Expressing Systemic Contexts in Visual Models of System Specifications.

José D. de la Cruz, Alain Wegmann, Gil Regev

School of Computer and Communications Sciences
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne, Switzerland
josediego.delacruz@epfl.ch[1], alain.wegmann@epfl.ch, gil.regev@epfl.ch

Abstract. Decision support systems can be used to manage systems. Managed systems are described by system specifications. System specification notations, such as UML, often separate in different diagrams the static specification and the dynamic specification of the system of interest. As a consequence, precious contextual information disappears, leading to misunderstandings during the interpretation of the specification. We claim that these problems result from a mechanistic view of reality. By taking a systemic view of reality, it is possible to develop a specification technique that integrates the static and dynamic aspects of the system and, hence, make the contextual information explicit. The benefit is the creation of more expressive system specifications that are less error prone when used for designing and managing systems.

1 Introduction

In decision-making, people have to take decisions about controlled systems. Often a controlled system is managed by an operator and a decision support system [1]. When the controlled system is not in an expected state, the operator needs to decide what actions to take to regulate the system. When taking its decision, the operator uses a model of the controlled system. Such models are often represented with a graphical specification. We will show in this paper that these graphical representations do not make explicit a lot of the contextual information (for example, what are the effects of the different actions of the controlled system). Our approach for system specification makes contextual information explicit in diagrams. Thus, the operator, who works with a graphical system specification to take her decisions, has less implicit information to include in her decision process.

Our goal is the representation of contextual information in graphical system specifications. We base our work on the Merriam-Webster [2] definition of context; a definition which is well formed from a systemic modeling standpoint. They define the "context of something" as the "interrelated conditions in which something exists or

[1] The work presented in this paper was supported (in part) by a fellowship awarded by the CFBE and the EPFL-SOC.

occurs". Our goal with this paper is to understand both what "something" is and what the "interrelated conditions" are when this definition is applied to graphical system specifications.

UML [3] is one of the most popular graphical notations for modeling software (UML takes its roots in software development), systems [4] and businesses [5]. UML consists of a set of diagrams that can be categorized into structure-related diagrams (part I of [6]) and behavior-related diagrams (part II of [6]). Examples of structure related diagrams are: class, object, and deployment diagrams; examples of behavior-related diagrams are: interaction, activity, and state diagrams[2]. The choices made to classify the diagrams as either structure-related or behavior-related dramatically reduce the possibility to express the "interrelated conditions" in which model elements exist: i.e. reduce the capability to express context.

The way UML diagrams are categorized (and the kind of elements they represent) can be explained by the epistemological principles that underlie mechanistic human thinking, and in particular scientific thinking. As claimed by Lemoigne [8], the universal ontology principle drives us to look for theories that are "time-invariant" (*universal ontology epistemological principle*). This leads all of us (including the UML language designers) to carefully separate diagrams that are time-independent (structure-related) from diagrams that are time-dependent (behavior-related). Quite often, the time-independent diagrams are considered as more general as the time-dependent diagrams. For example, a class diagram describes the concepts known by a system at all times. A sequence diagram is "just" an occurrence of a behavior (among the many possible). To make a class diagram specific to a context (i.e. to show only the concepts necessary to describe a behavior) appears to reduce its generality and, hence, to reduce its value. So, class diagrams are usually not context-dependent. This example illustrates why diagrams do not represent time-dependent together with time-independent model elements. So such diagrams cannot represent contextual information (i.e. the "interrelated relations" between these two kinds of model elements).

This categorization of diagrams would not be so much of a problem if some additional graphical elements were added to represent the relations between the elements represented in the separate diagrams. For example, in a sequence diagram, it could be useful to represent the state changes of objects that result from the processing of messages. Unfortunately, another epistemological principle strikes here. It is one aspect of reductionism, called *Occam's razor* [9]. This principle leads us to minimize the number of elements used in our theories (and in the diagrams). As a consequence, we do not consider useful to represent the context of what we model. We consider that this context is obvious, already known by the modelers, and will clutter the diagram with superfluous information, distracting the modeler from the "essential" message.

[2] The use case diagram is an exception as it describes structure and behavior, however, the use case diagram has some limitations (e.g. diagram centered on one system) which limits its use as a general purpose diagram [7].

In summary, the categorization of the UML diagrams and the loss of contextual information could be explained by the implicit application to the modeling language of both the *universal ontology* and the *Occam's razor* epistemological principles.

We claim that we can define three modeling principles that can contribute to bring contextual information back into graphical modeling languages applied to system specifications. Based on these principles, we define the elements that need to be represented in a system functional specification and we propose a graphical notation that makes the context explicit (i.e. the modeling elements are represented with their "interrelated conditions"). The notation is relatively close to UML, so people with UML experience can understand it.

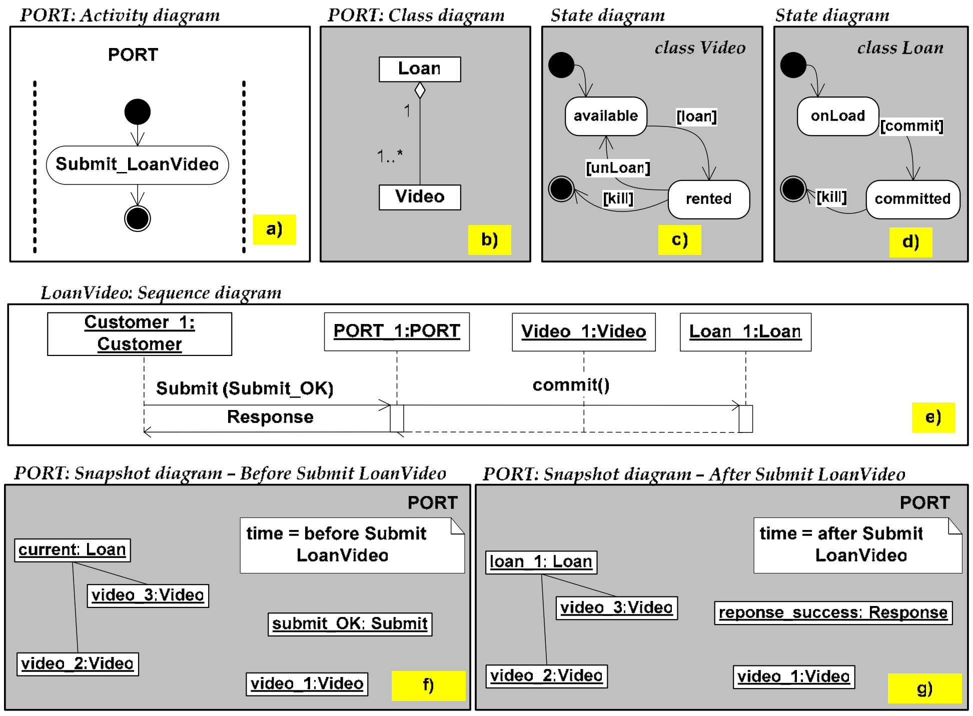
The structure of the paper is as follows. In Section 2 we present an example to illustrate the research questions we address in this paper. This example will be used through the paper. In Section 3, we present the three principles underlying our notation and explain what needs to be made explicit in the diagrams. In Section 4, we present the solution we obtained for the example of Section 2. Finally, in Section 5, we relate our work to other research.

2 Example and Research Questions

We illustrate our point with the example of a video rental store. The focus of the analysis is the store's IT system called the *POint of Rental Termin*al (*PORT*). We provide, in this Section, the UML specification of one service provided by the *PORT*.

The *PORT* registers the loan of videos as well as the return of rented videos made by the customers. For renting videos, a customer must first log into the *PORT*. Then, she selects the videos she wants to rent. The selected videos are all put into one virtual basket. At the end of the selection process, the customer performs either a *Submit* or a *Cancel*. The *Submit* action confirms the rental of the videos already in the basket; a loan record will then be created in the *PORT*. If the customer cancels the action, no loan record will be created.

The concepts known by the *PORT* in this example are: *video* (in states *rented* and *available*), *customer*, and *loan (*in states *onLoad* and *committed)*. We will model the *Submit* action, when the videos are already selected and a confirmation from the customer is required to finalize the loan.



```
package VideoStore
context PORTSelf::Submit_LoanVideo ( submit: Submit)   : Response
inv:    Video_List <> null    -- ok

pre:
      submit_OK <> null                              &&            // User input for committing the Loan
      submit_OK.oclIsInState(OK)                     &&            // User input for committing the Loan
      video_List_1 <> null                           &&
      video_List_1.asSet.size() > 0                  &&            // Avoid duplicates
      current <> null                                &&
      current. oclIsInState ( onLoad )               &&            // State is OK
      current.asSet.size() > 0                       &&            // Avoid duplicates
      current->forAll ( v:Video | v.oclIsInState( rented ))
post:
      response_success <> null                       &&            // User input for committing the Loan
      response_success.oclIsInState(Success)         &&            // User input for committing the Loan
      video_List_1 <> null                           &&
      video_List_1.asSet.size() > 0                  &&            // Avoid duplicates
      caddie = null                                  &&
      loan_1 <> null                                 &&
      loan_1. oclIsInState ( onLoad )                &&            // State is OK
      loan_1.asSet.size() > 0                        &&            // Avoid duplicates
      loan_1 ->forAll ( v:Video |
                  v.oclIsInState( rented )           &&
                  v@pre.oclIsInState( rented )       &&
            )                                        &&
      loan_1.asSet() = current@pre.asSet()                  // The sets have the same elements
--endpackage
```

h)

Fig. 1. A simplified set of UML diagrams + OCL code for defining the service *Submit_LoanVideo*.

Figure 1 describes the service *Submit_LoanVideo*. This service is represented as an activity executed by the *PORT* (Fig.1a). The *PORT* system has one list, which includes the videos that belong to the video store (Fig.1b). The states of both videos and loans change over time (Fig. 1c and 1d). The objects of class *Video* can be in either state: *available* or *rented*. The transitions are triggered by the messages shown in Fig 1*c*. In general, there is one state machine by object. We can also see that the objects dialog during the activity (Fig.1e). For instance, the *PORT* has videos *Video_1*, *Video_2* and *Video_3* before the loan (Fig. 1f); then, the customer loans *Video_2* and *Video_3*. As a result, *Video_2* and *Video_3* become part of the loan *Loan_1* (Fig. 1*g*). *Video_1* is the only instance that is not rented at the end of the activity *Submit_LoanVideo*

The UML system specification is made up of a set of seven diagrams and one OCL description.

Each diagram represents a different concern of the system. This makes the individual diagrams more legible than if they would be merged together. But, as a consequence, the overall specification cannot be easily understood as a whole. There are no visible links between the elements represented in the different diagrams. In most cases, it is the modeler who “glues” the diagrams together within his or her mind [10]. The seven diagrams required to model the single, partial scenario *Submit_LoanVideo* in figure 1 illustrates this problem.

The OCL description is used to add contextual information. OCL [11] is a textual language for UML. It is used as a complement to the diagrams but is not fully integrated with them. OCL adds one more artifact to interpret.

This example raises the following questions:

1. “Can we use fewer, more integrated, diagrams?” The principle of separation of concerns [12] says that each aspect of the description must be treated individually. However, our experience shows that the overview of the specification is lost in the process.
2. “Can we represent more contextual information in diagrams?” OCL is the means for communicating information that binds diagrams together. We would like to introduce similar information as OCL but graphically.
3. “Can we create a model that graphically shows what the services we model actually do?” Traditionally, the changes made by the service are made explicit by snapshots. However, to fully understand what happens, the modeler needs to write and read OCL in addition to all the diagrams. As a consequence, the task of understanding the goal the modeler wants to achieve with the service requires a large effort from the modeler [10].

3 Systemic Modeling of Systems

In order to make context explicit in system specification, we need to first define the concepts we use to specify systems. Our modeling ontology is inspired by the

ISO/ITU standard RM-ODP [13] that defines how to model systems. A formal description of RM-ODP we have developed is available in [14].

We define the model elements system and environment[3]. The "**system"** can be defined as a whole (i.e. only its externally visible functionality is described) or as a composite (i.e. its composition is described). One cannot design a system of interest (SoI) without taking into consideration the immediate "**environment"** that interacts with it. The "**supra-system"** of the SoI is "the next higher system in which the SoI is a component. The immediate environment is the supra-system minus the SoI itself." [15].

System functionality is described with "**information objects"** and "**actions"**. Information objects describe information that the system has about itself and about its environment. The system's information is modeled with the "**state"** of information objects. Actions modify the state of information objects. Note that the information objects and the actions can be further considered as whole or as composite.

In our approach, we also define two hierarchies: the organizational level hierarchy and the functional hierarchy[16]. If a system is represented as a whole at a given organizational level then it is represented, at the next organizational level, as a composite (i.e. showing its sub-systems). The "**organizational level hierarchy"** is useful to capture system components (e.g. an IT system made of software components). The "**functional level hierarchy"** captures different levels of detail in the functionality of the systems. For example, an action might be described as a whole in one level of functionality and as a composite in the next one. All these concepts are informally defined in [16] and formally in [17].

The definition of context is the "interrelated conditions in which something exists or occurs" [2]. The term "something" can be replaced by the concepts defined in our modeling ontology. We therefore need to show the "interrelated conditions" (in terms of system, environment, information object, action, state) in which the system, environment, information object, action, state exist. Our main focus in this paper is system specification. Hence, we will mainly analyze the relationships between the concepts: system, information objects, states and actions. We will briefly mention the relation between the system and its environment but this is the topic of future work.

The "interrelated conditions" between these key elements can be grouped to create a few principles that can explain how to describe context:

- System / environment complementarity
- Action / information object & state complementarity
- Whole / composite complementarity

In this Section, we present these modeling principles and their impact on the notation. In some cases, we illustrate the principle with illustrations taken from the *PORT* example.

[3] In RM-ODP, the term "system" designates an entity in the universe of discourse and not a model element in the model. For sake of simplicity, in this paper, we consider that the term "system" designates the representation of a "system" in the model. So "system" is a model element. This paper does not address any issues related to the universe of discourse.

System / environment complementarity
The most obvious “interrelated condition”, as stated in the Merriam-Webster definition, is between the system and its information objects and actions. All actions and information objects are within a system. In the notation, the information objects and the actions should be surrounded by a rectangle that represents the system to which they belong (as shown in Figure 2 for the Information Object *X* of *S*).

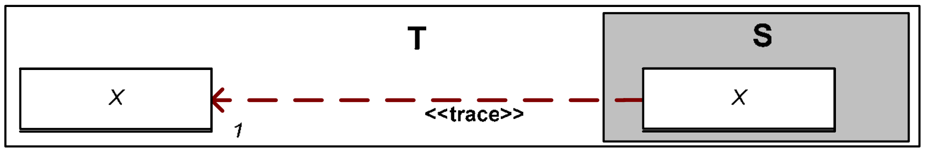


Fig. 2. “Interrelated condition” between a system and its environment

Another “interrelated condition” that we will just mention in passing here is the relation between the system and its environment. A system has information about itself and about its environment. For example, in the video store example, the *PORT* has some information about the physical video that exists in its environment. So, one of the “interrelated conditions” in which an information object exists is the relation between the information object itself (belonging to a system of interest) and what it represents (in the environment of the system of interest). In Figure 2, *T* represents the supra-system of *S* and *X* shown in the left part of *T* is in the immediate environment of the system *S*. The information object *X* in *S* represents the knowledge of *S* about *X* in *T*. This is represented by the <<trace>> relationship. A more systematic analysis of the relationship between a system and its environment is part of our future research.

We capture the necessity to relate actions, information objects and states to a system in which they exist and the necessity to relate a system to its environment as the “system / environment complementarity principle”.

Action / information object complementarity
An action changes the state of one or more information objects. Quite often the action’s identifier makes implicitly references to the information objects that change state. For example, the modeler can guess that an action *rent* in a video store refers to a video and a renter because she knows the meaning of the word *rent*. With this knowledge, the modeler can guess the relationship between the elements in the diagrams (e.g. in UML, the action *rent* shown in an activity diagram and the *video* concept shown in a class diagram as illustrated in Figure 1). This is not sufficient as the concepts used can be defined in multiple ways. For example, it is unclear if the rent is related to some payment or not. We can eliminate this ambiguity by making explicit, in one diagram, the “interrelated conditions” between the actions and the information objects involved in the actions. So one of the “interrelated conditions”, in which an action exists, is the set of information objects that are modified by the action.

This relation between actions and information objects can be further developed. When a modeler represents an action, implicitly she is referring to a change of state of information objects. Vice-versa, when a modeler represents the change of state of information objects, she is referring to an action. This is known as the state/behavior duality. This duality leads the modeler to often consider that modeling either the action or the state change is sufficient to specify the system. We claim that, if we want to make the context explicit, we need to model both. So, the "interrelated conditions" in which an action exists include the states (before and after the action) of the information objects involved in the action. Vice-versa, the state is related to the actions that consumes them or modifies them. Figure 3 represents an action *A* that modifies the state of the information object *X* and created the object *Y*. The arrow between the states *S1* and *S2* illustrates the state transition resulting from the execution of *A*.

To show the relation between the actions and their effects, we need a way to relate them. This is done by an identifier (*eventA* in Figure 3), that designates all changes. It is also necessary to represent what triggers the action's execution (and how the action enables system exchanges with its environment). This is done by special information objects that act as parameters. The stereotype <<*par in*>> indicates that the information object is an input parameter for the system (<<par out>> would indicate an output parameter). In Figure 3, the fact that *parA* enters the system via *Param* (an input parameter) triggers the state change of *X* (rounded arrow from *S1* to *S2*) and the creation of *Y (*shown by the multiplicity change *0→ 1).* All the changes are marked by *eventA*. One of the labels is underlined and this highlights what triggers the changes. We can see in Figure 3 that a UML class diagram, a UML activity diagram and a UML state diagram can be merged together.

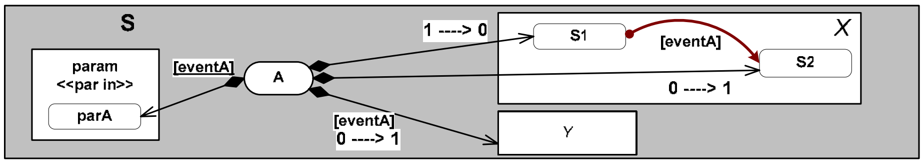


Fig. 3. Action/State complementarity

Information objects live in the context of actions. We call this the ***context of existence*** of an information object**.** In figure 3 we see that action *A* is the context of existence of all the information objects shown. This means that none of the represented information objects will out-live the action *A*. In Section 4, we will illustrate how we can represent that an information object can potentially exist during the whole lifecycle of a system (this will be done by relating information objects to the action that represents the overall system lifecycle).

We capture the necessity to relate action, information objects and state as the "action / information object / state complementarity principle".

Whole / composite complementarity

In systemic modeling, the modeling elements (e.g. system, action, information object) can be interpreted *as whole* or *as composite*. A modeling element *as whole* appears as monolithic and its internal structure is hidden for the modeler. A model element *as* *composite* exposes to the modeler the component elements and the way they are related. The *whole* is defined as the result of the composition of its components. The composition of the components can be understood as we know what the *whole* is. So, as system theory shows[18], *whole* and *composite* are both necessary as they define the context of each other: the *whole* is part of the "interrelated conditions" of existence of the *composite* (and vice-versa). It is because we can recognize the *whole* that we can see the components and vice-versa. For example, in a video store, the action *Rent* is understood because we implicitly know that such action includes getting information about the renter and getting information about the videos to be rented. Vice-versa, it is because we know the component actions *GetRenter* and *GetVideos* that we can imagine the existence of the composite action *Rent*. Even if this point appears to be "hair splitting", it is actually crucial if we want to make explicit the implicit information that is hidden in the diagrams.

In figure 4, we apply this principle to the action *A*. Action *A as whole* (the bubble on the top) is equivalent to *A as composite*, composed of the actions *A1, A2, A3* and of the constraints of execution between them. The triple association between *A as whole* and *A as composite* makes this equivalence relation explicit. In other words, *A as whole* can be substituted by *A as composite*.

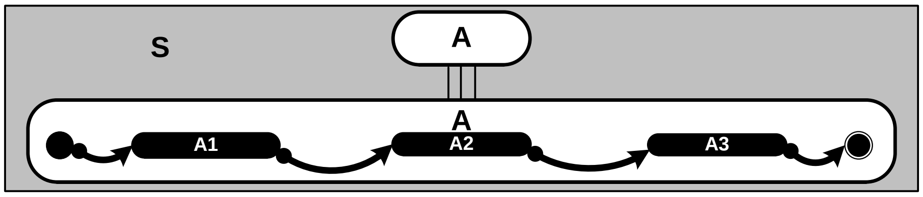


Fig. 4. Action *as whole* and action *as composite*

The action *A as whole* and the action *A as composite* define two functional levels in the functional level hierarchy. One interesting question is: what action is at the top of the functional hierarchy of a system of interest? In other words, what is the action in a system which includes all the actions the system does? This is the action that corresponds to the **system lifecycle**. This lifecycle action captures the behavioral context in which all other actions exist. The lifecycle action starts when the system is created and ends when the system is dismissed (i.e. the action lasts from system "cradle to grave"). The system's lifecycle action is at the top of the functional hierarchy. The added-value to model the lifecycle is to force the modeler to think on how the system is created and how it is phased out. This can highlight critical issues in terms of system initialization or information retrieval at system phase-out. In figure 5 we can see that the *lifecycle* of the system *S* is equivalent to the set of all the actions that the system can execute (*A, B, C* and their execution constraints). Each action can be further refined. For example, action *B* is decomposed into actions *B1, B2, B3* together with their execution constraints. In the figures 4 and 5 we have used the most simple execution constraint among the actions, but in real cases, more complex constraints might be used, including loops, partial order, etc.

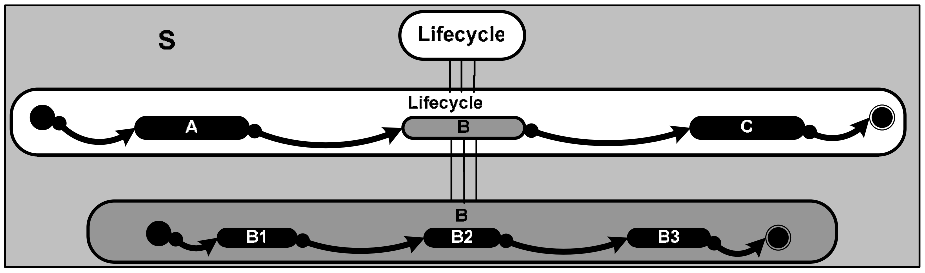


Fig. 5. Lifecycle composition with actions

The same complementarity whole / composite of actions shown in figures 4 and 5 can be found on information objects. However, presenting this is out of the scope of this paper.

We capture the necessity to relate whole and composite as the "whole / composite complementarity principle".

Summary
In this section we have presented three principles that drive our modeling process:

- System / environment complementarity
- Action / information object & state complementarity
- Whole / composite complementarity

These principles permit the inclusion of contextual information in graphical system specification. Contextual information has to do with the "interrelated conditions" in which model elements exist. Through our systemic approach we have shown that any system has a lifecycle, made up of actions that will change the state of the information objects that exist in the system. These information objects represent information about a system of interest and about its environment. The relations between all these model elements need to be explicit if we want to model explicitly the context

To accept adding the additional contextual information we propose, requires abandoning the two epistemological principles we have presented in the Introduction.

The universal ontology epistemological principle states that we can have "universal models". We have shown in our discussion that the actions and the information objects are bound to the lifecycle of the object in which they exists. So, instead of a universal ontology principle we rely on the "lifecycle epistemological principle" that we define as "all actions and information objects exist in specific lifecycles (of systems and actions/information objects)".

To accept to contextual relations between the model elements requires being less strict in applying the Occam's razor. We still wish to limit the number of concepts we use in the theory (for example in our method, we have 6 main concepts: system, environment, action, information object and state). However, we should not limit the analysis of the relationships between them to only relations between information objects or relations between actions (as, for example, normally done in UML). We should keep the possibility to represent all relationships between actions, information objects and state. In other words: we claim that the Occam's razor should not be applied to eliminate the contextual information.

4 Application

Systemic Enterprise Architecture Methodology (SEAM) models complex systems by applying systemic principles [16]. SEAM is based on explicit epistemological principles (such as the "lifecycle principle") and on the three complementarity principles we just presented. SEAM aims at a holistic and hierarchical representation of the systems. Its main application is enterprise architecture and service modeling.

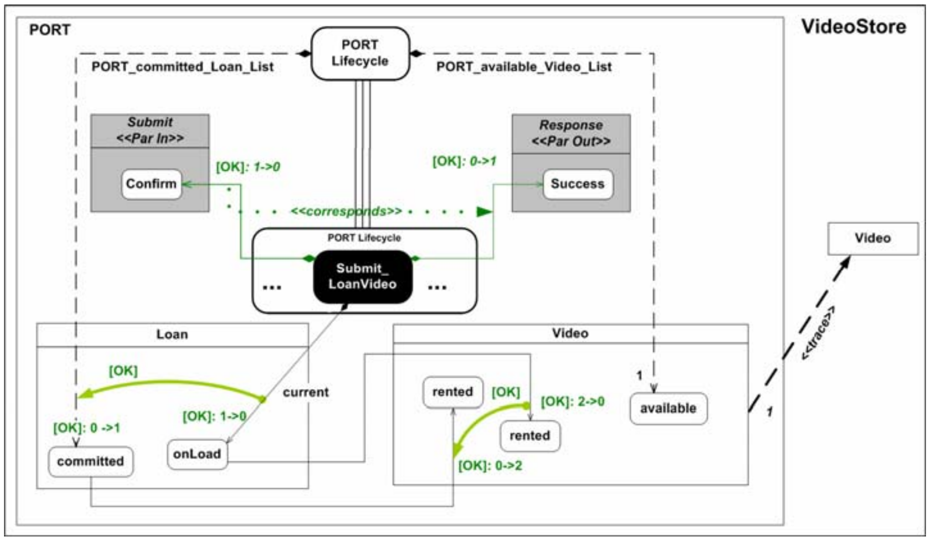


Fig. 6. SEAM diagram for service *Submit_LoanVideo*.. Whenever the input parameter *Confirm* arrives, the *OK* event is enabled in all transitions.

Figure 6 illustrate our technique to represent contextual information. Figure 6 represents, in one diagram, the *Submit_LoanVideo* action that was modeled with the 7 UML diagrams and the OCL code in Figure 1. We can interpret the diagram in Figure 6 as follows:

The *PORT* system manages:

- *Video* which represents the *PORT* information about the physical video in the *VideoStore*. The Video can be either *rented* or *available*. The system keeps the knowledge of which *Video* is available (through the lifecycle association called *PORT_Available_Video_List*).
- *Loan* which represents the fact that a video is rented. The *Loan* can be either *committed* or *onLoad* (when prepared). The system keeps the knowledge of

which *Loan* exists (through the lifecycle association called *PORTcommitted_Loan_List*).

All these represent what we would call the “invariant” information in the system. This information will exist during the whole lifecycle of the system (but will not be universal!).

We need now to represent the action *Submit_LoanVideo*.

- Actions have parameters. The parameter information objects are special cases of information objects that enable communication with the environment of the system. *Submit* and *Response* (in gray in Figure 6) are parameters. Parameters exist only during the execution of the action and they are not known to other actions.
- Actions have pre and post conditions. These are visible in Figure 6 (beginning and end of the rounded arrows). The action takes as pre-condition a *Loan* in the *onLoad* state and the parameter *Submit* in the state *Confirm.* The parameter is coming from the system’s environment. As a post-condition, the action creates a *Loan* in the *committed* state and outputs a message to the environment (via the parameter *Response*).
- Actions are triggered. In Figure 6, event *[OK]* that corresponds to the fact that the *Submit* input parameter is in the state *Confirm*, triggers the action. As a result, all transitions marked by *[OK]* (e.g. *Loan* from *onLoad* to *committed*, *rented; Video* referenced by *Loan/committed* and not by *Loan/onLoad* and output parameter *Response* generated) are executed.
- Actions work on specific information objects. This is represented by the relation *current* that goes from the action towards the information object Loan. The relation *current* exists only in the context of the action *Submit_LoanVideo*. It is not known by any other actions in the lifecycle of the *PORT* system. On the other hand, the information objects *PORT_available_Video_List* and *PORT_committed_Loan_List* exist in the context of the lifecycle of the *PORT* system; thus, they exist for all actions. Information can be exchanged between actions with the interplay between information objects that exist at different levels of action (e.g. information exchanged between actions using information objects that exist in the system lifecycle). The detailed presentation of this aspect is out of the scope of this paper.

If wished, it is also possible to represent the action with two diagrams: one showing the state of the system before the action (Figure 7a) and one after the action (Figure 7b). These two diagrams explain the meaning of Figure 6 but provide less information as the relation between both diagrams is not made explicit. The effect of the action can be understood by the fact that the cardinality of the association between *Loan* in state *committed* and *Video* in state *rented* changes from *0* to *2* while the other one changes from *2* to *0*.

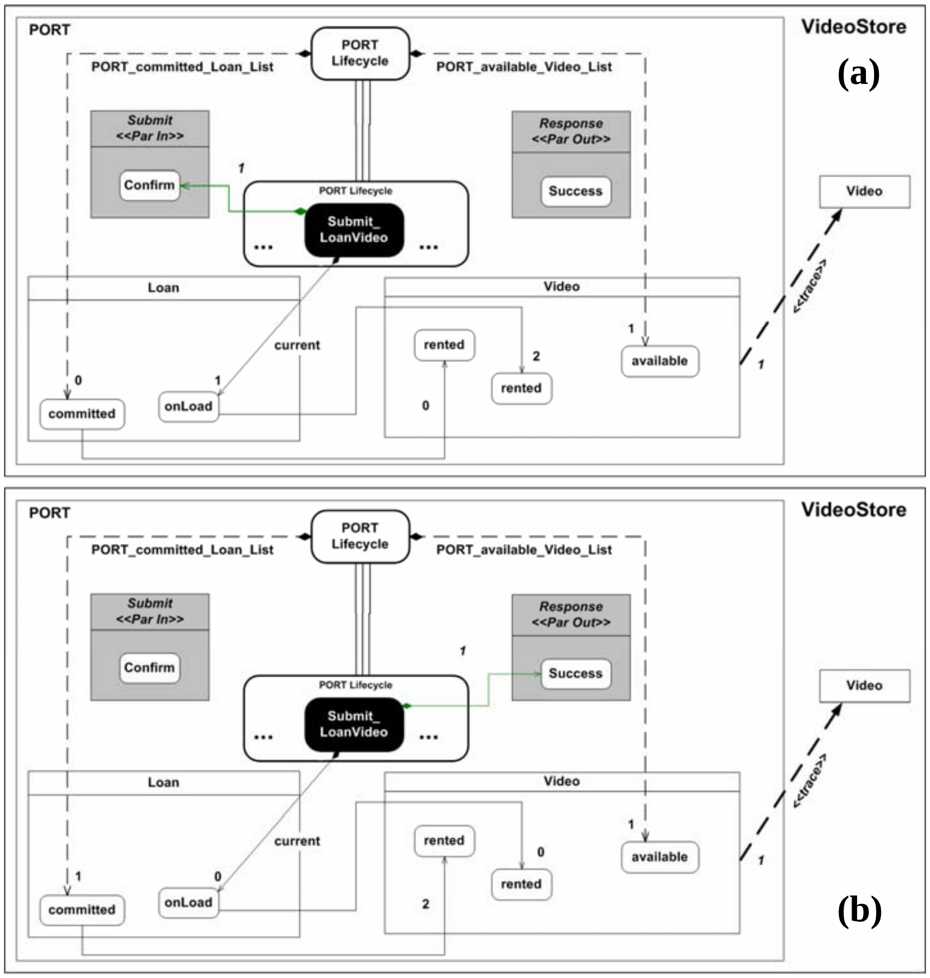


Fig.7. Describing behavior of action *Submit_LoanVideo* in two diagrams. (a) is the state of the system before the action, (b) the state of the system after the action.

In summary, Figure 6 is a general purpose dynamic diagram that integrates the pre- and post-conditions shown in Figure 7 into a single diagrammatic context. Its purpose is to make explicit the changes caused by the action *Submit_LoanVideo*; these changes are not evident in figures 7*a* and 7*b* (as no relations are shown between the two diagrams). We can say that change is a property that emerges to the modeler whenever he compares two diagrams of system state as the ones in Figure 7. With our notation, as shown in Figure 6, we make such changes apparent to the modeler.

Now, we can show how the contextual information we propose to add can address the problems identified in Section 2. Let us discuss each problem:

- "Can we use fewer, more integrated, diagrams?" We have shown that the diagrams can provide more contextual information if "structure-related" and “behavior-related” elements can appear in the same diagram. It could even be conceived that one diagram can represent all information (about a simple situation). This shows that the definition of what should be in the diagram does not need to be enforced by the modeling language but should only be the result of modeler’s taste and needs.

- “Can we represent more contextual information in diagrams?” With the definition of context we have selected, it is possible to make all contextual information explicit. There is only a limit dictated by the legibility of the diagram.
- “Can we create a model that graphically shows what the services we model actually do?” With our notation we have shown that we could show pre and post conditions for actions. They are represented by the “change” relationship.

5 State of the art

UML [5] is the *de facto* standard of the software industry for object-oriented analysis and design.. Studies on usability of UML, such as [10, 19], focus only on single diagrams, avoiding the problems caused by the heterogeneous notations. However, some authors consider the UML richness as notation has negative effects since dealing with many items require more analytic effort from the designer[20]. It is clear that UML can be improved to show more contextual information as we define it in this paper.

Reasoning with Diagrams (RwD) is an initiative where two sets of graphical formal specifications have been created, namely Spider and Constraint diagrams. Such diagrams are used for reasoning about properties of the resulting object’s instances after an action. The first ones describe sets and constraints between sets [21]. The latter ones extend spider diagrams to show relations between sets and their elements [22]. The specification of behavior can be done via 3-D contract boxes [23]. RwD specifications are similar to ours. However, their scope is limited to expressing constraints, and their notation is incompatible with UML (they use 3-D layouts for contracts, and special notation for expressing constraints).

Object-Process Method (OPM) is Dori’s work on system modeling [24]. As SEAM, it is a holistic approach. Dori has created a graphical and a textual notation (OPD and OPL, respectively), that are exchangeable at any moment of the development. This notation is incompatible with UML but the OPM tool supports the translation of OP models to UML.

Brézillon proposes the use of Contextual Graphs [25]. They distinguish different viewpoints (contexts from observers) on a process or situation, and inherently the roles of the agents that participate in the system. They facilitate reasoning for decision making but they are not designed to represent behavior as presented in this paper. Our approach complements the one of Brézillon as our goal is to support precise system specification.

6 Conclusions and Future Work

This paper addresses the graphical representation of contexts in visual system specification. We show that the two epistemological principles (*universal ontology* and *Occam's razor*) can explain the way we currently structure graphical specifications. We claim that, if we adopt the *lifecycle epistemological principle* (instead of the *universal ontology principle*) and if we do not eliminate the contextual relationships, we can get system specifications that exhibit the "interrelated conditions in which the model elements exist". In other words: we can have system specifications that make the contextual information explicit. It is worth highlighting that the Merriam-Webster context definition is very well written from a systemic standpoint.

Practically, we identified three modeling principles that need to be adopted to model the contexts:

- System / environment complementarity
- Action / information object & state complementarity
- Whole / composite complementarity

With these principles, we have shown how an action (*Submit_LoanVideo)* can be graphically specified in its in context if we make explicit its relations to the information objects it modifies (*Video* and *Loan*) and its relations to the parameters that enter and leave the system (*Submit* and *Response).*

The proposal made in this paper has been validated in software engineering courses and by making an explicit mapping between the proposed notation and the Z specification language [26]. Future work includes: modeling of relations between system and environment, modeling of relations between whole / composite, scalability of the notation, tool support.

In the context of decision support system, our proposal can bring benefits when it is necessary to document the systems that need to be controlled with a decision support system. This is necessary if we expect that the operators, that will use the decision support system, should have an explicit understanding of the controlled system they will have to regulate. Further work includes the validation of the notation in relation with an existing decision support system.

References

[1] Brézillon, P., Pasquier, L., Pomerol, J.-C.: Reasoning with contextual graphs. European Journal of Operation Research 136 (2002) 290-298

[2] Merriam-Webster OnLine. Merriam-Webster, Inc. [Online]. Available: www.m-w.com

[3] OMG. Unified Modeling Language (UML).[Online]. Available: www.omg.org

[4] OMG. SysML Specification v. 0.9 Draft.[Online]. Available: http://www.sysml.org/artifacts.htm

[5] Eriksson, H.-E., Penker, M.: Business modeling with UML Business Patterns at work. Wiley (2000)

[6] OMG. (2004) Unified Modeling Language: Superstructure 2.0 Final adopted specification, ptc/03-08-02.[Online]. Available: http://www.omg.org/docs/ptc/03-08-02.pdf
[7] Wegmann, A., Genilloud, G.: The Role of ¨Roles¨ in Use Case Diagram. In: Proc. 3rd International Conference on the Unified Modeling Language (UML2000) (2000) 210-224
[8] Le Moigne, J.-L.: Les épistémologies constructivistes. Presses Universitaires de France, Paris (1995)
[9] Audi, R.: Cambridge dictionary of philosophy. Cambridge University Press (1999)
[10] Dori, D.: Why significant UML change is unlikely. Communications of the ACM (CACM) 45 (2002) 82-85
[11] Richters, M., Gogolla, M.: On Formalizing the UML Object Constraint Language OCL. In: Proc. Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling (1998) 449-464
[12] De Win, B., Piessens, F., Joosen, W.: On the importance of the separation-of-concerns principle in secure software engineering. In: Proc. ACSA Workshop on the Application of Engineering Principles to System Security Design (WAEPSSD) (2002)
[13] ISO/IEC: 10746-1, 2, 3, 4 | ITU-T Recommendation X.901, X.902, X.903, X.904. Open Distributed Processing - Reference Model. http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm (1995-1996)
[14] Naumenko, A.: Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP. EPFL (2002)
[15] Miller, J. G.: Living Systems. 2nd edition. University Press of Colorado (1995)
[16] Wegmann, A., Balabko, P., Le, L.-S., Regev, G., Rychkova, I.: A Method and Tool for Business-IT Alignment in Enterprise Architecture. In: Proc. Conference on Advanced Information Systems Engineering (CAiSE'05) (2005)
[17] Le, L. S., Wegmann, A.: Definition of an Object-Oriented Modeling Language for Enterprise Architecture. In: Proc. Hawaii International Conference on System Sciences (HICSS'05) (2005)
[18] Weinberg, G.: An Introduction to General Systems Thinking: Silver Anniversary Edition. Dorset House Publishing (2001)
[19] Argawal, R., Sinha, A. P.: Object-oriented modeling with UML: a study of developers' perceptions. Communications of the ACM (CACM) 46 (2003) 248-256
[20] Miller, G. A.: The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. Psychological Review 63 (1956) 81-97
[21] Howse, J., Molina, F., Taylor, J., Kent, S., Gil, J.: Spider Diagrams: A Diagrammatic Reasoning System. Journal of Visual Languages and Computing 12 (2001) 299-324
[22] Gil, J., Howse, J., Kent, S.: Towards a Formalization of Constraint Diagrams. In: Proc. 2002 IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2001) (2001)
[23] Kent, S., Gil, J.: Visualising action contracts in object-oriented modelling. IEE Proceedings - Software 145 (1998) 70-78
[24] Dori, D.: Object-Process Methodology: A Holistic Systems Paradigm. 1 edn. Springer Verlag (2002)
[25] Brézillon, P.: Context dynamic and explanation in contextual graphs. In: P. Blackburn, C. Ghidini, R. M. Turner, and F. Giunchiglia, (eds.): Modeling and Using Context (CONTEXT-03). Springer Verlag (2003) 94-106
[26] Spivey M.: The Z Notation: A Reference Manual. European. 2nd edition. International Series in Computer Science, Prentice Hall , 1992.