9 Example: Initial development

Example 1: Initial development

Introduction

What

By reading this theme it is possible to learn TIMe “by example”. We follow a company that develops and produces access control systems according to the TIMe Process models.

In order to navigate freely among the examples and their corresponding methodological base, the reader should consult the generic project diagram.

The company

For many years, the Sesam Sesam company had great success with their door locks and system keys. Their selling point was the highly flexible way that keys and locks could be coded to give user groups different access rights in a building complex.

But even their system had two main drawbacks:

1. *Lost keys*. Whenever a key was lost, they had to change the locks to prevent unauthorised persons to gain access. The cost for new keys where not high, but the cost for changing the locks and the security risk involved was too high.
2. *Code limitations*. Although the system was very flexible, it was based on fully mechanical locks and keys with inherent limitations in the coding that could be achieved.

To overcome these problems, and to stay in front of competition, the Sesam Sesam people were continuously looking for improvement opportunities. They saw that electronics and computers were rapidly becoming attractive alternatives as the prices went down and the reliability up. They also heard rumors of competitors looking at the new technology, and decided to start planning a new product family. Being a very systematic and mature company they adopted TIMe as their methodology, and started off using the Developing from scratch processes of TIMe.

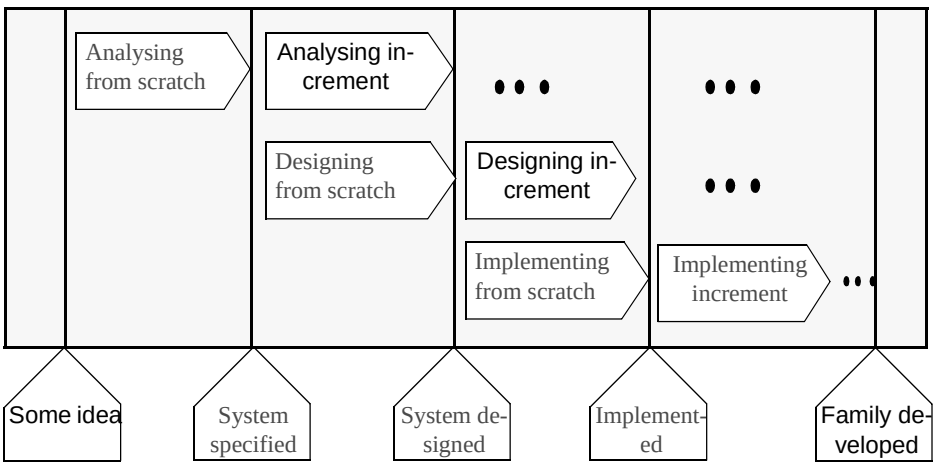


Figure 9-1: Developing from scratch

They set up a team consisting of senior people from development, marketing and production, and a steering committee that involved the top management (since this was a key strategic issue for the company).

The first task was Analysing from scratch (p.9-4), which contains an internal milestone where the technical feasibility and the business potential can be evaluated. If the evaluation turned out to be favourable, they would continue with Specifying from scratch to develop specifications and then Designing from scratch and Implementing from scratch. Each of these would result in a milestone with Documentation (p.9-68).

The development process is intimately connected to the production of a set of descriptions. The relation between the different descriptions and the various milestones representing the progress of time is found in the map of descriptions and milestones. The figure contains both references to the strategies and activities associated with a certain description and it gives examples of the various descriptions taken from this initial development.

Analysing from scratch

This activity involves the activities in:

- Analysing Domain (p.9-4).
- Analysing Requirements (p.9-18).
- Application Specification (p.9-21).

and the resulting descriptions are found in Domain Descriptions (p.9-68).

Analysing Domain

For the domain analysis of the Access Control System we use the strategy for domain analysis which involves:

1. Make Domain Statement (See also activity) (p.9-4)
2. Make Dictionary (see also activity) (p.9-5)
3. In parallel:
   - Make Domain Object Model (see also activity) (p.9-6)
   - Make Domain Property Model (see also activity) (p.9-7)
4. Harmonise Domain Descriptions (p.9-13)

Our first step is to identify and understand the most important concepts of the domain. At first we are less interested in the interrelationships than the pure concepts of the subject. This is done by making a Domain Statement and a dictionary.

Make Domain Statement (See also activity*)*

The Domain Statement may in the first round be made for the Domain, without any consideration of a system. An extract of the Domain Statement can be seen in Figure 9-2 "Domain Statement V1" (p.9-5). The complete Domain Statement made according to the guidelines can be found in Domain statement V2.

Figure 9-2: Domain Statement V1

Open figure

Area of concern

Access control has to do with controlling the access of users to *access zones*. Only a user with known identity and correct access right shall be allowed to enter into an *access zone*. Other users shall be denied access.

Stakeholders

Users of the system, those responsible for the security of the access zones.

Services

The user will enter an access zone through an access point.

The authentication of a user shall be established by some means for secret personal identification (code). The authorisation is based upon the user identity and access rights associated with the user.

A supervisor will have the ability to insert new users in the system.

Users shall be able to change their secret code.

Helpers

We assume some central means to establish access rights automatically.

Make Dictionary (see also activity*)*

If we just consider the concepts of the domain statement and not consider any system, we get the first version of the dictionary, see Figure 9-3 "Domain specific Dictionary" (p.9-6).

Figure 9-3: Domain specific Dictionary

Open figure

Access point	A point of access into an access zone.
Access zone	A physical or logical zone guarded by a set of access points.
Authentication	To establish the identity of a user.
Authorisation	To establish the right of a user to enter an access zone.
Authorizer	The entitity which determines authentication and authorisation.
PIN	A personal identification means.
User	A person with known identity with authorisation to enter specific access zones.
User name	A user name.
Access Granting	The role of granting (or not granting) a user access.

The first dictionary will always be very sketchy and we shall accept to maintain the dictionary throughout the project.

Make Domain Object Model (see also activity*)*

The next step is to make the Domain Object Model - this includes to describe the main concepts as classes, their relations, connections, attributes, aggregation, localisation and possible generalisation hierarchies.

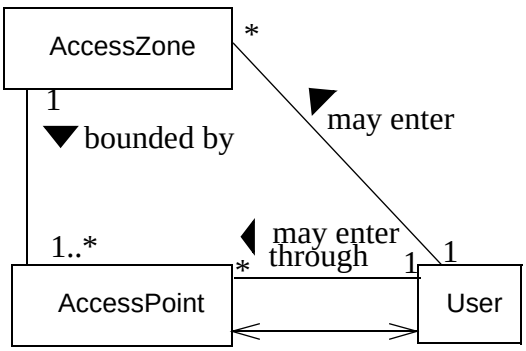
The starting point is the domain statement (Figure 9-2 (p.9-5)) and the dictionary (Figure 9-3 (p.9-6)).

Domain Object Model, classes, relations and connections

The most general object model of a Domain is an object model with the identified classes and their relations as in Figure 9-4 "The access control domain" (p.9-6). These classes come about by studying the Domain Statement and the Dictionary.

Figure 9-4: The access control domain

Open figure



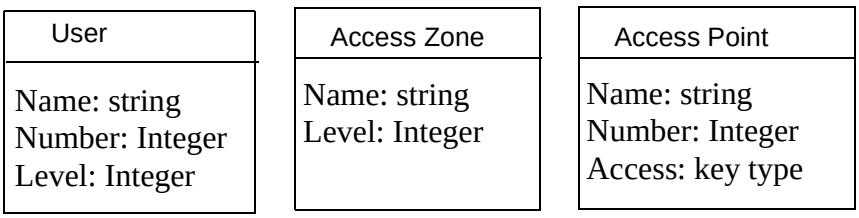
This is a Domain Object Model of the access control domain, expressed in the UML notation. We recognise the main concepts in the dictionary as classes in the diagram. Relations help to understand the Domain. We know that the User and AccessPoint objects will be active objects, while AccessZone objects are passive. This is indicated by the communication connection between AccessPoint and User. It may later turn out that there are more active objects.

Domain Model, attributes

We then identify the necessary attributes, see Figure 9-5 "Attribute specification" (p.9-7). These may either be obvious from the domain or they may come as a result of required properties.

Figure 9-5: Attribute specification

Open figure



Domain Object Model, generalisation/specialisation

At this point in the development, we find no reason to define generalization relations. Informally we may consider access points of different specializations such as unidirectional access points and bidirectional access points. Furthermore we may consider different specializations of what security is needed. Some access points may only require a proper physical identification such as a card, while other access points may require the presentation of a secret personal code.

Such considerations are currently deferred.

Domain Object Model, aggregation, localisation

We conclude that for this example there is no need for aggregation or localisation at this point.

Make Domain Property Model (see also activity*)*

We have identified the User as a concept in the domain. We know that the User will require four functions, involving access points and possibly other objects:

- User Access (p.9-8)
- PIN changing (p.9-9)
- New User (p.9-10)

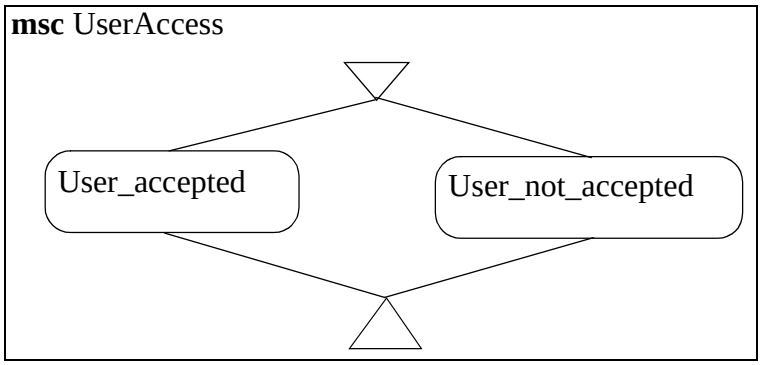
For these we make mscs describing the most important cases of interaction.

User Access

The handling of the user access is at the very core of our concerns. From the domain statement (Figure 9-2 (p.9-5)) we read that users either get access to an access zone or they do not. This can be expressed through a simple MSC-96 model of the service.

Figure 9-6: User Access

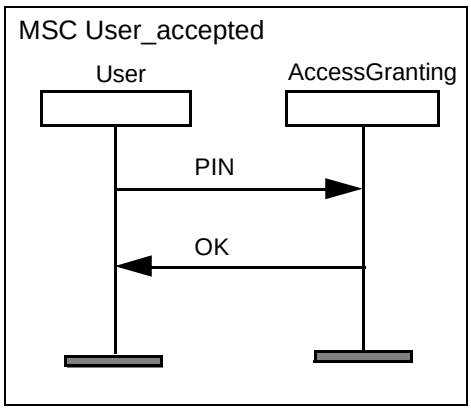
Open figure



From Figure 9-6 "User Access" (p.9-8) we cannot determine which entities are involved in the service, but from Figure 9-7 "MSC User_accepted" (p.9-8) we define that the user communicates with a *Access Granting* role to determine his desired access to the access zone.

Figure 9-7: MSC User_accepted

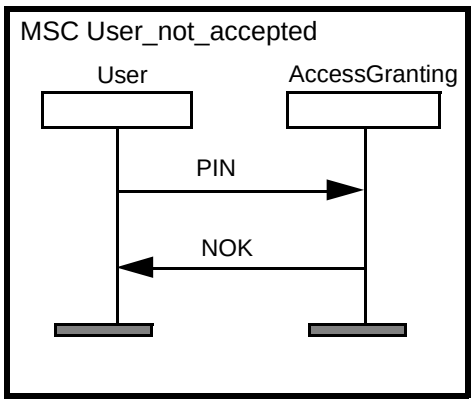
Open figure



The other alternative is not more complicated.

Figure 9-8: MSC User_not_accepted

Open figure



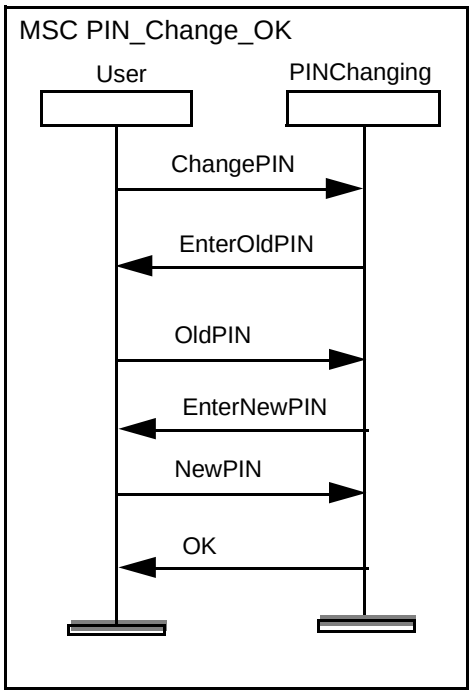
The reason why we introduce an Access Granting role and not just let AccessPoint do the access granting is that this is not obvious from the domain descriptions. We only know that the User uses the access points in order to get access, but it may be so that access points are just interface objects and that the real validation of users is done by some other objects not identified yet.

PIN changing

One of the scenarios when the PIN code is changed is described in Figure 9-9 "MSC User changing PINwith success" (p.9-10).

Figure 9-9: MSC User changing PINwith success

Open figure



Note that PINChanging in is not a class in the Domain Object Model, but a functional role.

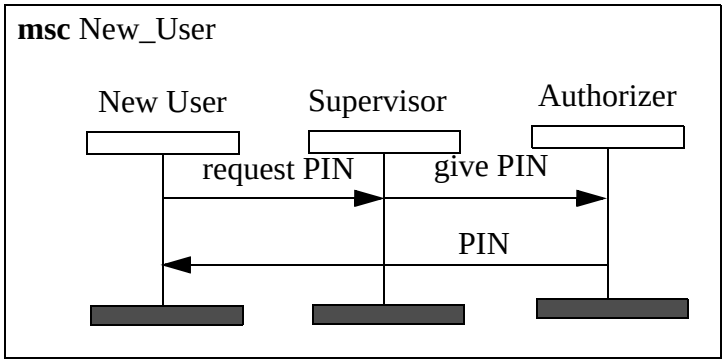
The reader should note that our property model of a PIN Change only shows one successful scenario. The unsuccessful ones are not described. This is currently deferred, but we shall return to the problem of incompleteness and impreciseness shortly.

New User

There is definitely also a need to allow new users access to the zones. We have in the domain statement Figure 9-2 "Domain Statement V1" (p.9-5) made clear that only the supervisors can enter new users into the system.

Figure 9-10: New User

Open figure



We have now introduced another role Authorizer to show that we do not expect the supervisor to take personal care of the access of all users. There is some automatic means which controls the access. This may be an electronic device or it may in principle be a human being. Nevertheless there is a system of access rights which is exercised by some active objects.

More services?

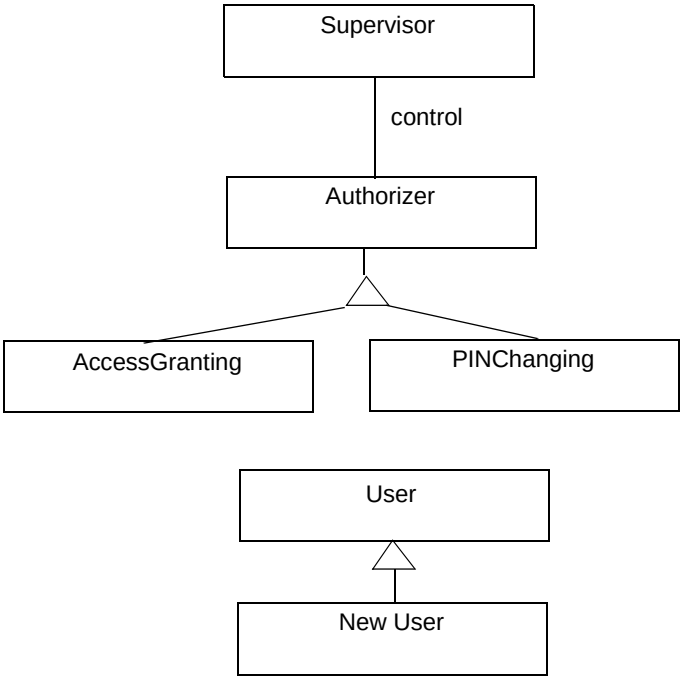
We have described three different services. It is obvious that more services are conceivable and most probably necessary. We may have the need to delete users from the system and we may want to define more diverse access points. For the sake of simplicity we shall keep to these three services in our initial development.

Role model

We have introduced some (functional) roles which describe in an abstract fashion the counterparts of the user when exercising the services. The roles are not necessarily independent of each other. We define an UML model which describe the relations between the roles.

Figure 9-11: Role object model

Open figure



We define here that the Access Granting and PIN Changing roles are actually specializations of the more general Authorizer which we introduced in our New User service.

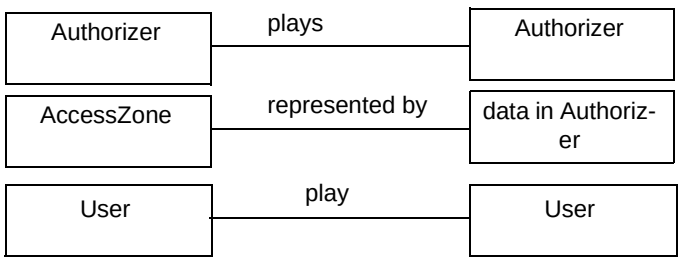
Casting

Having defined a tentative object model Figure 9-4 "The access control domain" (p.9-6) and a property model with role Figure 9-11 "Role object model" (p.9-12), we have the need to define the relations between the objects (classes) and the roles. In the domain this may not be entirely clear what objects should play what roles. The casting may be deferred to later stages in the development or contain unknown relations.

It is also the case that when trying to define the casting, new insight is achieved concerning the domain as such. In our case we saw the need to introduce an *Authorizer* object into the domain object model.

Figure 9-12: Casting Access Control

Open figure



We also see from Figure 9-12 "Casting Access Control" (p.9-12) that not all objects actually play roles since roles are normally played by active objects. The access zones are not active objects, but rather represented by passive data in other objects such as in the *Authorizer*.

Harmonise Domain Descriptions

When describing the domain we have approached the task from different perspectives. We have had the informal prose perspective, we have had the more rigid dictionary perspective; we have had the object model perspective and the property model perspective. The perspective have not been entirely independent as we have used the insight of the domain statement and dictionary in our work with object and property models.

Still the different perspectives are independent enough to produce new insight which must be carried over to all the other descriptions for consistency. This is especially true when we try actively to tie the descriptions together as we do with the casting.

Our casting Figure 9-12 "Casting Access Control" (p.9-12) results in an understanding which must be carried over to the object model.

Furthermore it is a matter of taste to what level of completeness, detail and precision the domain descriptions should be brought. It is reasonable to take at least one more iteration on all descriptions after the first round of sketching the domain.

Harmonize Domain Statement (see also activity***)***

We want to make the domain statement more complete and we make sure that we actually consider all the aspects laid down by the strategy for making domain statements.

Figure 9-13: Domain Statement V2

Open figure

Excecutive summary and area of concern

Access control has to do with controlling the access of users to *access zones*. Only a user with known identity and correct access right shall be allowed to enter into an *access zone*. Other users shall be denied access.

Stake holders

In addition to users, there will implicitly be owners of the access zones. The rules for which users are granted an identity are laid down by these owners, but this issue is considered to be outside the domain.

Passive Objects and associations

Access zones are passive objects. Their associations with other objects are completely given by the domain object model.

Active objects and connections

Users and access points are active objects. The connections are given by the domain object model. There are also supervisors (operators) who have the responsibility to determine the access rights of users to the access zones.

Services

1. Users with known identify shall be allowed access, while others shall be denied access.
2. Users shall be able to change their personal identification.
3. Only supervisors shall have the capability to insert new users into the system.

Harmonize Domain Dictionary (see also activity***)***

Here we recognize that we have introduced a number of new concepts during our work with the services and the role model. These concepts must be carried back to the dictionary.

Figure 9-14: Harmonised Domain specific Dictionary

Open figure

Access Granting	The role of granting (or not granting) a user access.
Access point	A point of access into an access zone.
Access zone	A physical or logical zone guarded by a set of access points.
Authentication	To establish the identity of a user.
Authorisation	To establish the right of a user to enter an access zone.
Authorizer	The entity which determines authentication and authorisation.
Authorizer	Also: the role of storing PINs
PIN Changing	The role of changing the PIN of a user
PIN	A personal identification means.
Supervisor	A person who controls the authorizer
User	A person with known identity with authorisation to enter specific access zones.
User name	A user name.

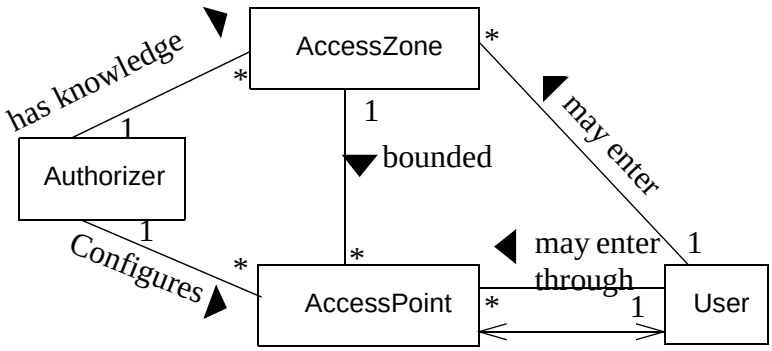
Harmonize Object model (see also activity***)***

We saw in Casting (p.9-12) that a better understanding of the domain could be achieved through introducing *Authorizer* as a concept in the domain. This insight may come early as a part of the domain analysis or it may come later as a result of more specific family and system analysis.

In our case the insight was achieved as a result of trying to perform casting in the domain analysis.

Figure 9-15: Domain model of Access Control V2

Open figure



In Figure 9-15 "Domain model of Access Control V2" (p.9-15) we see that the introduction of *Authorizer* has also triggered the redefinition of some of the relations.

Harmonize Property model (see also** activity**)

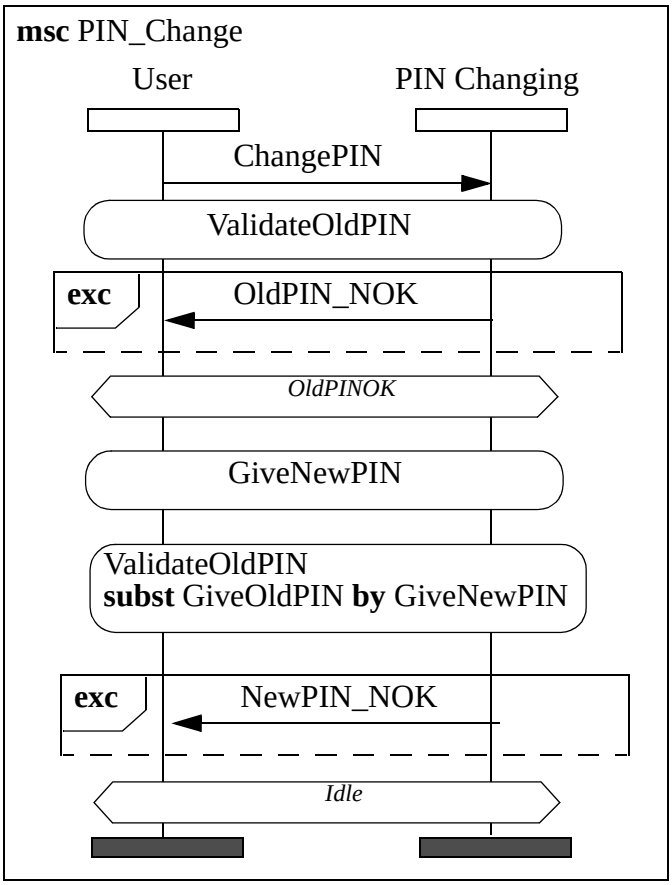
Property models are of course also affected by the changing of the object model, but here we shall focus on the strategy to make a property model more precise and more detailed. The need to make a property model of the domain more precise and detailed may come from external sources determined to understand the model, but failing to do so. It may also be the case that questions are asked about the model which it is not capable of giving an adequate answer to.

Here we merely refer to the through walkthrough of the PIN Change service as an example of how to develop property models. The reader should also consult the strategy for domain property modelling.

Here we show what the result of the process of making PIN change more precise and more detailed

Figure 9-16: Change PIN (MSC-96)

Open figure



In Figure 9-16 "Change PIN (MSC-96)" (p.9-16) we see that in addition to the successful cases we have introduced non-successful cases. Furthermore we have taken a stand on *who* should determine the PIN-code. That the PIN is to be changed does not mean that the user is allowed to choose his own PIN as it may be selected by the system (the

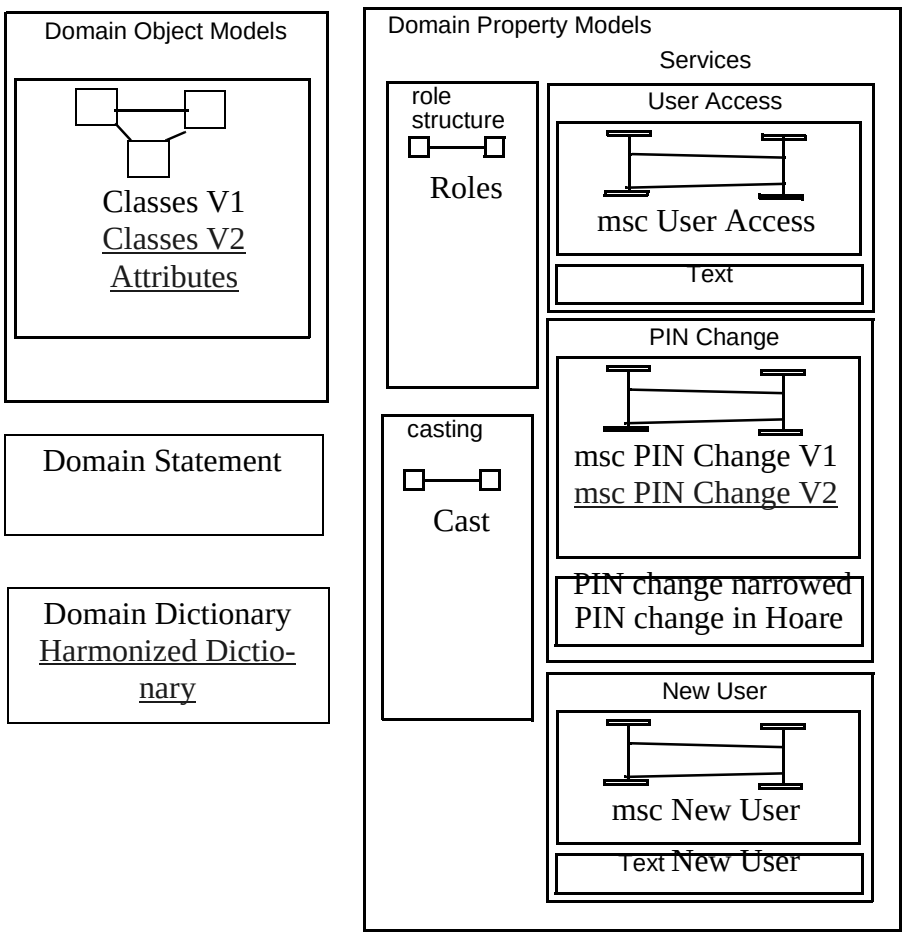
PINChanging role). Here we show that the new PIN is actually given. We have made the service more detailed by requiring that the given new PIN should also be validated. We give no indication of what the validation should include.

Summarizing domain descriptions

In Figure 9-17 "Domain Descriptions of Access Control" (p.9-17) we give an overview of the domain descriptions which the interactive reader can use for his exploration of the description. There are more descriptions connected through the map than given here in the textual file itself.

Figure 9-17: Domain Descriptions of Access Control

Open figure



Analysing Requirements

We have not identified the Access Control system in the Figure 9-15 "Domain model of Access Control V2" (p.9-15). It says a lot about the Domain, but nothing directly about the system. Indirectly, however, it tells us what kind of entities and relationships the system should handle.

The System and its Application Context

We make System contexts and object models in order to evaluate different solutions, while properties are handled in the Application Specification (p.9-21) activity.

Introducing the system implies that we have to decide on the border of the system: what is part of the system and what is part of the environment. Do we know all the *real* users of the system. In this case this is rather simple. AccessPoints are parts of the system, while AccessZones and Users are part of the environment. We have already specified that the User from the domain model represents all possible users, also them without any PIN, but we have not specified what happens if such a user tries to access an access zone. The Access Zones are in the environment and the system performs Entry Control and Exit Control for the Access Zones. (Each Access Point will be served by a Local Station in the system.)

We have to decide on the main technology to be used. This includes a decision on which kind of “keys” the users should use. Ordinary keys are abandoned, fingerprints is not mature technology, “køfri”-technology is considered to be too expensive, so we end up with plastic cards. We also constrain ourselves to make access system, where the access zones are rooms entered through doors.

Making system family statement (see also activity***)***

We document the decisions by refining the existing Domain Statement with system specific elements, see Figure 9-18 "Problem Statement, with system specific elements" (p.9-19).This is based upon the domain specific Domain Statement, but includes the fact that a system is introduced. The system specific concepts are emphasized, that is *operator*, *card*, *door*. In the domain analysis we just specified that access points may be blocked and that they report their status - now we introduce the Operator as the special user of the system that takes care of part of the blocking and get the status. It may still be so that some kind of blocking is done automatically and that the status also may be interesting for other persons than the Operator.

Figure 9-18: Problem Statement, with system specific elements

Open figure

Relation to domain

The main purpose of the access control system is to control the access of users to access zones. Only a user with known identity and correct access right shall be allowed to enter into an access zone. Other users shall be denied access.

Services

- *New User:* The *Operator* shall be able to enter new users.
- *User Access*: The authentication of a user shall be established by means of a magnetic strip *card* holding a card code and a secret personal identification number, PIN, entered by the user. The authorisation is performed by the system on the basis of the user identity and access rights associated with the user.
- *Change PIN*: It should be possible by the user to change his PIN.

Interfaces and environment

When a user is authenticated and authorised the access zone may be entered through a *door*. The environment not controlled by the system is considered as a special zone which every user may enter. Therefore, a *door* is seen as a connection between two access zones. Some *doors* may only be passed in one direction while other *doors* may be passed in both directions.

It should be taken into consideration that one access point can control the access to a set of access zones and that acccess to one access zone can be controlled by a set of access points.

Introducing a system also raises a few questions. Examples are:

- What will the physical interfaces be and what are the protocols to be used?
- What are the design constraints, e.g. requirements to fault tolerance, security, modifiability?
- Should the system keep track of where the Users are, i.e. know in which Access Zone each User is at any time? We decide to answer no!
- Should the system count the Users that pass each Access Point? If yes, should we ensure that the User really passes through? The answer is no!
- Will there be different kinds of Access Points? It is reasonable to believe that bidirectional doors are different from unidirectional doors and that the authentication and authorisation requirements will depend on the direction. The answer is yes.

- How should the access rights be represented? By giving each Access Zone an access level and each User an access capability? Or by explicitly listing which Access Zones are open to each User? We decide to use the latter approach!

System family dictionary (see also activity***)***

The dictionary is updated to become a dictionary, where the system specific entities are included, that is all concepts that are introduced with the introduction of the system are defined in the dictionary, see Figure 9-19 "Dictionary, with system specific concepts included" (p.9-20)

Figure 9-19: Dictionary, with system specific concepts included

Open figure

Access Granting	The role of granting (or not granting) a user access.
Access point	A point of access into an access zone.
Access zone	A physical or logical zone guarded by a set of access points.
Authentication	To establish the identity of a user.
Authorisation	To establish the right of a user to enter an access zone.
Authorizer	The entitity which determines authentication andauthorisation.
Authorizer	Also: the role of storing PINs
Card	A personal identification means. Typically a plastic card.
Card id	A unique identification of a card stored in machine-readable form on the card. We distinguish between user and supervisor cards.
Door	A controlled passage from one access zone to another.
Operator	A person with known identity and authorisation to change the status of the system. (See also supervisor)
PIN Changing	The role of changing the PIN of a user
PIN	A personal identification number. This number should be kept secret by the user and typed in on enering the access zone.
Supervisor	A person who controls the authorizer
User	A person with known identity with authorisation to enter specific access zones.

Non-functional Requirements

In addition to the functional requirements laid down in the service descriptions, there are general requirements to the performance of the system which cannot easily be described in languages like SDL and MSC. We call them non-functional requirements and Figure 9-20 "Non-functional requirements of AC system" (p.9-21) shows the non-functional requirements for the access control system we want to make.

Figure 9-20: Non-functional requirements of AC system

Open figure

Modifiability. The system shall be modifiable to accommodate other services than to open doors. One possible service will be an automatic teller (minibank).

Size. The system shall be flexible with respect to the number of access zones and access points. It shall be able to serve from one to 100 zones each having from one to 100 access points. The total number of access points in a system is limited to 1000 and the total number of users to 10 000.

Processing capacity. The system shall be able to serve six users a minute at each Access Point up to a total continuous peak load of 600 users a minute. Higher input rates shall not lead to loss or corruption of data, only to longer delays.

Error handling. A single error shall not affect the (normal) operation of more than 10 access points.

Security. The authentication and authorisation information shall be secured against unintended access.

Application Specification

(See also activity).

When to leave the general and abstract domain and property models and when to dig into the details of the system, is very much dependent upon the possibilities and resources of the project and the company. A general statement is that projects start digging into details far too early. On the other hand the project should not wander around in descriptions which are not adequate for the questions which are at hand. There is no reason to keep using informal UML diagrams when SDL is called for.

The application specification consists of five parts:

1. Make a context diagram (p.9-22)
2. Specifying the domain given objects (p.9-23)
3. Specifying the system given objects (p.9-28)
4. Specifying the interface given objects (p.9-28)

5. Specifying the services (p.9-29)

Make a context diagram

When the system and its environment has been identified we classify the entities in the environment in two main categories:

- The entities stemming from the Domain analysis: In this case: Users, Access Zones
- The entities that are introduced in connection with the introduction of the system: In this case: Doors, Cards and Operators.

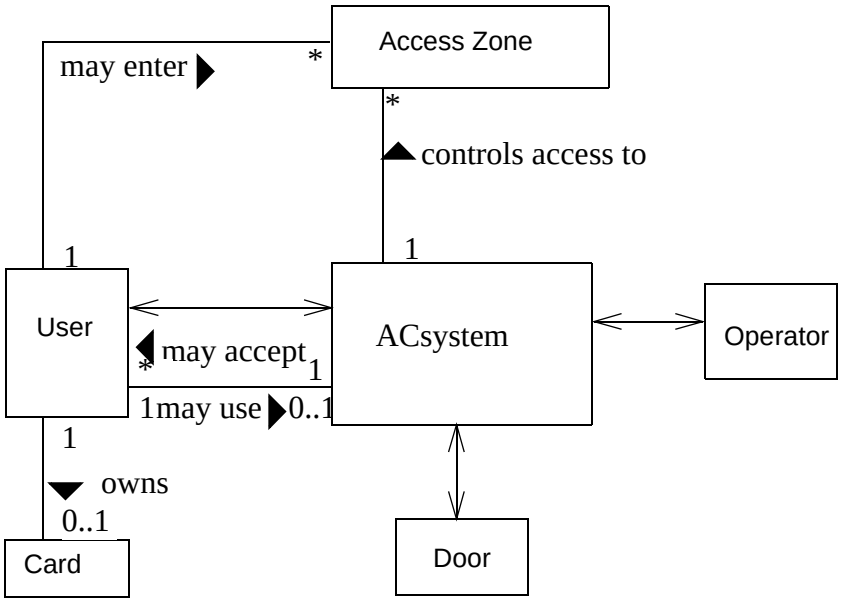
For each of these we ask ourselves if they are *users* of the system, if they are processes that are *controlled* by the system, if there are *other systems*, or if they are just *known entities*.:

- Known Entities: Cards, AccessZones
- Other Systems: None
- Controlled Processes: The actual doors and panels are controlled processes, and the corresponding parts of the system are controlling these.
- Users: Users and Operators are different kinds of users: they both use the actual system. Users are Domain Specific, while Operators are System Specific Users. There are no entities in the Environment that are just influenced by the system without being in direct contact with it, unless we include the owners of the Access Zones that may be paid by users entering a zone.
- The notion of User includes all possible persons that may try to enter some Access Zone, either with a valid card, but a bad PIN code, or with a non-valid card, and even persons with no card at all. This is important when the robustness of the system shall be designed - it will not just be well-behaved users that will try to use the system. This is in the Object Model represented by User having one or Zero cards.

The system context is depicted in Figure 9-21 "The access system context" (p.9-23). Here we see the system itself and the system environment. Not everything in the environment is shown, only the parts that are related to the system.

Figure 9-21: The access system context

Open figure



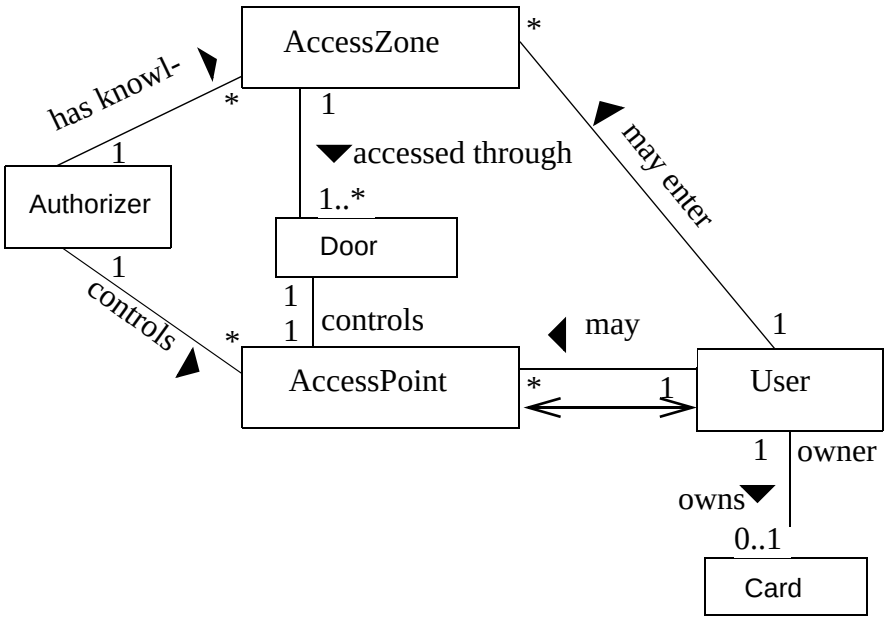
Specifying the domain given objects

When the environment of the system and the main technological solution has been chosen, we do the same as for Domain Analysis, that is refining the object model taking into account objects and classes coming from the fact that the system is introduced.

We make a new Object Model that incorporates the Doors and the Cards and their relations to the already identified types. This is described in Figure 9-22 "The access control domain, system specific" (p.9-24).

Figure 9-22: The access control domain, system specific

Open figure



From the diagram we can see that a User may enter several Access Zones and that each Access Zone may accept several Users.

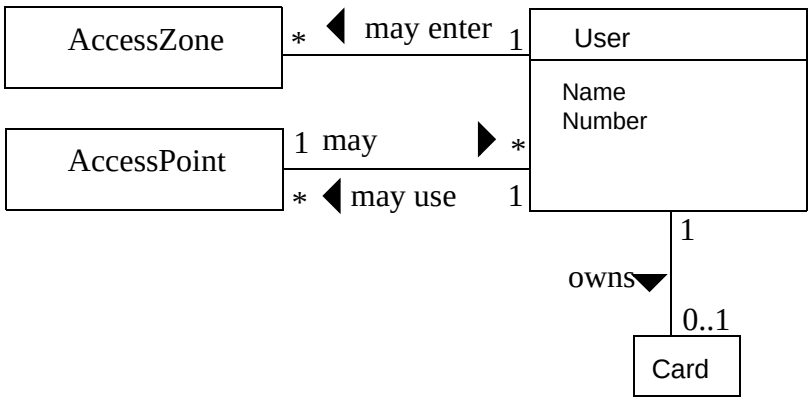
Analyse each class

In order to understand each concept type better and describe precisely the constraints on related entities, we make explicit *class definitions with constraints on their environment.*. For each of the class we also consider the property models involving the class and determine if the property models have implications for relations, connections and attributes associated with the class.

The concept of User and its relation to the environment is described in Figure 9-23 "The class definition of User" (p.9-25).

Figure 9-23: The class definition of User

Open figure



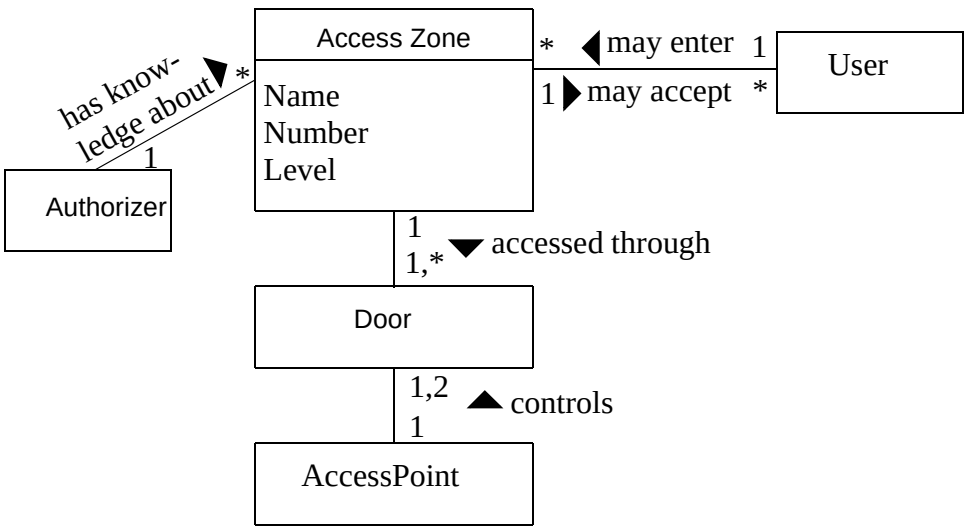
The attributes of the User are defined to be a name and a number. In our Domain each User shall own a Card and have the right to enter at least one Access Zone. From the type environment it is clear that an instance of User:

- shall own exactly one Card, (Users with no card cannot be known to the system, but having a card is not the same as to be allowed access)
- may enter one or more Access Zones,
- may use zero or one Access Point (at one time).

The concept of Access Zone and its environment is described in Figure 9-24 "The class definition of Access Zone" (p.9-25)

Figure 9-24: The class definition of Access Zone

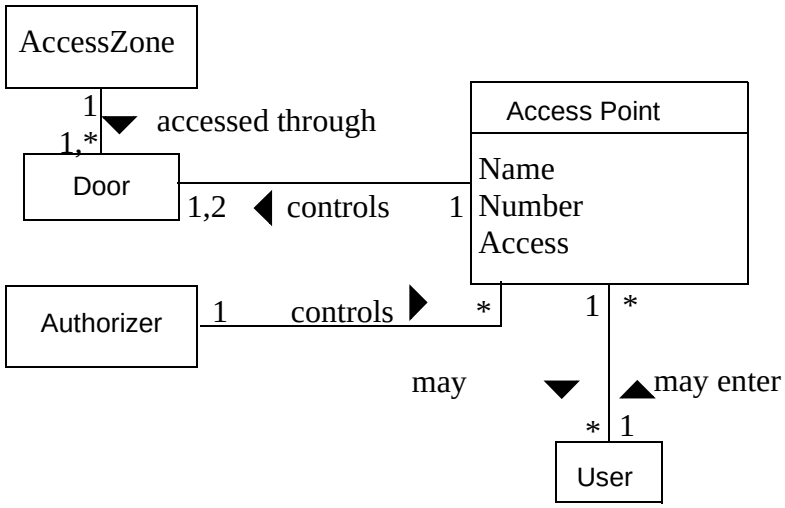
Open figure



The concept of Access Point and its environment is described in Figure 9-25 "The class Access Point with environment" (p.9-26)

Figure 9-25: The class Access Point with environment

Open figure



Generalisation/specialisation hierarchies

Will all the Access Points be similar? No, the requirements for authentication and authorisation will vary depending on the relative access restrictions on the Entry Zone compared to the Exit Zone. In some cases, no authentication and authorisation is needed at all. It is sufficient that the User operates a simple push key to open the door. In other cases the User must enter the card and, in addition, enter a PIN. Finally, there will be access points where the PIN is not required, only the card.

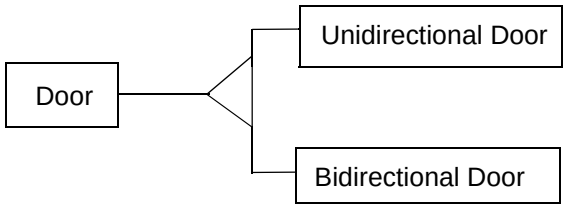
Two Access Points may control a single Door in the case when the Door is bidirectional. Will the difference between Doors have any consequences for the Access Points? We do not know that yet, but it is reasonable to believe that some extra coordination will be needed when two Access Points control the same Door.

From this we gather that there will be different types of Doors and Access Points. Thus, to complete our object models we should define all the subclasses.

This results in the specialisation hierarchies in Figure 9-26 "Classification of Doors" (p.9-26) for Doors,

Figure 9-26: Classification of Doors

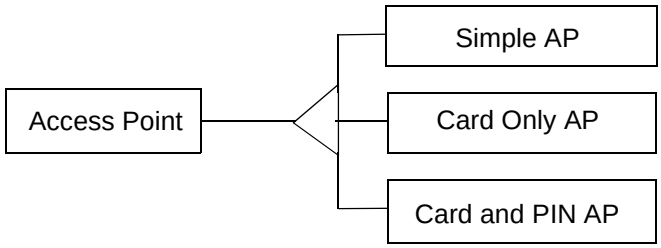
Open figure



and in Figure 9-27 "Classification of Access Points" (p.9-27) for AccessPoints.

Figure 9-27: Classification of Access Points

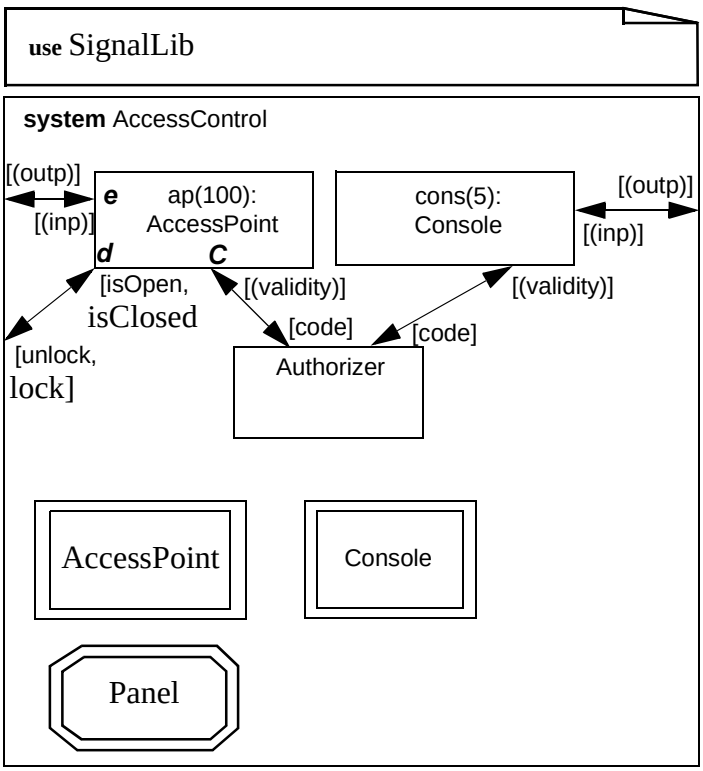
Open figure



In Figure 9-28 "AccessControl System V1" (p.9-27) we define the Access Control system containing objects to match the domain specific concepts AccessPoint, Authorizer and Console

Figure 9-28: AccessControl System V1

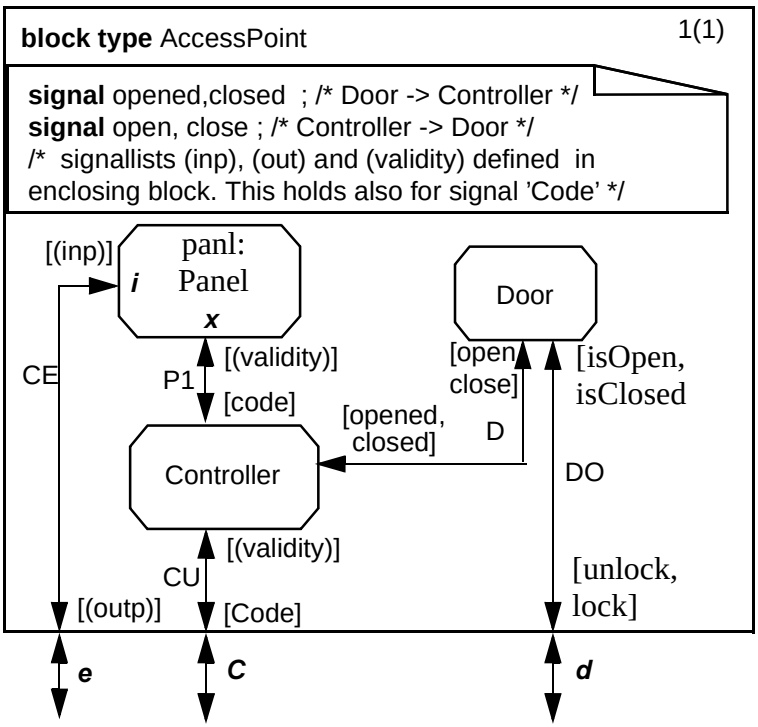
Open figure



We may also continue to peek into the structure of the Access Point which is shown in Figure 9-29 "AccessPoint V1" (p.9-28) .

Figure 9-29: AccessPoint V1

Open figure



Specifying the system given objects

From Figure 9-29 "AccessPoint V1" (p.9-28) we see that a system specific concept *Panel* has emerged. It represents the interface between the user and the access point controller. The access point Controller is also a new concept. It represent some calculating capacity assumed part of the access point.

We do not want to go into the details of these processes at this stage.

Specifying the interface given objects

We have already in Specifying the system given objects (p.9-28) introduced the interface object *Panel* which is the low level interface between the user and the access control system. From the context diagram in Figure 9-21 "The access system context" (p.9-23) we may perform the general “mirroring” and achieve the first version of the access control system structure in SDL

In Figure 9-28 "AccessControl System V1" (p.9-27) we see that the supervisor (operator) is mirrored by a *Console* and the *User* mirrored by the *AccessPoint*. Furthermore we assume that the Authorizer concept will find its counterpart inside the system.

We assume a *Panel* as a low level interface both in the *Console* and in the *AccessPoint*. This is why the definition of *Panel* appears external to the *AccessPoint*.

Specifying the services

Having worked with the object models for a while developing our conceptual understanding of the structure of the access control systems, we return to property modelling before we have reached the full definition of the object model. We want to use the property descriptions of the services to reach the behavior descriptions in the object model.

The following services have been described in the domain model and now we shall refine them in the context of an access control system.

- Domain model: User Access (p.9-8)
- Domain model: PIN changing (p.9-9)
- Domain model: New User (p.9-10)

These models are refined in

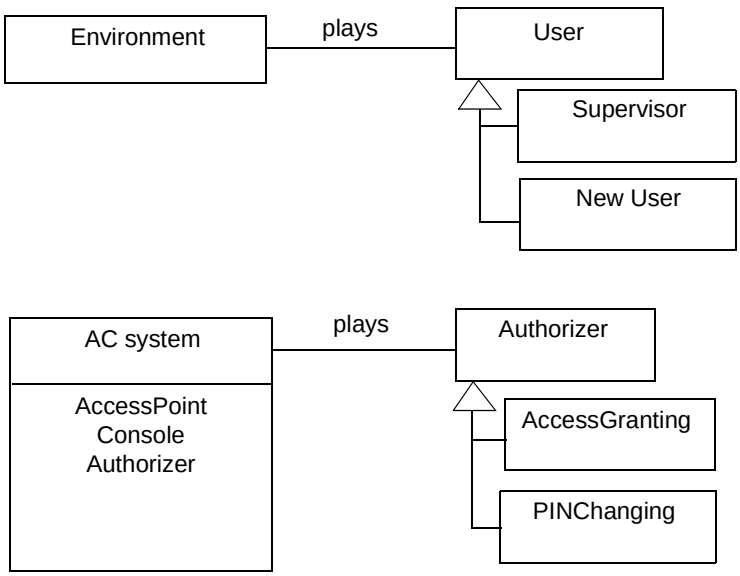
- Application model: User Access (p.9-30)
- Application model: PIN Change (p.9-31)
- Application model: New User (p.9-32)

and they all use MSCs which we classify as Auxiliary MSCs (p.9-32).

Having worked on the object model, in order to associate the roles of the domain property model with the objects of the object model, we need to perform casting.

Figure 9-30: System specific casting

Open figure

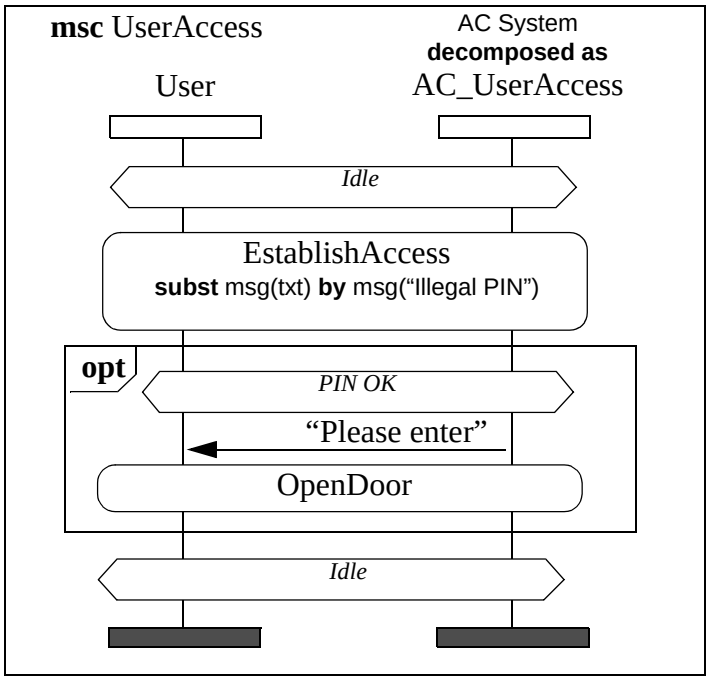


The casting gives little insight into the actions of the new objects which are contained inside the AC system, but it defines the casting (play) relation. We may now define property MSC diagrams of the services where the AC system becomes one instance. This description can be called a (“black box”) specification. The corresponding domain service model is used for inspiration.

User Access

Figure 9-31: UserAccess V1

Open figure



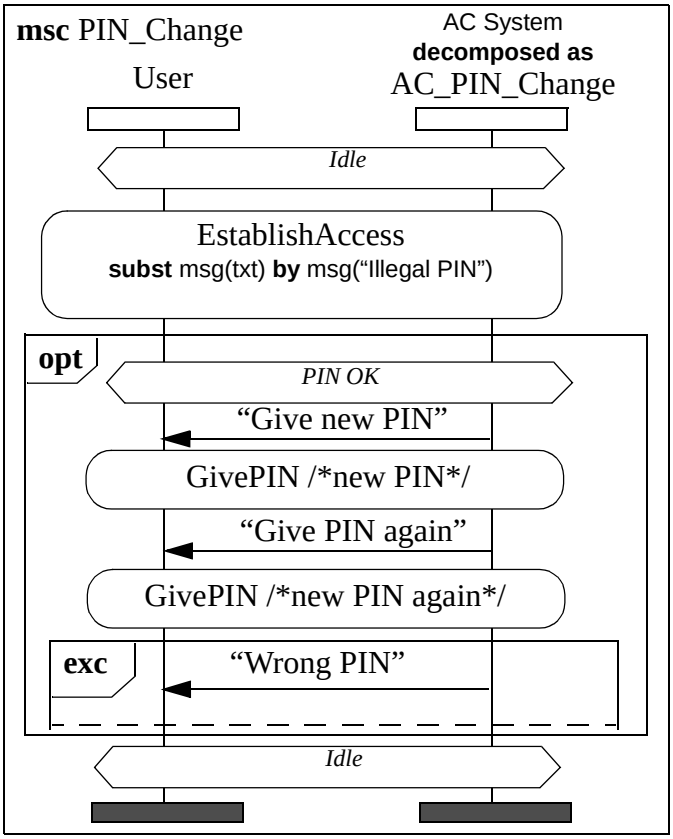
The MSC-96 diagram in Figure 9-31 "UserAccess V1" (p.9-30) represents the service interaction overview. The details are hidden inside the referenced MSCs EstablishAccess and OpenDoor which are found in Figure 9-34 "EstablishAccess V1" (p.9-33) and Figure 9-35 "OpenDoor" (p.9-34).

We also notice that the instance *AC System* is decomposed. When the decomposition is followed, we enter a more detailed specification of the services which is the subject of the more detailed application design activity.

PIN Change

Figure 9-32: PIN_Change V1

Open figure

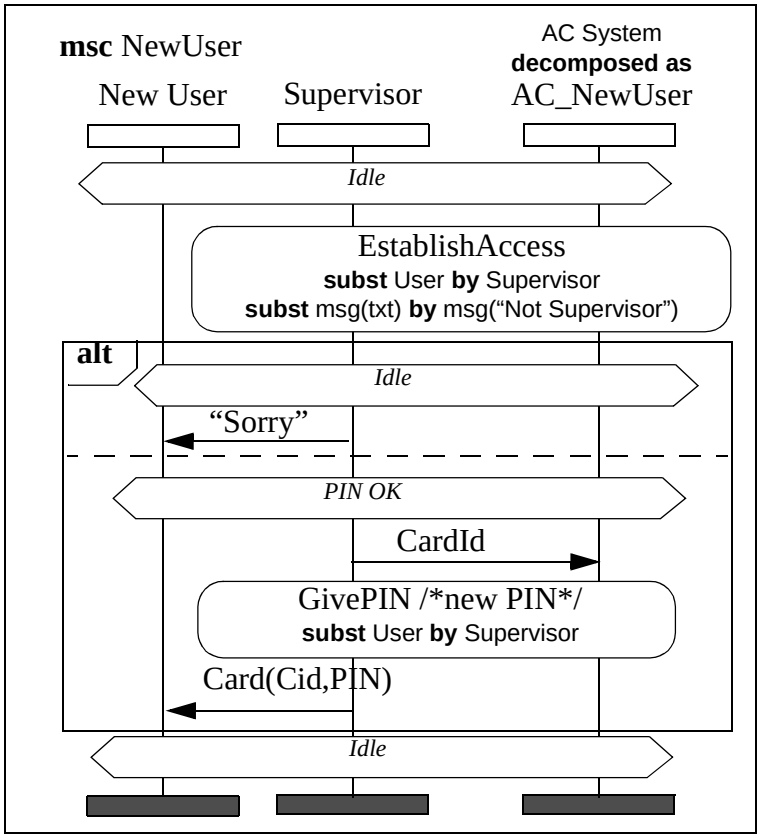


The PIN Change service shown in Figure 9-32 "PIN_Change V1" (p.9-31) also include references to auxiliary MSCs found in Figure 9-34 "EstablishAccess V1" (p.9-33) and Figure 9-36 "GivePIN" (p.9-34).

New User

Figure 9-33: NewUser V1

Open figure



The service New User shown in Figure 9-33 "NewUser V1" (p.9-32) utilize the MSC substitution mechanism to modify the EstablishAccess MSC referenced such that the User is replaced by the Supervisor. The auxiliary MSCs are found in Figure 9-34 "EstablishAccess V1" (p.9-33) and Figure 9-36 "GivePIN" (p.9-34).

Auxiliary MSCs

Since certain behavior patterns (interaction patterns) are common among the services, we present these common patterns in a section by itself.

Figure 9-34: EstablishAccess V1

Open figure

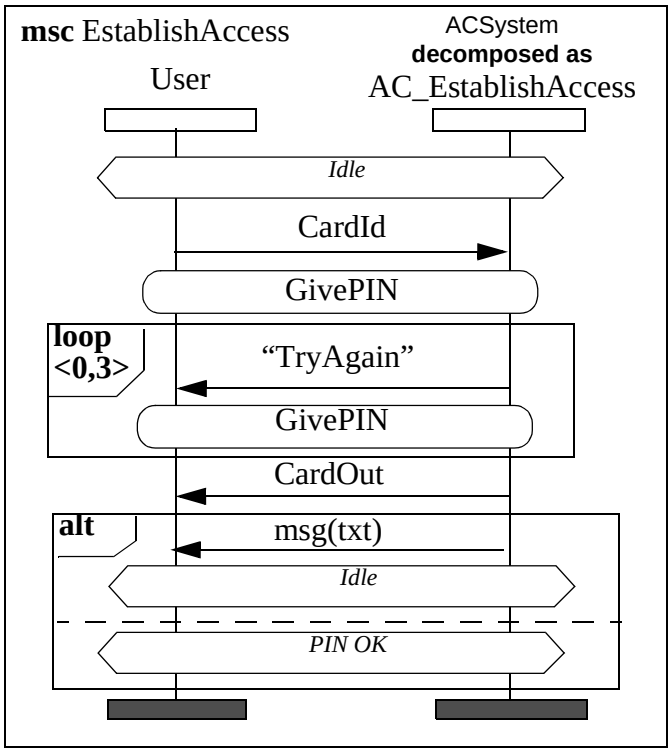


Figure 9-35: OpenDoor

Open figure

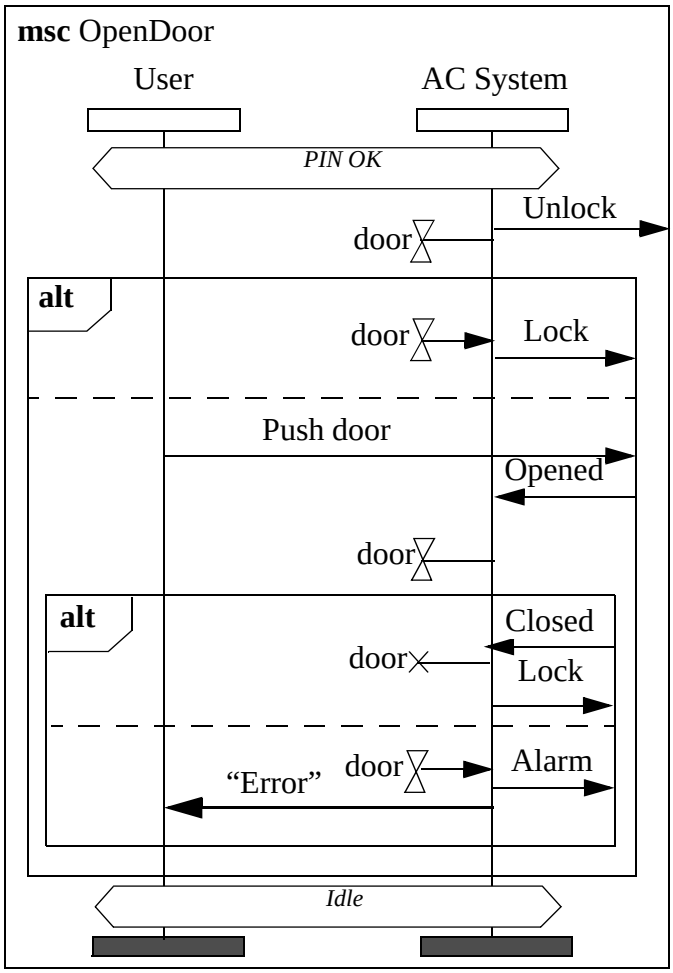
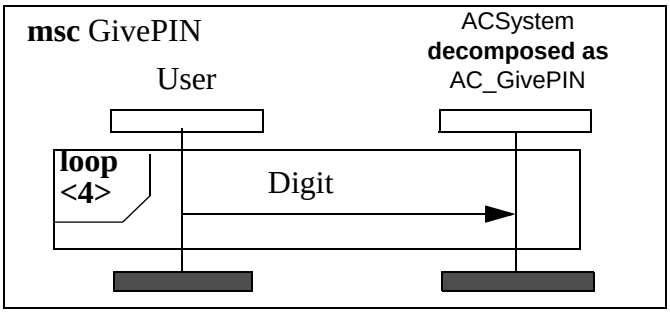


Figure 9-36: GivePIN

Open figure



Application design

(See also activity).

We have in Application Specification (p.9-21) described the top levels of the object structure and the services by “use cases” where the system as such appears as one instance. This is basically a “black box” specification. In order to approach an implementation further and to be more aware of the problems and potentials of our system, we should make a more detailed specification.

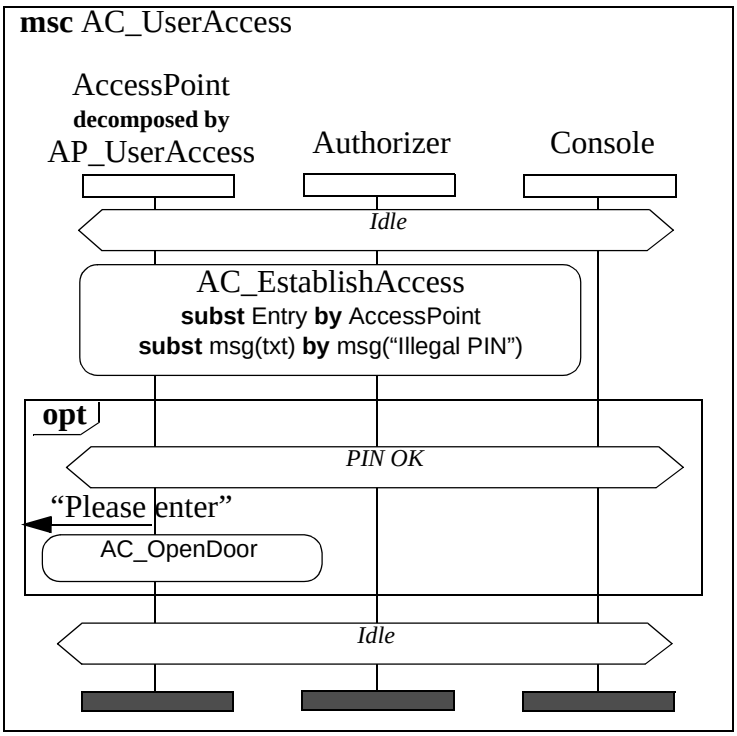
Property orientation and combined approach

We start by working on the property model, but we will supplement the property oriented approach with an object oriented one such that we perform a mixed approach to system description and the constructive use of MSC.

We decompose the AC system instances of our specification MSCs according to the structure found in Figure 9-28 "AccessControl System V1" (p.9-27). If we take User Access as an example this results in Figure 9-37 "AC_UserAccess V1" (p.9-35).

Figure 9-37: AC_UserAccess V1

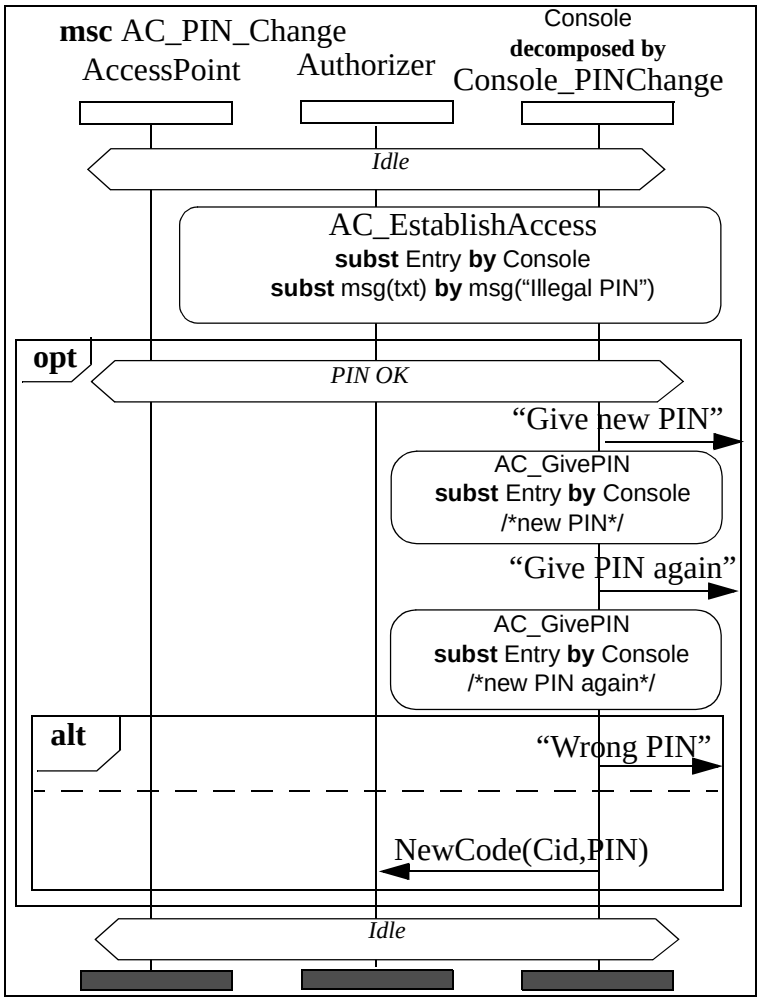
Open figure



For PIN Change the decomposition results in Figure 9-38 "AC_PIN_Change V1" (p.9-36).

Figure 9-38: AC_PIN_Change V1

Open figure



The interface consistency of the decomposition is easily established since the decomposition diagram follows the exact same structure as the diagram where the decomposed instance is located.

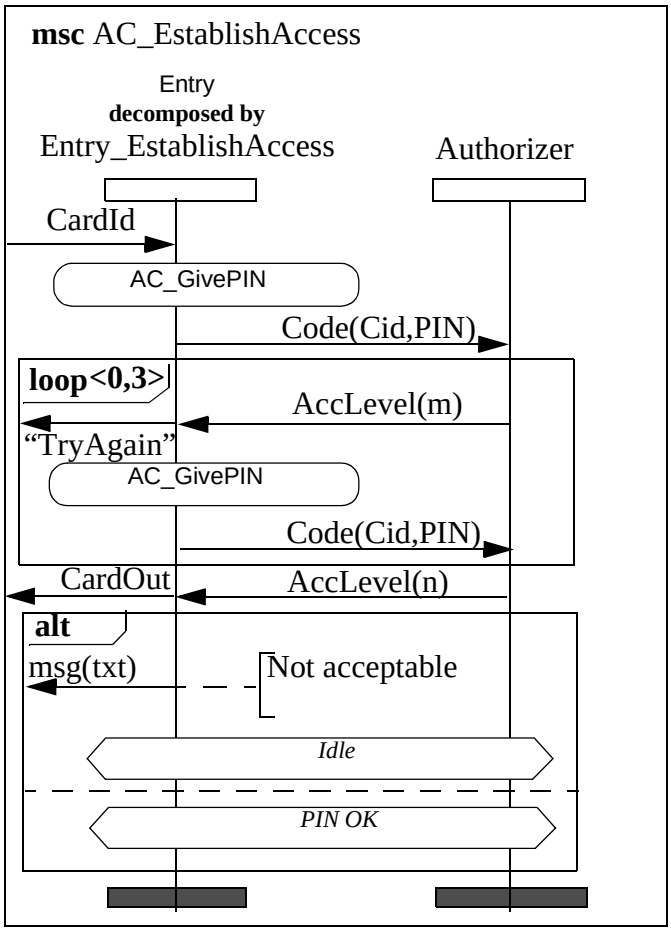
Discovering Entry

In both User Access and PIN Change (and also New User) we have applied the auxiliary MSC *EstablishAccess*. In User Access the access is establish through an Access Point while in PIN Change and New User the access is done through a console. Since we want to keep the structures simple and keep the orthogonality between instance decomposition and referencing auxiliary MSCs (see in the MSC tutorial and confer the Figure of this orthogonality), we need to describe the lower level *AC_EstablishAccess* in way which generalises (or parameterizes) such that the *AccessPoint* and the *Console* can be seen as similar entities.

We conclude that in *AC_EstablishAccess* we wanted to introduce an instance called *Entry* which could be filled by both the *AccessPoint* and the *Console*. *AC_EstablishAccess* is shown in Figure 9-39 "AC_EstablishAccess V1" (p.9-37).

Figure 9-39: AC_EstablishAccess V1

Open figure



Already at this point we halt and consider whether our new insight from the property model should be carried over to (harmonized with) the other descriptions in particular the SDL structure model.

Harmonizing with the object model

We have discovered a concept Entry which is not property reflected in the object model. We said above that both AccessPoint and Console could be Entry. Said in object oriented words this could mean that there could be a concept Entry which AccessPoint and Console were subtypes of.

We decide to work on this idea and discover that this is in fact fruitful.

Figure 9-40: AccessControl System V2

Open figure

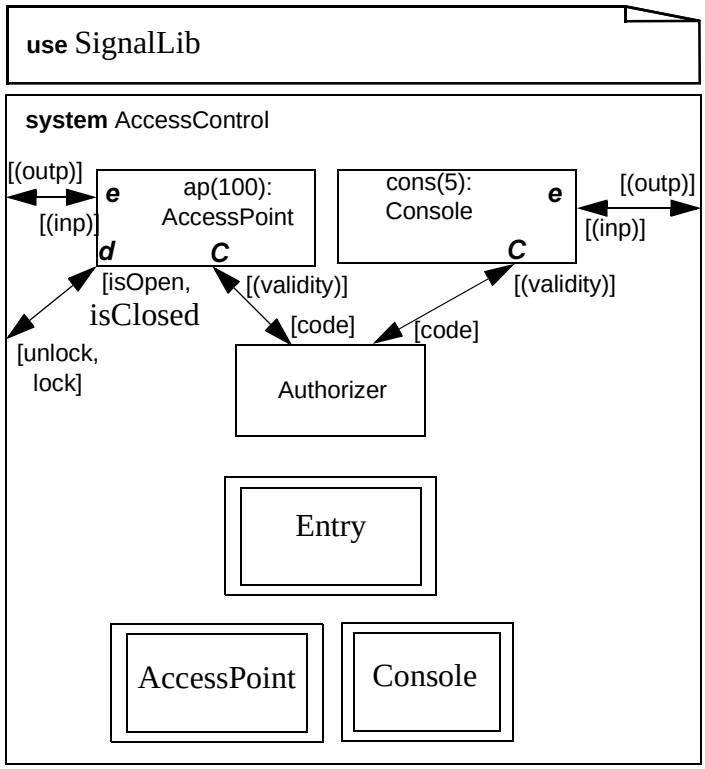
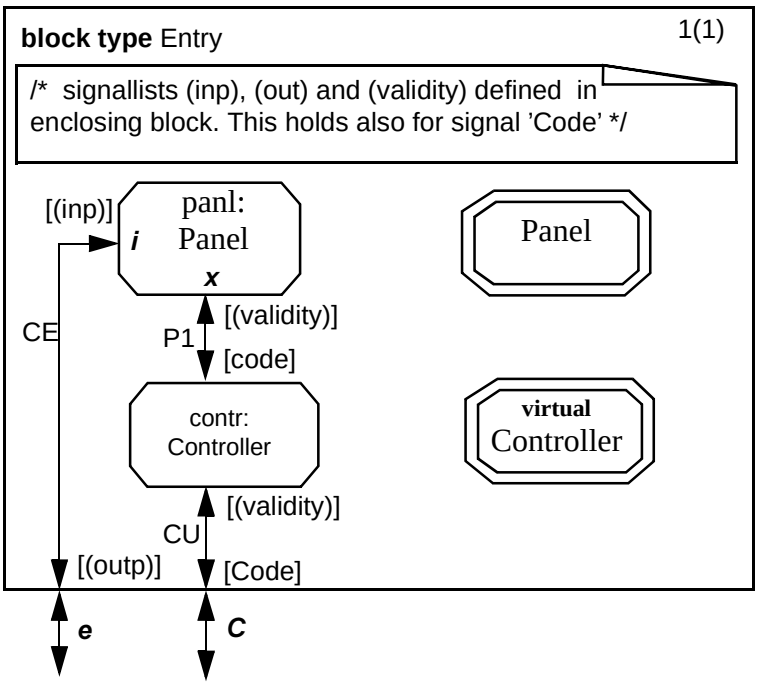


Figure 9-40 "AccessControl System V2" (p.9-38) shows the modified system structure. We notice that the low level interface Panel now has been contained in Entry as shown in Figure 9-41 "Entry" (p.9-39).

Figure 9-41: Entry

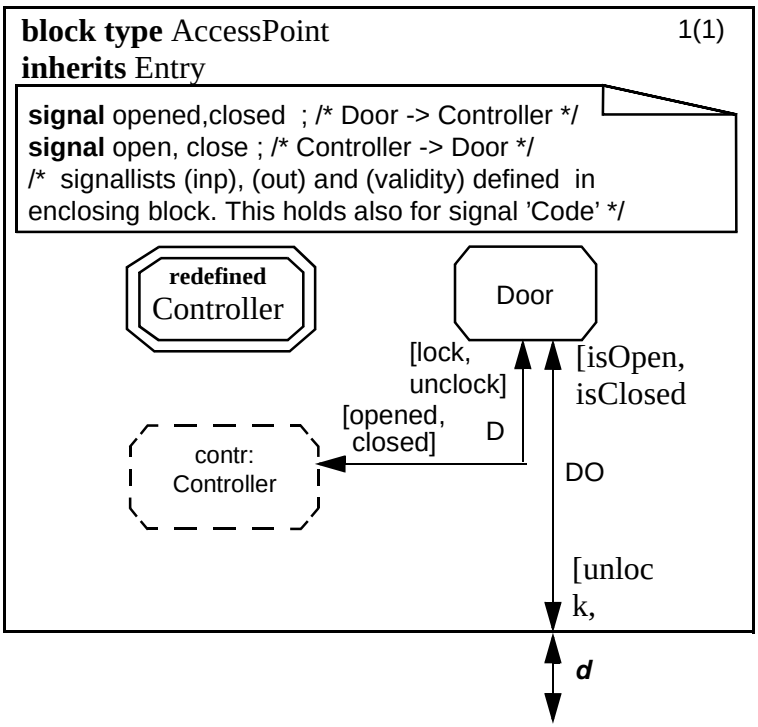
Open figure



Having introduced Entry, the AccessPoint becomes slightly simpler as shown in Figure 9-42 "AccessPoint V2" (p.9-40).

Figure 9-42: AccessPoint V2

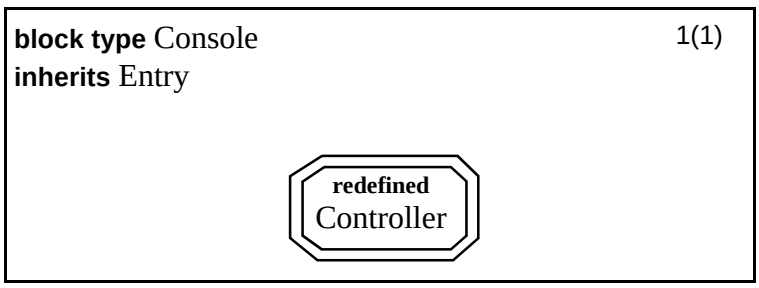
Open figure



The Console becomes almost empty as shown in Figure 9-43 "Console" (p.9-40).

Figure 9-43: Console

Open figure



The low-level unintelligent Panel

A Panel is contained in both the AccessPoint and the Console. From the idea that the Panel is the ultimate interface given entity and the Controller the logical “brain” of the Entry, we postulate that the Panel should have no specific knowledge of the services as such and that the same Panel should be used in both AccessPoint and Console.

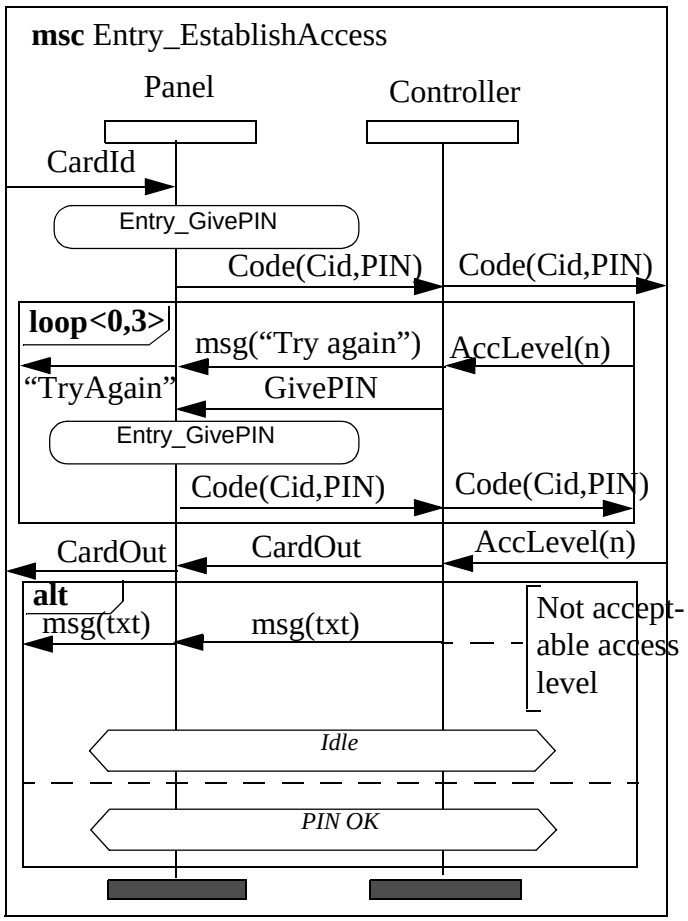
Formally this is shown in Figure 9-41 "Entry" (p.9-39) by the fact that *Panel* is not virtual while *Controller is* virtual.

Approaching more detail through decomposition

To specify our “dumb” Panel we continue our decompositions. Now Entry should be decomposed leaving the Panel interaction explicit. We show in Figure 9-44 "Entry_EstablishAccess V1" (p.9-41) the decomposition of Entry_EstablishAccess.

Figure 9-44: Entry_EstablishAccess V1

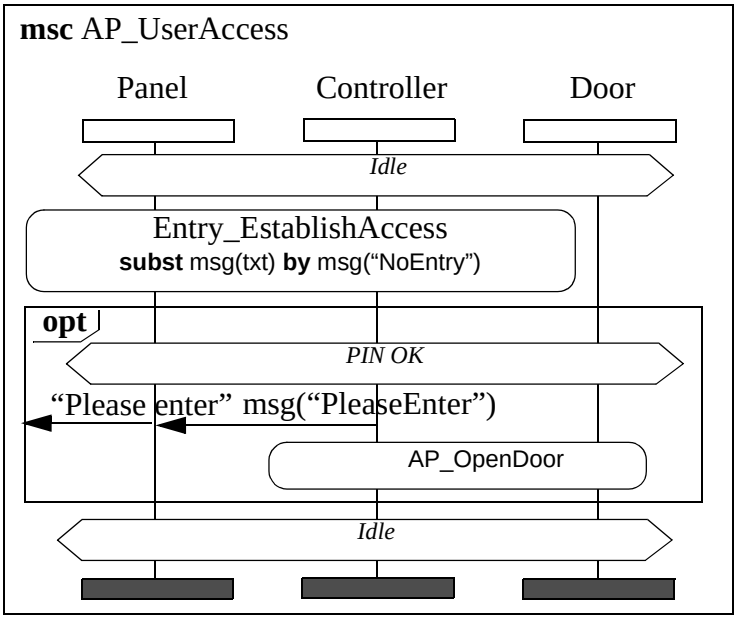
Open figure



The decompositions of Entry are also carried out from every service. Again we make sure that the orthogonality principle between decomposition and MSC references are kept as Entry_EstablishAccess is referenced from AP_UserAccess the decomposition of AccessPoint in UserAccess shown in Figure 9-45 "AP_UserAccess V1" (p.9-42)

Figure 9-45: AP_UserAccess V1

Open figure



Having done all the decompositions of Entry (including those of AccessPoint and Console) we have reached a good specification of the Panel interaction. We could expect to be able to produce Panel automatically from the MSCs through the MSC-to-SDL skeletons.

Producing SDL skeleton from MSCs for Panel?

When trying to apply the skeleton technique, we discover that it is not well suited here. There are no local conditions related to Panel, and using the global conditions does not do the trick since they are actually states of the service as such and not the local behavior of the unintelligent Panel. The attempt is not reported here, but the attempt ends in a need to unify a fairly large set of states. This may be possible, but it is not easily done automatically, and it seems unnecessary complicated for a simple process like Panel.

We decide to continue on the SDL track and specify Panel in SDL inspired by the MSCs.

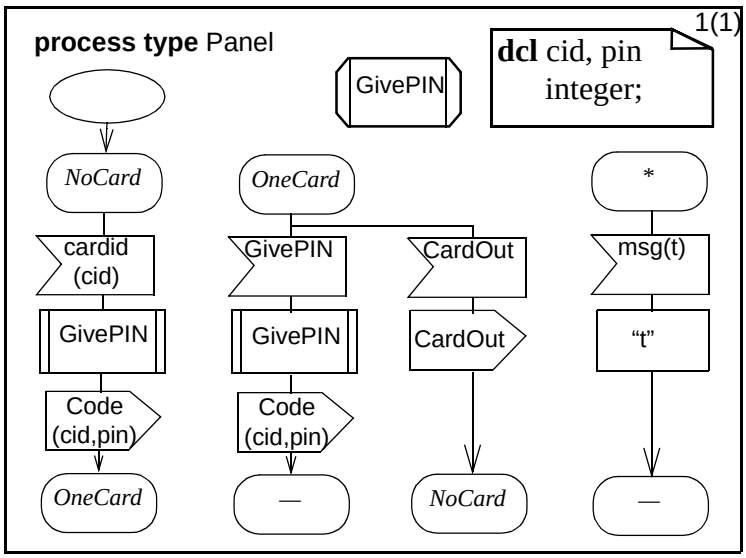
Specifying Panel in SDL

The Panel is "dumb" as it reacts blindly to the signals received from the controller and the inputs from the user. It could suffice to have only one state and then a number of transitions from that state. We believe this would work, but good engineering practice recommend otherwise.

Our strategies for modelling behavior recommend that the state space should be found from what the user will identify as control states from the outside. In the case of the Panel it is reasonably obvious that the user will distinguish between the situation where there is a card in the Panel card reader and the situation where there is no card in the Panel.

Figure 9-46: Panel

Open figure



By distinguishing between these two states we also achieve more robustness in our description. We can now not only define situations which are correct, but also situations which represent errors. Such situations occur when we get unexpected signals in a state.

Figure 9-46 "Panel" (p.9-43) shows the definition. The reader should notice that we have introduced a procedure for the user inputting a PIN. We have considered this trivial and have not described that here. It is clear that the entering of the PIN could be made more elaborate by introducing timers and properly handling partial PINs. This adds little to our story.

The reader should also bear in mind that we have implicitly said that all default transitions should be considered harmful and erroneous.

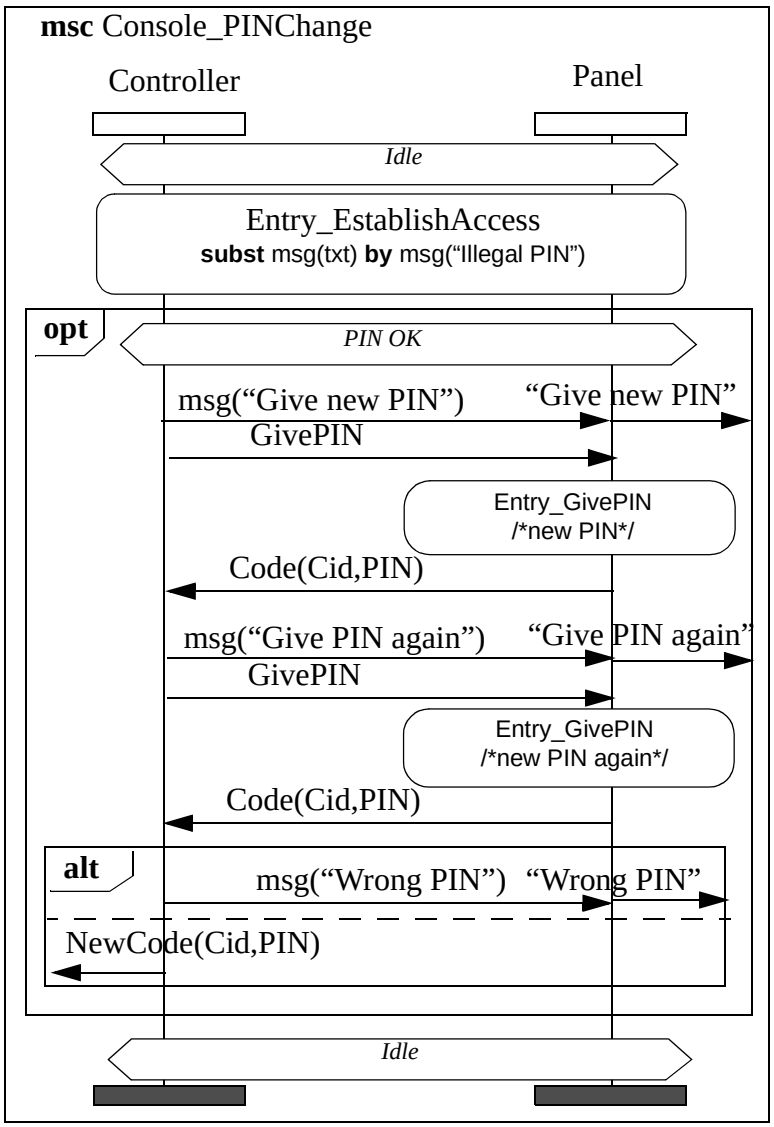
Checking the Panel by executing PIN Change

Having designed the Panel in SDL we are in a position to perform some model checking. We should try to find out whether the MSCs of the services projected down to the Panel can be fulfilled by our specification of Panel. We assume that all other components act according to the MSCs, but the Panel performs according to its SDL description.

Our example service is PIN Change found in Figure 9-47 "Console_PIN_Change V1" (p.9-44).

Figure 9-47: Console_PIN_Change V1

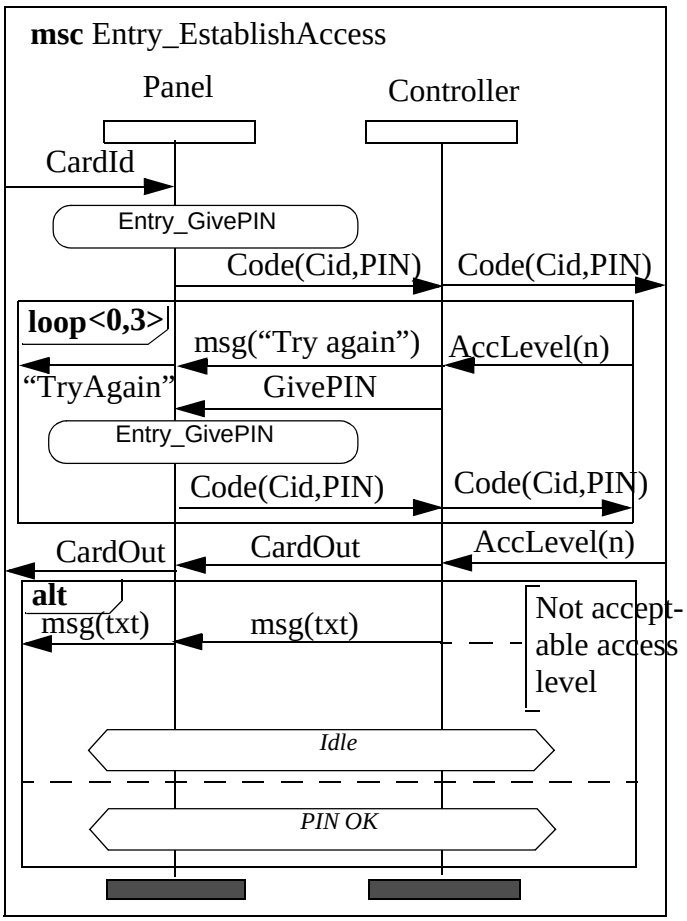
Open figure



Since this service starts by a reference to an auxiliary MSC found in Figure 9-48 "Entry_EstablishAccess V1" (p.9-45), we should start by executing that.

Figure 9-48: Entry_EstablishAccess V1

Open figure



For our model checking to work, we must align the MSC and the SDL descriptions. Here we assume that condition *Idle* corresponds to the Panel being in *NoCard* state. Then our execution proceeds as follows:

1. Input of *Cardid* shown in MSC Figure 9-48 "Entry_EstablishAccess V1" (p.9-45). This is legal and results in the SDL Figure 9-46 "Panel" (p.9-43) executing *GivePIN* which is exactly matched by the MSC. The SDL then outputs *Code* signal which again is matched exactly by the MSC. Panel then enters state *OneCard*.
2. Assume entering the loop<0,3> of the MSC Figure 9-48 "Entry_EstablishAccess V1" (p.9-45). This means receiving *msg("Try again")* which in the SDL is simply forwarded to the User (environment) which matches the MSC completely. Panel is still in *OneCard* state.
3. Input of *GivePIN* signal (shown in MSC) is now the next event. This is legal in the SDL and results in another execution of the *GivePIN* procedure which is matched by the *Entry_GivePIN* MSC reference. This is again followed by the *Code* output which also matches the MSC. *Panel* is still in state *OneCard*. The situation at the end of the loop is very similar to the start of the loop and further iterations cannot upset the consistency between the MSC and the SDL descriptions.

4. Assume exiting the loop and continuing. *CardOut* signal is received (from MSC) and this is legal in the SDL. The SDL specifies forwarding the *CardOut* to the *User* meaning that the card is ejected. *Panel* now enters *NoCard* again.
5. The MSC now specifies an alternative. We have to perform them both. One alternative is simply that the execution enters the condition *PIN OK* while the other represent a situation where the user is not allowed to enter.
6. Assume alternative where the user is not allowed to enter. Then *msg(txt)* is received. From the substitution in the MSC reference of Figure 9-47 "Console_PIN_Change V1" (p.9-44) this actually means *msg("Illegal PIN"*). This is simply forwarded to the environment (User) and *Panel* remains in *NoCard*. This matches the MSC. The MSC specifies that the whole system is back to *Idle* which locally corresponds to *Panel* being in *NoCard* which is OK.
7. Now we leave the referenced MSC and return to the MSC of the service in Figure 9-47 "Console_PIN_Change V1" (p.9-44). There is an option where only the case where condition PIN OK holds initially will be considered. Formally the global conditions have no constructive semantics to define the legal continuations, but we choose to interpret the MSC this way. In this particular case this does not really matter. Thus we assume that we are in the situation where PIN OK, the MSC specifies the input of *msg("Give new PIN")*. This is forwarded to the user and the *Panel* stays in state *NoCard*. The consistency is still present.
8. Now the MSC specifies the input of *GivePIN*. Alas! *GivePIN* is illegal in *NoCard* state of *Panel*! (It is defined as a default transition which formally is legal, but we have decided to consider all default transitions harmful. Why we have them at all is just the lack of space in the diagrams in order to be able to show them in reasonable space in this textbook).

Our conclusion must now be that we have found that the service PIN Change cannot be performed consistently with the current definition of Panel.

We now face a problem. Either the services are inadequately specified in the MSC document, or the definition of Panel is wrong. As mentioned earlier, we could specify Panel by only using one state. In this case this would do the trick, but it may not be the best solution.

Our problem is that we wanted to give the new PIN when the user had received his card. Imagine that the new PIN should in some encrypted form be stored in the card. This is not specified now, but we can imagine such situation in the future or in some related system. Then the user has actually received his card too early. Another possible situation is that the user leaves after having received his card, but having neglected to finish the service. This has no consequences for the user, but only for the system which is left in a situation where it must be helped by some timer to exit from the PIN Change service. This is not specified here either. Holding the card back to the end of the service would at least give consequences for the user who leaves the premises without completing the service. He will be without his card (provided that the card reader actually keeps the card).

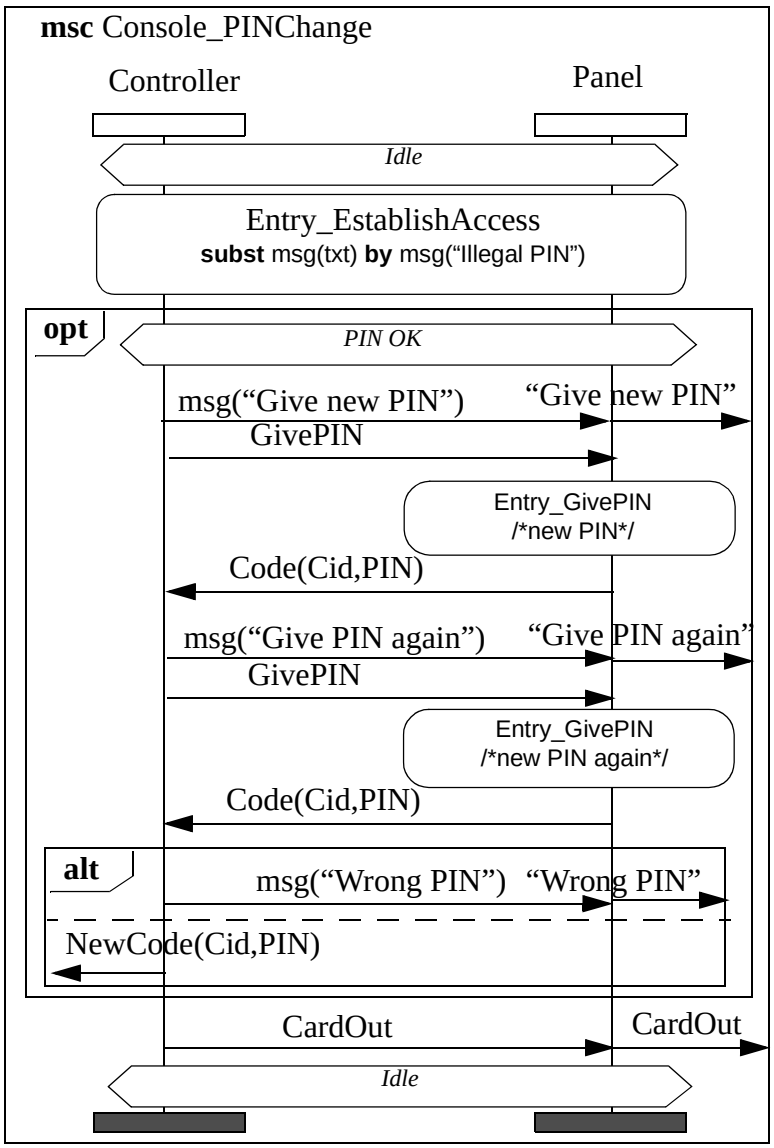
Our considerations result in a decision to redesign the services and keep the Panel definition as it was made.

Redesigning the services

The redesign of the services must imply that EstablishAccess cannot give the card back to the user. We simply delete the return of the card from EstablishAccess and insert it in the proper place in the three other services. All MSCs must be updated as a result of this change. As an example we show the PIN Change service which was the initiator of this change in Figure 9-49 "Console_PIN_Change V2" (p.9-47).

Figure 9-49: Console_PIN_Change V2

Open figure



Model checking this service results in complete consistency since the *Panel* will be in state *OneCard* all the way down to the final CardOut signal. In state *OneCard, GetPIN* is a valid input.

The service level Controller of Console

We will use a different strategy to specify Controller. Controller is a virtual process in Entry. The default Controller is only a start transition leading to the state Idle. The controller of Console and the controller of AccessPoint will be specializations of the Entry controller.

We have decided that to perform PIN Change and New User services, this can only take place at a console. We thus have these characteristics which distinguishes the services:

- User Access: takes place at an AccessPoint.
- PIN Change: takes place at a Console and the User is a normal user.
- New User: takes place at a Console and the performer is a supervisor.

To distinguish between the two latter services at the console we introduce the assumptions given in Figure 9-50 (p.9-48) of the returns from the Authorizer.

Figure 9-50: Assumptions for our solution

Open figure

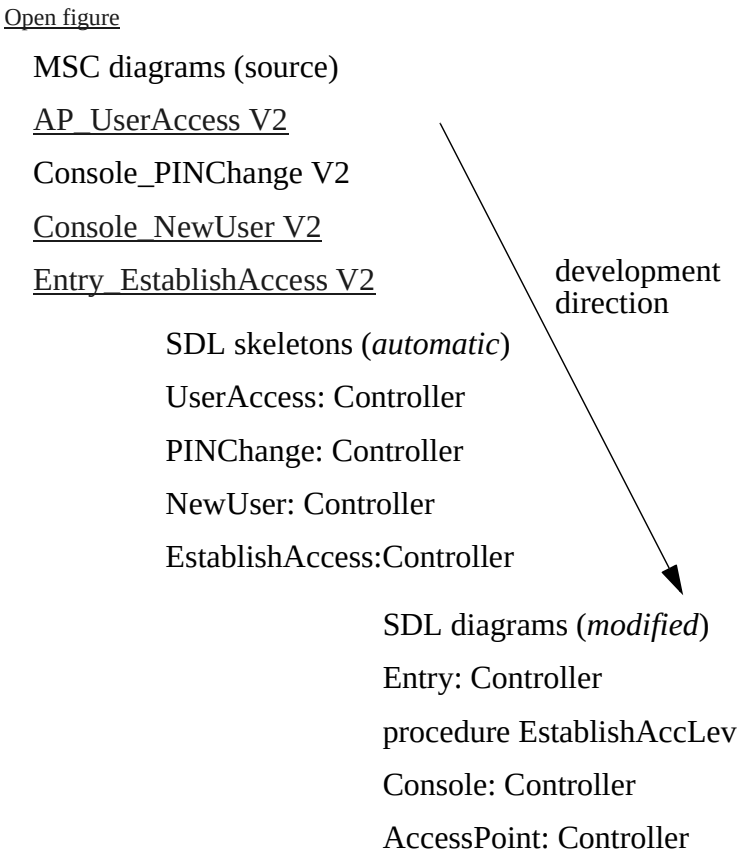
We assume that the authorizer is able to give a more advanced return than a mere OK and NOK. We assume that the authorizer knows the difference between the supervisors and the normal users.

Thus the return with AccLevel has the following possibilities for parameter value:

- -2: Supervisor with illegal PIN
- -1: Normal user with illegal PIN
- 0: Not a valid card
- 1: Normal user with legal PIN
- 2: Supervisor with corresponding PIN
- -99: Error occurred

The controller is the performer of the services. Therefore the global conditions of the MSCs corresponds more closely to the expected states in the controllers than in the Panel. We may therefore have more confidence in producing SDL from MSC.

Figure 9-51: Controller skeletons



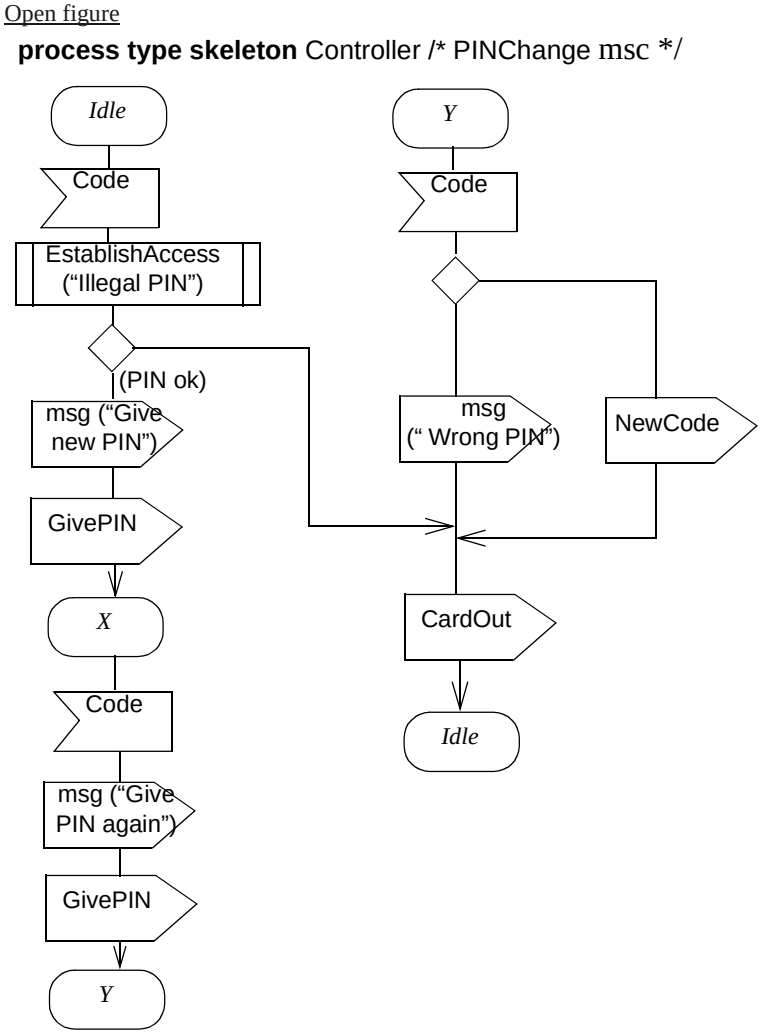
An overview of our technique is given in Figure 9-51 "Controller skeletons" (p.9-49). We shall concentrate on Controller of Console.

Service skeletons

Through using our recommended semi-automatic techniques for producing SDL skeletons from MSC, we reach the following result from the service PIN Change shown in Figure 9-52 "Controller skeleton from PINChange" (p.9-50).

Figure 9-52: Controller skeleton from PINChange

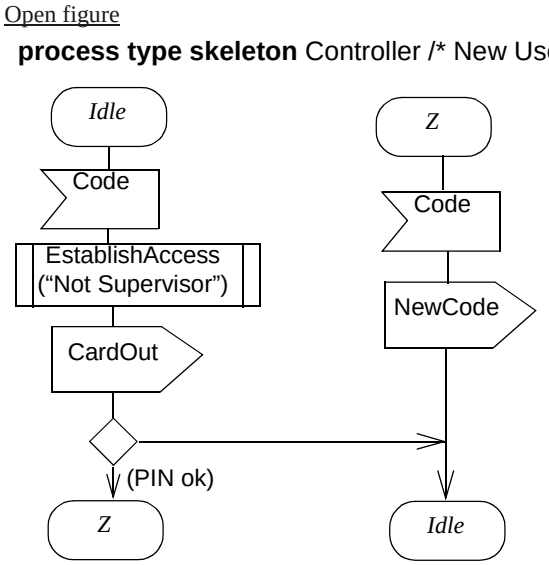
Open figure



Similarly we reach the skeleton shown in Figure 9-53 "Controller skeleton from New User" (p.9-51) from service New User.

Figure 9-53: Controller skeleton from New User

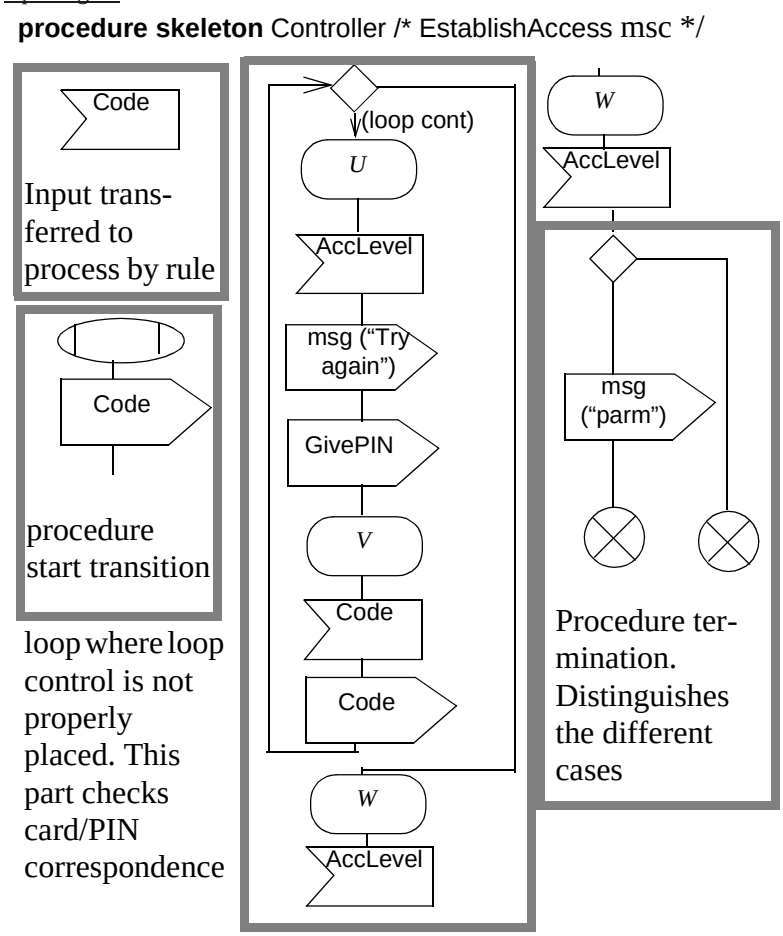
Open figure



In both these skeletons we refer to the procedure EstablishAccess and this can also be given a skeleton from the same technique shown in Figure 9-54 "Controller skeleton from EstablishAccess" (p.9-52).

Figure 9-54: Controller skeleton from EstablishAccess

Open figure



The technique now calls to unify the two transitions given by the two services as they both have the same "signature" (Idle,Code).

Manual program transformations of the skeletons

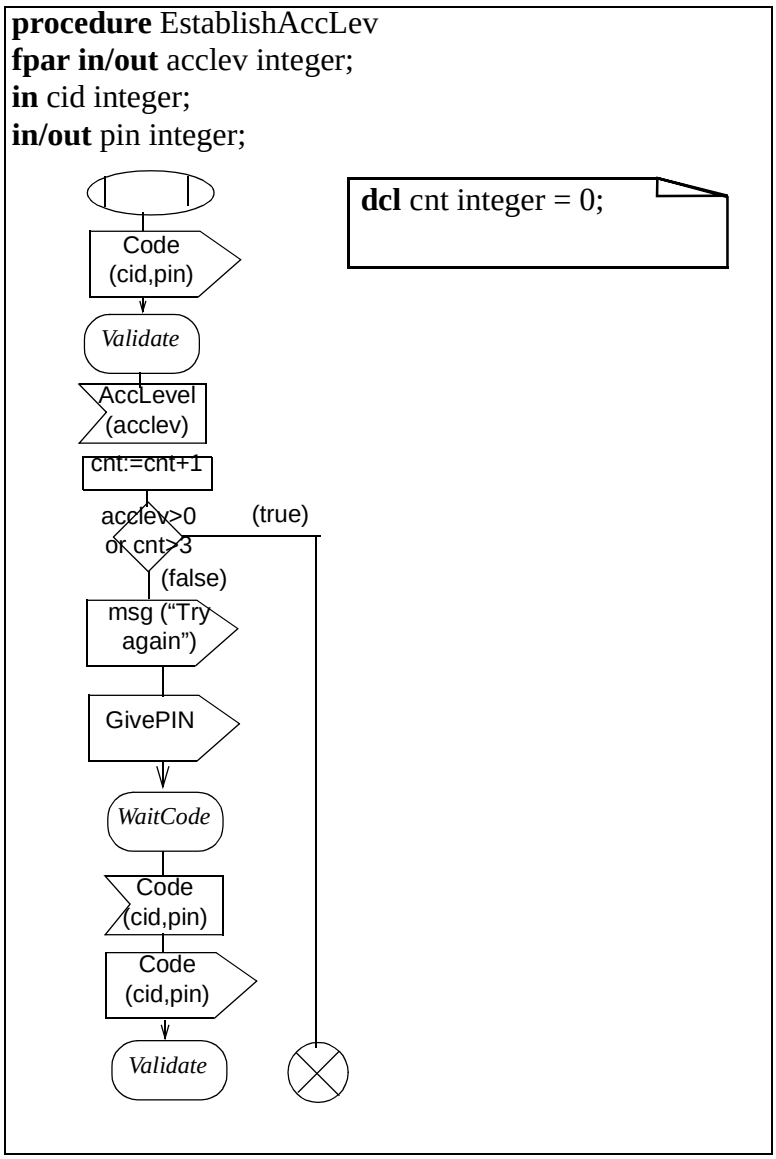
The distinction between the services actually lies within the EstablishAccess procedure skeleton. This is not practical. Therefore we divide the EstablishAccess concept in three parts which have been shown in Figure 9-54 "Controller skeleton from EstablishAccess" (p.9-52).

1. Input of Code. This has been moved out of EstablishAccess already by applying the rule 11 of the technique.
2. The main body loop. This is not very well produced automatically. This is due to the fact that MSC does not indicate anything about the cause of the loop exit. In this case the loop control is on the access level returned from the signal AccLevel. We reformat the loop such that the loop control is properly placed. We isolate this part in a procedure EstablishAccLev. We add local data and parameters.
3. The decision on access level. The return from EstablishAccLev is an indication of which access level the card and PIN is on. This indication is used to distinguish between the services and its different outcomes.

The result of our manual massage of the skeletons is shown in Figure 9-55 "procedure EstablishAccLev in Entry:Controller" (p.9-53) and Figure 9-56 "Controller in Console" (p.9-53).

Figure 9-55: procedure EstablishAccLev in Entry:Controller

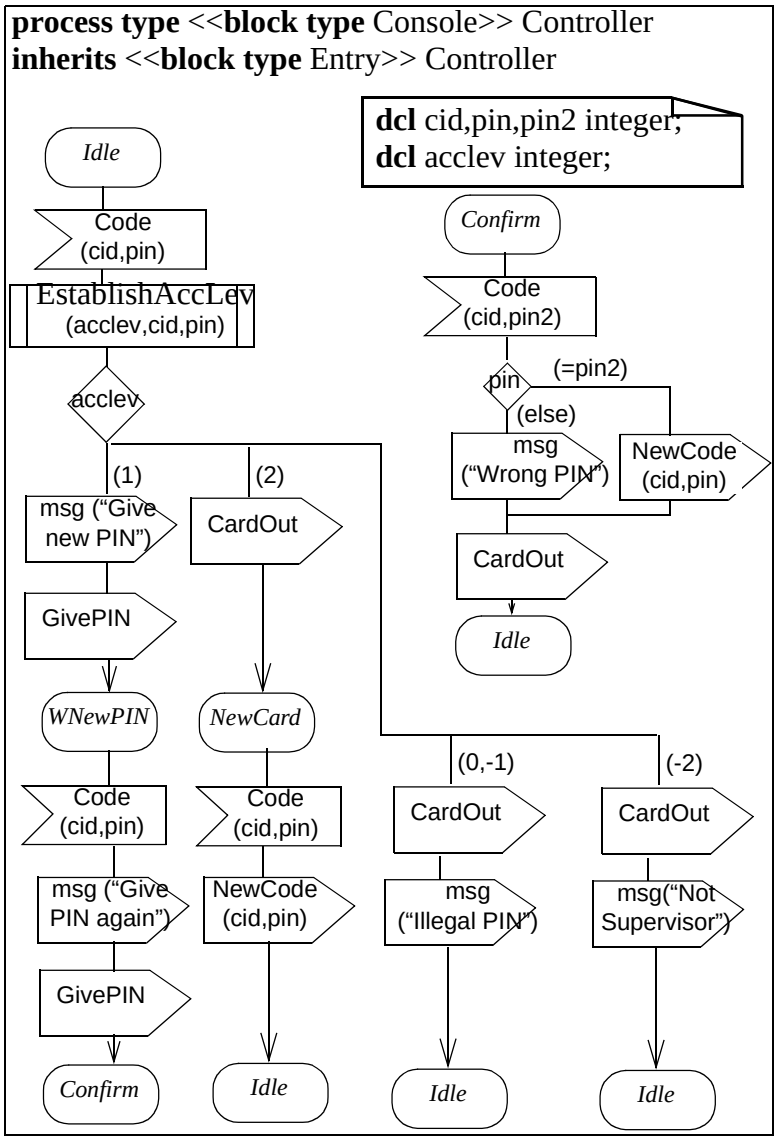
Open figure



The reader is urged to discover the resemblance between the skeletons and the final SDL diagrams. We have added sensible names to the states and data to the decisions and the signal parameters. Model checking of the services PIN Change and New User wrt. Controller of Console will result in consistency. This is almost a tautology, but the exercise should be performed nevertheless because the introduction of data and simple transformation of loops etc. *may* also contain (stupid) errors.

Figure 9-56: Controller in Console

Open figure

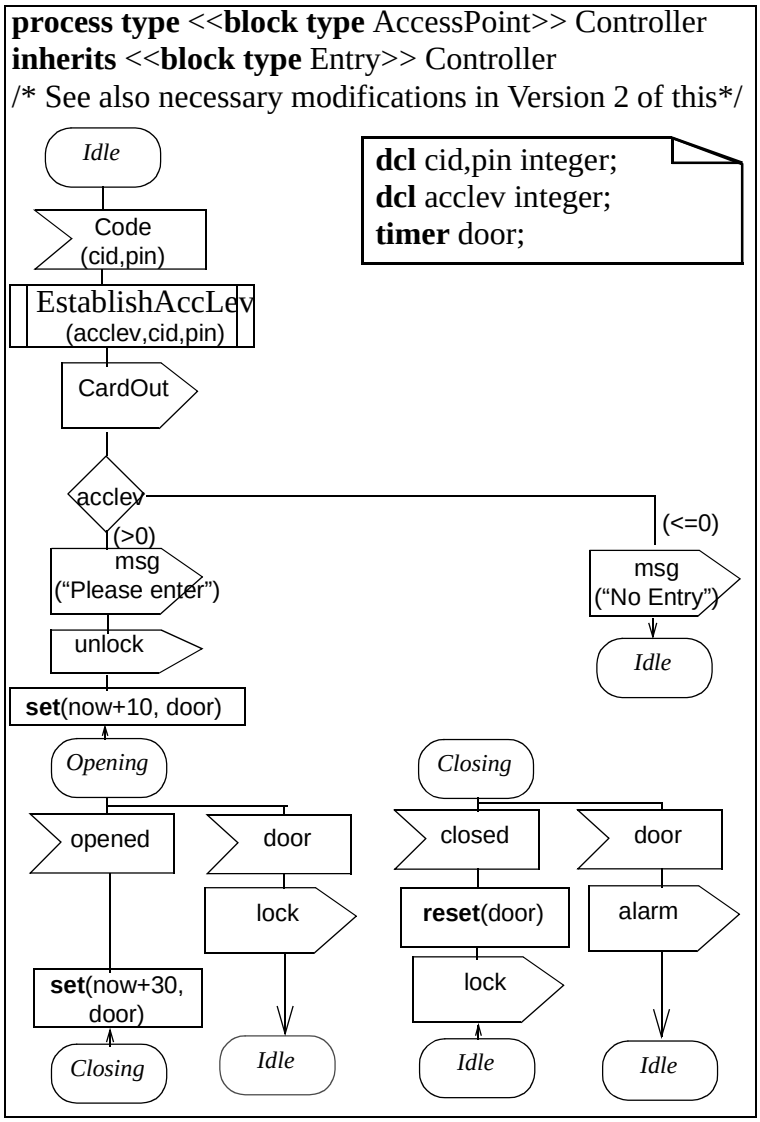


The service level Controller of AccessPoint

Figure 9-57: Controller in AccessPoint V1

Open figure

WE could produce the Controller of AccessPoint in a very similar fashion, and the result would be equally attractive. We assume that the technique has been used or that the controller has been made manually. In either case we choose to take advantage of the produced procedure EstablishAccLev. The result is shown in Figure 9-57 "Controller in AccessPoint V1" (p.9-54).



Model checking for consistency between MSC and SDL

We may of course perform model checking and show that the Controller may perform the service User Access as specified by the MSC referring to the auxiliary MSC Figure 9-35 "OpenDoor" (p.9-34). The consistency is not difficult to assert.

Is this sufficient?

Validation of Controller of AccessPoint

The fact is that it is not sufficient to assert the consistency between the service specification in MSC and the SDL specification. The reason is that a normal MSC document only defines a set of possible traces and not necessarily *all legal* traces.

Furthermore MSC is not well suited to disclose whether there are cases which have not been thought of. The main reason for that is probably because MSC does not describe causalities, but merely orderings of events. MSC says nothing about what causes under-

lie alternative cases of execution. Also within a sequential execution there is not by necessity any connection between the events. They may even reside on different concrete processes.

SDL on the other hand, defines imperatively how executions are. We know exactly what the causes of alternative courses of action are. We know the set of possible executions on every stage in the execution. This is not the case with MSC.

We now return to our Controller of AccessPoint. We have shown that it may perform the service which was called for namely User Access. So what may be the problem?

We want to perform an analysis of Controller which could disclose whether there is any chance of executing a default transition (which is considered harmful). This is a type of analysis which is provided by modern validators. Manually many problems can be found by looking closely at situations where there is a possibility legally to receive input from several independent sources.

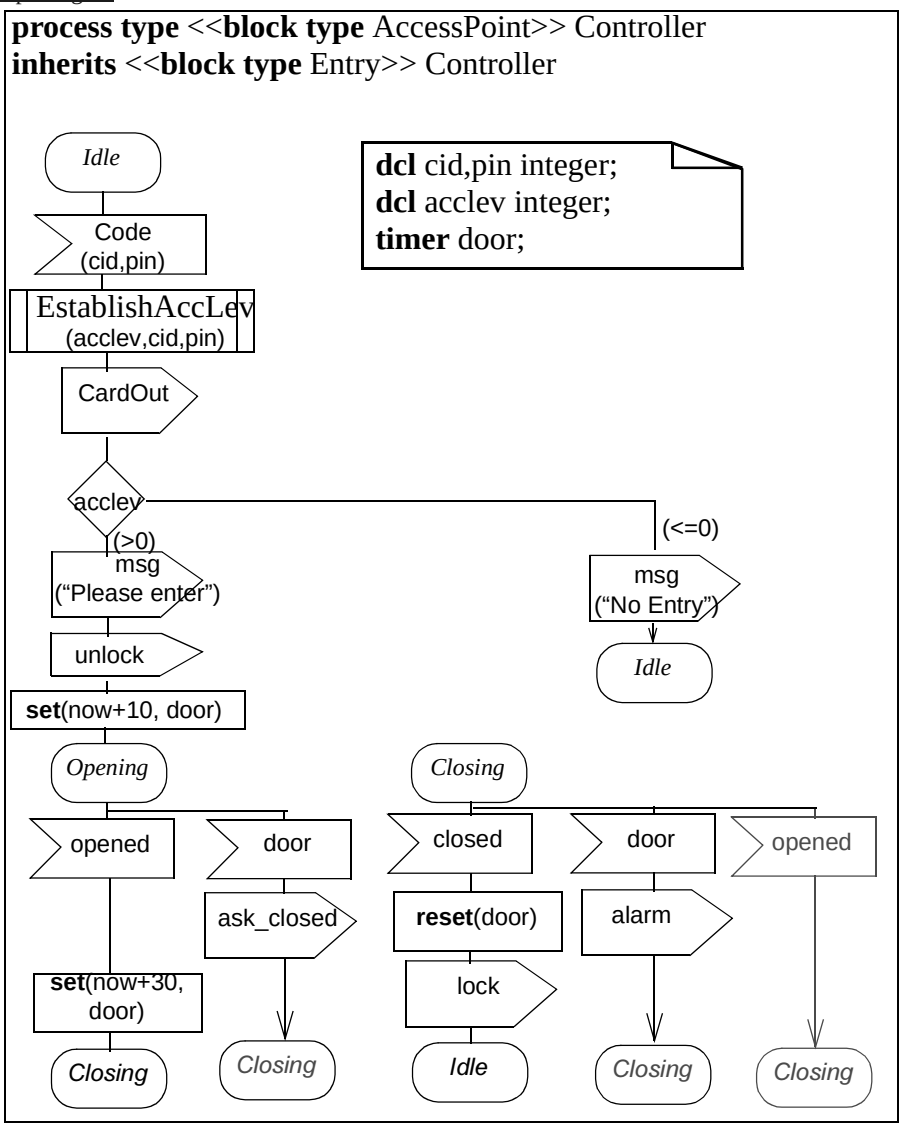
In our case we have a situation where we can either receive an *opened* signal or a *door* timer. We have covered them both, but have we covered the case where they actually appear both at the same time? What if the user has just managed to open the door, but before the *opened* signal has reached the controller, the timer expires. This situation is theoretically possible, but one can easily imagine that it will not happen often in reality. In our specification in Figure 9-57 "Controller in AccessPoint V1" (p.9-54) we will consume the timer first and then enter *Idle*. Then we must consume *opened*, but this is illegal!

In Figure Figure: "Controller in AccessPoint V2" (p.9-57) we have modified the process slightly such that the reaction to the timer does not take for granted that the door is actually closed. Instead the process will ask whether it is closed and expect a *closed* signal from the door if it is indeed shut. We have also added a transition which throws away the opened signal in the case where the timer has expired in close concurrency with the opening of the door.

The sharp reader will also notice that the same story may repeat itself concerning the timer and the *closed* signal in state *Closing*. We have not properly specified a recovery after the timer has expired with an open door. We have merely changed the nextstate of the transition triggered by the expiration of the timer such that the process will not enter *Idle* before the door is closed.

Figure: Controller in AccessPoint V2

Open figure



The Authorizer

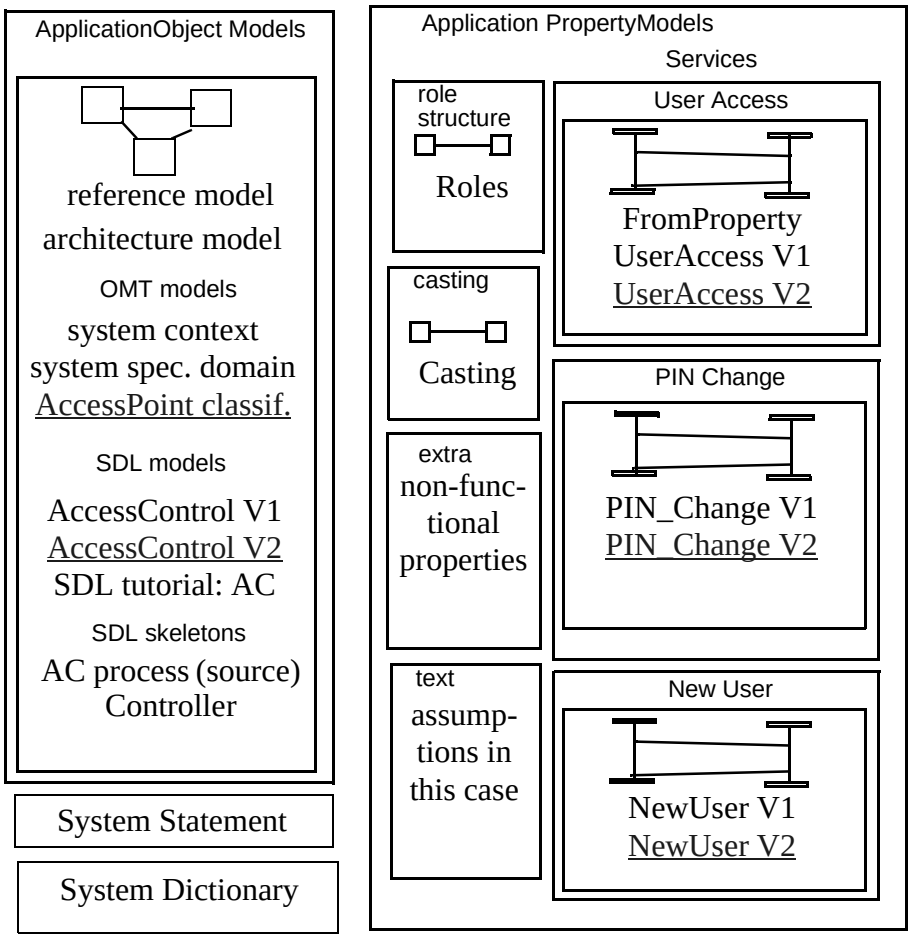
We have not given the specification of the Authorizer here. We trust that the reader will be able to provide this as an exercise. Authorizer can be made as a process which reacts to signals Code and NewCode and delivers AccLevel.

Summary Application Specification and Design

In Figure 9-58 "Application Descriptions of Access Control" (p.9-58) we give the overview of the application descriptions.

Figure 9-58: Application Descriptions of Access Control

Open figure



We have chosen to include all versions of the descriptions both those which proved erroneous and those which we have so far found no flaws.

Architecture modelling

(See also activities).

Make architecture design

The starting point for implementation design is made up of the design constraints and the functional design above. Figure 9-40 "AccessControl System V2" (p.9-38) shows the top levels of the functional design.

The functional design also contains process graphs, data and signal definitions.

Trade-off between hardware and software

The *physical user interfaces* of the AccessControl system are the Panels and Doors. The Panels have to be physically located at the Doors where the users need them. Does this mean that the AccessPoints should be physically distributed as well? Or should they be physically centralised in the vicinity of the CentralUnit?

To answer these questions we look at the channels represented in the functional design in order to find which ones are best suited to cover physical distances. SDL signals are defined independently of physical distances. One is therefore free to localise processes physically apart. But there will always be a certain delay and cost associated with signal transfer over distances. We therefore look for channels carrying a low signal traffic without strict timing constraints.

We want to distribute processes in a way that minimises the bandwidth needed over physical channels. (Distribute along the channels with few interactions and relaxed timing constraints. Keep strongly coupled processes together. This will often mean that a fair bit of processing should be performed physically close to the external interfaces.)

In the system the channels between the AccessPoints and the Authorizer satisfy these criteria best. We therefore decide to let these channels be the ones that cover distances.

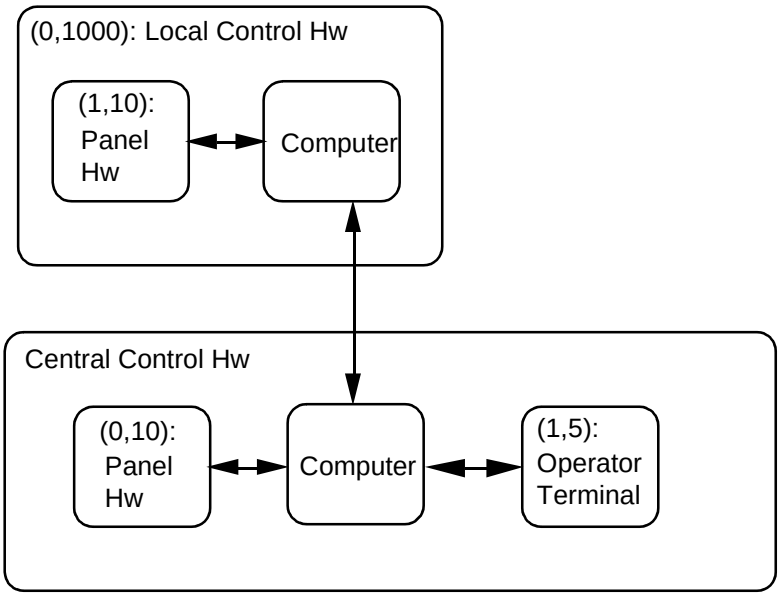
Does this mean that each AccessPoint should be a physically separate unit? Not necessarily. We may implement several AccessPoints in one computer when their Panels and Doors are located close to each other.

Make hardware design

Perhaps some AccessPoints can be co-located with the CentralUnit too? This could be a solution for small installations. A scheme that can be physically distributed or centralised depending on the physical distances and the size of each installation seems attractive. We therefore select the structure shown in Figure 9-59 "A possible Access Control hardware structure" (p.9-60) as our first attempt at a hardware architecture.

Figure 9-59: A possible Access Control hardware structure

Open figure



There will be at least one block of Central Control Hardware and from zero up to 1000 blocks of Local Control Hardware. In this architecture we intend to implement the Controller processes and the CentralUnit processes in software running on the various computers. We are not sure as yet how to implement the PanelControl processes, but software seems to be most likely option if the computer capacity permits.

At this stage of design there are still many open questions. What kind of computer to use, what kind of communication links to use and so on.

Before we carry on, we make two general observations:

1. The hardware architecture is different in structure from the functional design.
2. Some communication protocols will be needed to support the communication between the local and central hardware.

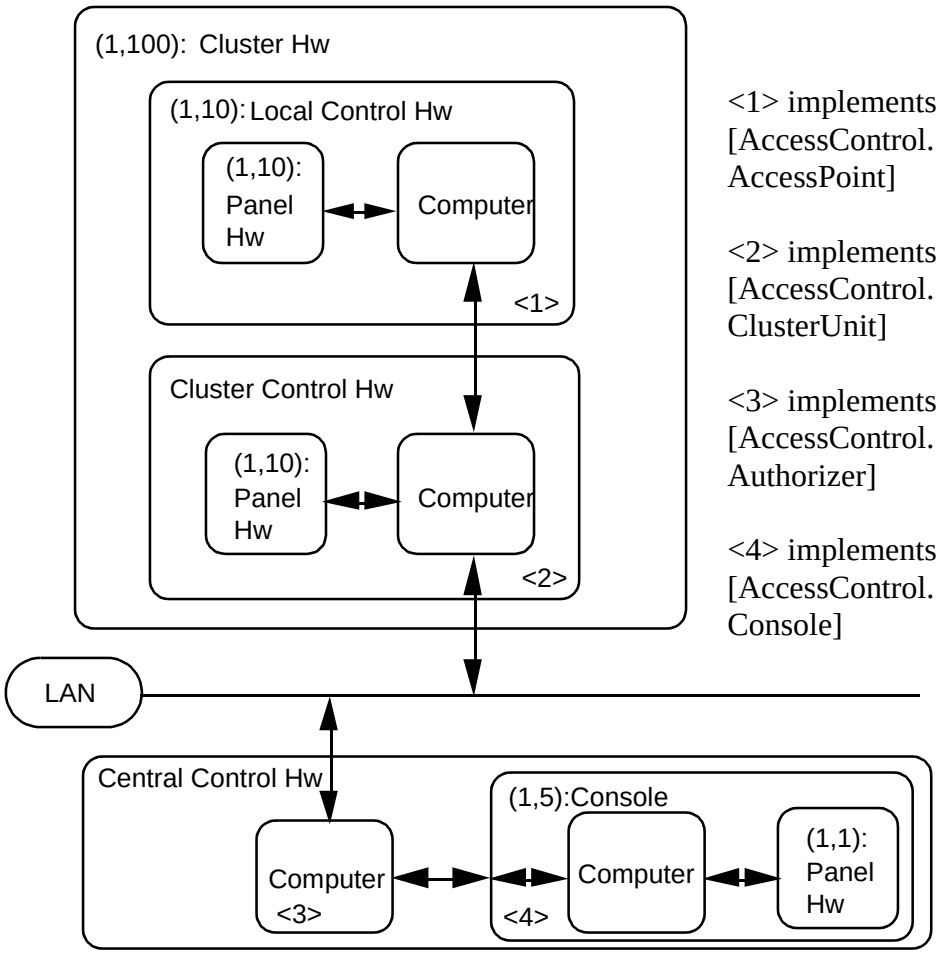
We calculate the Mean peak load of the AC system and have to conclude that it is possible that the central computer performing the validation with protocols etc. will be overloaded at peak load. We recall our non-functional requirements in Figure 9-20 "Non-functional requirements of AC system" (p.9-21).

We therefore decide to distribute the validation load to a number of *ClusterUnits*, each serving a group of *AccessPoints*.

In Figure 9-60 "Hardware structure with cluster Units" (p.9-61) is shown the new hardware structure we propose to use for large installations. The Central Hardware will be without Panels in this case. The clusters will be connected to the central hardware through a local area network, the LAN in Figure 9-60 "Hardware structure with cluster Units" (p.9-61).

Figure 9-60: Hardware structure with cluster Units

Open figure



In this solution the validation database will be distributed. There will be a copy of the central authorization process (and its database) in each cluster. This means that the Authorizer must handle updates in a distributed database. This introduces a new problem to solve in the functional design, but the AccessPoints and the authorization processes in each cluster may (hopefully) work just as before.

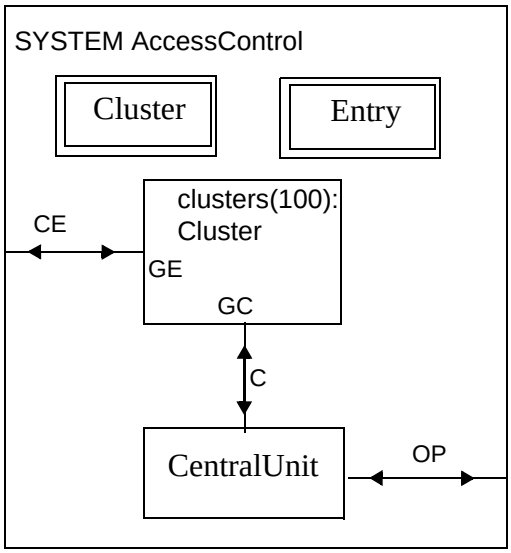
Make Software Design

We have assumed that the majority of SDL processes are implemented in software running on the various computers. Each of these computers will contain software that implements the local functions, the cluster functions and the central functions. In addition, they will have software for intercomputer communications, local input–output and error handling. Finally, they will most likely have an operating system.

As our next step, we return to functional design to make a refined and restructured definition of the complete functional properties that are visible to the user. Figure 9-61 "Redesigned Access Control system V3" (p.9-62) illustrates the top level of the resulting SDL description.

Figure 9-61: Redesigned Access Control system V3

Open figure



The initial functional design described in Figure 9-40 "AccessControl System V2" (p.9-38) was structured to render the functional properties with minimum complexity and maximum clarity, while the implementation design shown in Figure 9-60 "Hardware structure with cluster Units" (p.9-61) was structured to render the physical construction.The functional blocks in Figure 9-61 "Redesigned Access Control system V3" (p.9-62), map directly to the hardware blocks in Figure 9-60 "Hardware structure with cluster Units" (p.9-61).

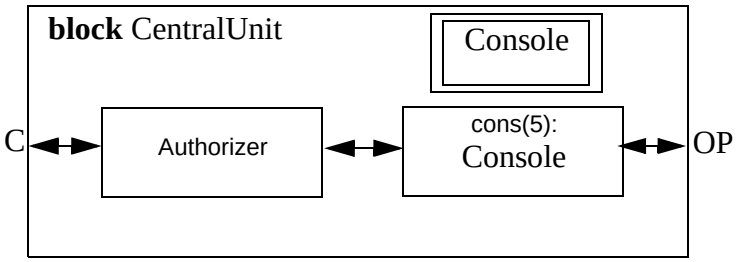
Note that the restructuring does not mean that everything has to be redefined. A majority of the processes from the first functional design may be left unchanged. As they are defined as stand alone types, it is a simple matter to put them into a new structural context together with some new processes.

In Figure 9-63 "Cluster with LocalUnits and ClusterUnits" (p.9-63) and Figure 9-64 "AccessPoint used in both LocalUnit and ClusterUnit" (p.9-64) we take *AccessPoint* as an example. We will use instances of *AccessPoint* in the *LocalUnits* as well as in the *ClusterUnits*. Those in the *ClusterUnits* will have direct, local access to the validation process, whereas those in the *LocalUnits* must communicate via physical links and protocols, but the signals will be the same.

The *CentralUnit* consists of the block *Authorizer* and the block set of block type *Console*.

Figure 9-62: Central Unit

Open figure



The reader should notice that we have done a restructuring of the functional specification, but we have not in this section made any specific software architecture description. We have taken this as being implied by the functional specification and the hardware specification.

Figure 9-63: Cluster with LocalUnits and ClusterUnits

Open figure

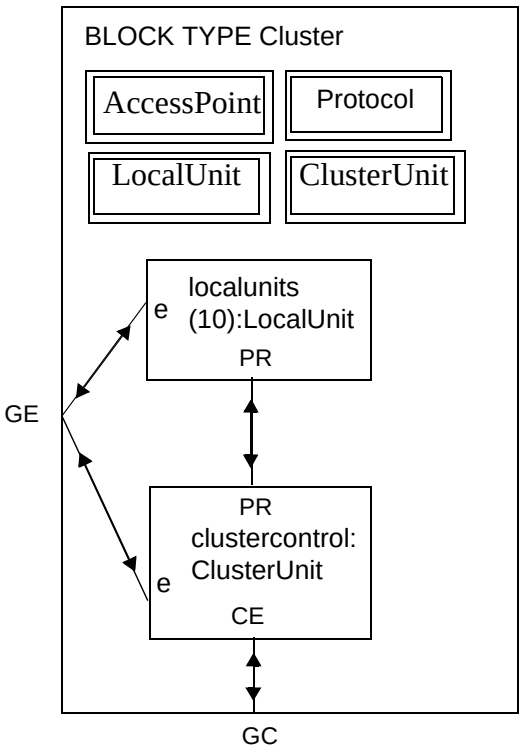


Figure 9-64: AccessPoint used in both LocalUnit and ClusterUnit

Open figure

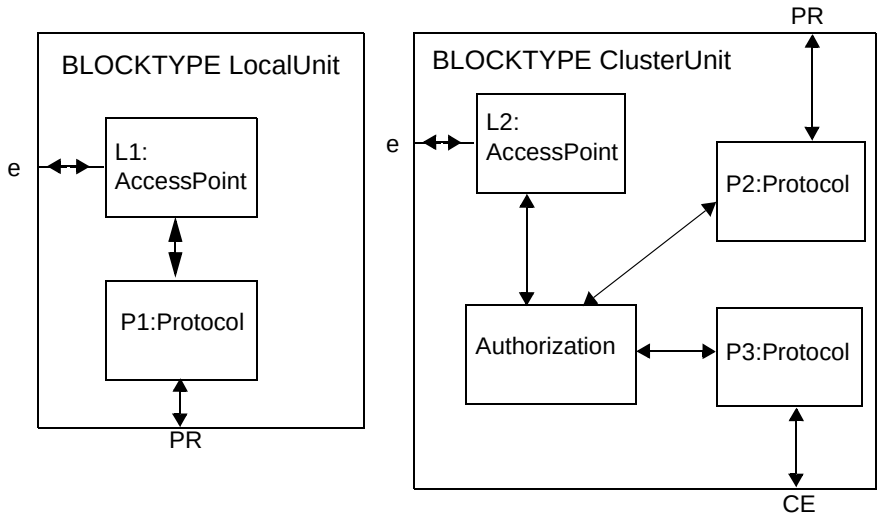


Figure 9-65: Make framework specification (See also activity**)**

In our development we have considered the Access Control system a first product and we have not put much effort in trying to generalize the design of the Access Control system such that a whole family can be built upon the design.

With the architecture design in Figure 9-60 "Hardware structure with cluster Units" (p.9-61), the task is now to make a line between the application specific parts and the implementation specific parts of the design, look for stable structures in both and define the parts that vary from system to system as virtual types.

When distribution has been taken into account we will have a stable structure of a *CentralUnit* and a number of *Cluster* blocks. Each *Cluster* will have a stable structure of protocol and validation parts, while for different access control systems the type of *AccessPoints* may be different.

If we turn the structure in Figure 9-61 "Redesigned Access Control system V3" (p.9-62) into a framework and let the distribution parts be stable, then we get the system *type* in Figure 9-66 "System type AccessControl V4 as a framework" (p.9-64). The structure of all system of this type will have at least the structure of one CentralUnit and a number of Cluster objects. As the type Cluster is a virtual block type, the Cluster objects in different system subtypes may be of different types.

Figure 9-66: System type AccessControl V4 as a framework

Open figure

The **virtual type** *Cluster*, defined in Figure 9-67 "Cluster as part of a framework" (p.9-65), similarly contains a stable implementation specific part and a stable structure, where the application specific **virtual** *AccessPoint* type is used to define just one part of the structure.

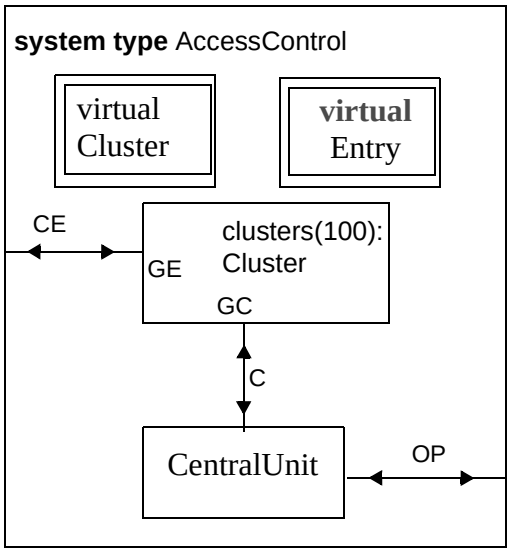
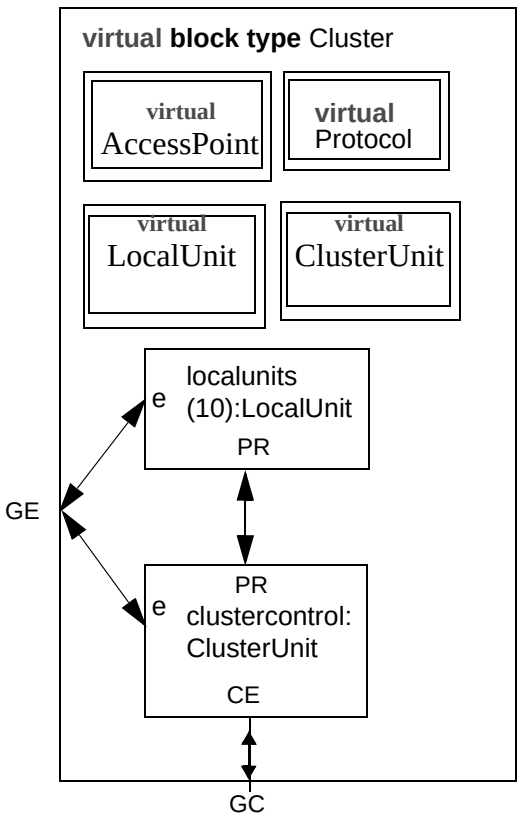


Figure 9-67: Cluster as part of a framework

Open figure



Note that the definitions of *Protocol*, *LocalUnit* and *ClusterUnit* are not affected by turning the system into a framework. The reason is that these constitute the fixed structure and are only using the type *AccessPoint* for defining blocks. We have, however, also chosen to make *Protocol* and *LocalUnit* virtual since the flexibility to change the support system of the framework can also be practical. If the stability of their descrip-

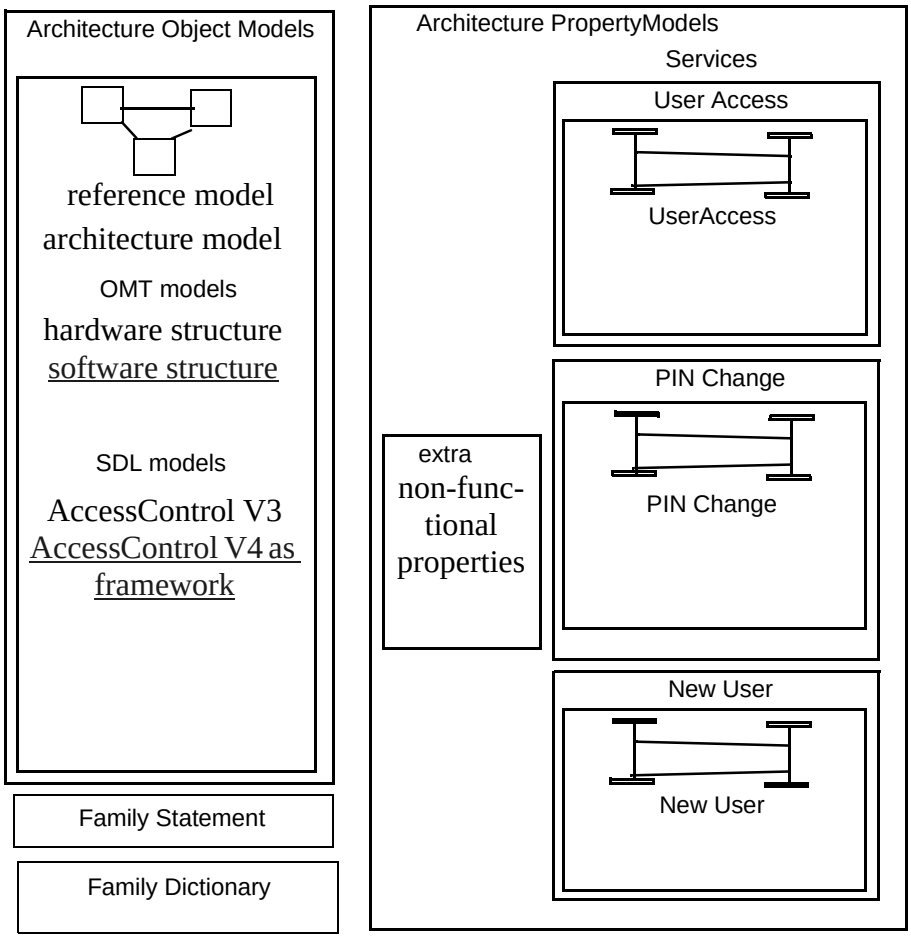
tions should be emphasized, either the types should remain non-virtual or it is possible to make a redefinition of *Cluster* (say *StableCluster*) where the virtual types *LocalUnit* and *Protocol* (say) are finalized. Building on *StableCluster* will then make it impossible to alter any definitions of *LocalUnit* and *Protocol*.

Summary Framework

We show a sketch in Figure 9-68 "Architecture of AC System" (p.9-66) of the structure of the descriptions under the framework strategy. The MSC documents have not been updated and are thus no included in the documentation.

Figure 9-68: Architecture of AC System

Open figure



Neither the application statement nor the dictionary have been updated.

Application Evolution

New functionality to be introduced:

- Blocking stations which can be disabled from the Authorizer;
- Logging stations which can log any transaction on the AccessPoint.

This version of the Integrated Methodology does not cover this in further detail. To look into the functionality of blocking and logging stations please refer to the SDL Tutorial.

Documentation

Domain Descriptions

For a structural map of the domain descriptions see Figure 9-17 "Domain Descriptions of Access Control" (p.9-17).

Family and Application descriptions

For a structural map of the application descriptions see Figure 9-58 "Application Descriptions of Access Control" (p.9-58).

Architectural Descriptions

For a map of the architectural descriptions see Figure 9-68 "Architecture of AC System" (p.9-66)

List of figures