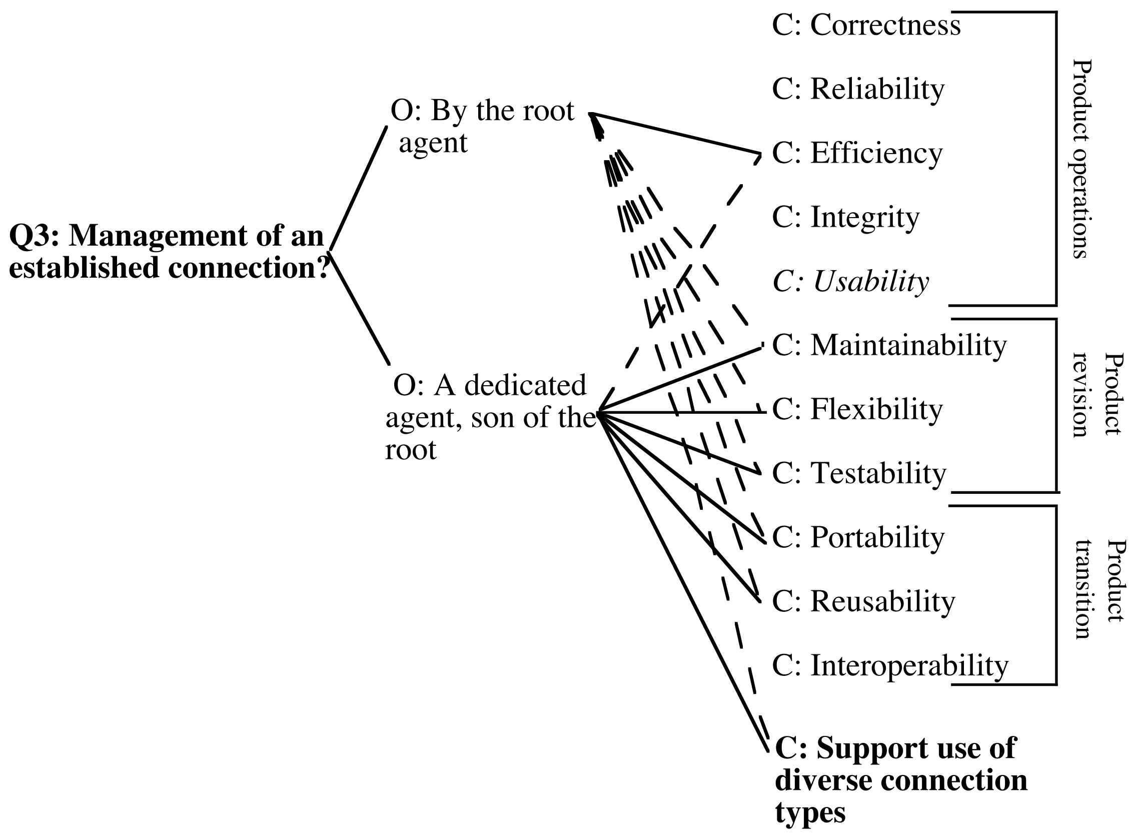
Q2: Management of the Interpersonal Control access?

O1: By the root agent

O2: A dedicated agent, son of the root



Same question arises about the management of an established connection. In this particular case, having a dedicated agent per connection is an easy way to handle different connections at a given time. This criteria is added to the list of software criteria.



References

[McCall 77]

J. McCall. *Factors in Software Quality*; General Electric Eds., 1977.

[Nigay, Coutaz 92]

L. Nigay and J. Coutaz. Document D18, BRA AMODEUS 1, 3066, 1992.

[Nigay 94]

L. Nigay. Conception et Modélisation logicielles des systèmes interactifs : application aux interfaces multimodales. Thèse de doctorat de l'Université Joseph Fourier, Grenoble, Janvier, 1994.

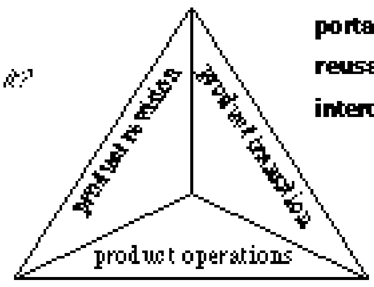
Appendix A: Definition of McCall factors

Factor	**Definition**
Correctness	Extent to which a program satisfies its specifications and fulfils the user's mission objectives.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Flexibility	Effort required to modify an operational program.
Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
Interoperability	Effort required to couple one system with another.
Maintainability	Effort required to locate and fix an error in an operational program.
Portability	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
Reliability	Extent to which a program can be expected to perform its intended function with required precision.
Reusability	Extent to which a program can be used in other applications-related to packaging and scope of the functions that programs perform.
Testability	Effort required to test a program to insure it performs its intended function.
Usability	Effort required to learn, operate, prepare input, and interpret output of a program.

Appendix B: Classification of McCall factors

- operational characteristics (product operations);
- ability to support changes (product revision); and
- capacity to adapt to new environments (product transition).

maintainability	*Can I fix it?*
flexibility	*Can I change it?*
testability	*Can I test it?*



portability	*Will I be able to use it on another mac*
reusability	*Will I be able to reuse some of the sof*
interoperability	*Will I be able to interface it with anothe*

correctness	*Does it do what I want?*
reliability	*Does it do accurately all of the time?*
efficiency	*Will it run on my hardware as well as it can?*
integrity	*Is it secure?*
usability	*Can I run it?*