# A 3D Visualization Approach to Validate Requirements

Alfredo Raúl Teyseyre

ISISTAN Research Institute, Facultad de Ciencias Exactas,
Universidad Nacional del Centro de la Pcia. de Bs. As,
Campus Universitario Paraje Arroyo Seco - (7000) Tandil - Bs. As., Argentine.

email: teyseyre@exa.unicen.edu.ar

**Abstract**

The importance of correctly determining the requirements of a system at the very beginning of the development process it is a well known fact. Experience shows that the incorrect definition of the requirements leads to the development of deficient systems, increases the cost of its development or even causes projects to fail. Therefore it is crucial for the clients to verify that the planned system satisfies their needs. In this context, visualization techniques appear as a useful tool to help the users in the process of requirements understanding and validation.

This paper describes an approach to validate system requirements with the user using 3D visualization techniques. The use of these techniques could reduce the communication gap between the clients and the developers resulting in a much more effective process of requirements validation. The approach tries to take advantage of the benefits of 3D visualization, complementing this with the advantages of formal specifications. As well as a research prototype tool, called ReqViZ3D, that materializes the proposal was developed. The merits of applying ReqViZ3D for the validation of requirements are illustrated using several case studies.

**Keywords**: Requirements Visualization, 3D Graphics, Requirements, Visualization, Formal Specifications.

## 1 Introduction

Meeting user requirements of a software system is a major challenge to software developers. Experience in a number of large projects reveals that a very large percentage of errors were consequence of the imprecision in the earlier stages of the development process [Potter et al., 1991]. Therefore, it is a well-accepted fact that it is crucial to express user requirements as completely, correctly and unambiguously as possible. Moreover, it is vital for the customers to be able to confirm that the planned system meets their needs, and this means that the system must be described in a way that they can understand it [Potts, 1991].

Many conventional approaches have been applied to validate requirements, but, most of them, fail in detecting errors [Kelly et al., 1992]. On the other hand, formal approaches, give clarity and precision at specification time. In that sense, formal specifications, enable us to denote unambiguously the meaning of a requirements specification document due to their formal syntax and semantics. However, except in safety-critical work, the cost of full verification is prohibitive [Jackson and Wing, 1996]. Moreover, formal specifications often fail in the user validation process since they are based on formal notations not always comprehensible by users and hence they fit better to software developers than customers. Therefore, in order to overcome these difficulties visualization techniques appear as an interesting alternative to explore.

Visualization is a method to comprehend information by the use of diagrams to represent it. Data are transformed into geometric representations that help users in the understanding process. In general, graphical representations provide a closer match to the mental model of the users than textual representations and take advantage of their perception capabilities.

In spite of their success in numerous computing areas, little research has been reported in the area of requirements visualization. The previous approaches enable developers to validate visually the specification of a system with the user, but their poor expressive graphics make difficult understanding. Moreover, neither of the works make use of current 3D graphics capabilities in order to present more real animations. However, 3D visualization techniques can be a powerful tool to facilitate the analysis and understanding of requirements. The use of visualization techniques could reduce the communication gap between the customer and developer resulting in a more effective requirements validation process [Parry et al., 1998]. In this context, the main objective of this work is using 3D visualization and animation techniques to validate requirements with the user.

This paper is organized as follows. Section 2 surveys current efforts towards requirement validation. Section 3 presents an overview of visualization and 3D graphics. Section 4 describes the approach and presents a case study. Two other examples are discussed in Section 5. Section 6 presents a brief description of the prototype tool ReqViZ3D. Finally section 7 outlines some preliminary conclusions and future work.

## 2 Related Work

Intuitively, the simple choice to capture the requirements of a software system is natural language. However natural languages specifications have been one of the main sources of ambiguity due its rich vocabulary and its expressiveness [Meyer, 1995]. As an alternative formal specification languages have been proposed. Formal specification languages have a formal syntax and semantics which makes it possible to unambiguously denote the meaning of the requirements. The best known formal specifications languages are Z [Spivey, 1988], B [Lano, 1996] and VDM [Jones, 1990] among others.

Although formal specification languages are precise, concise and unambiguous, which make them an excellent medium for communication between system designers, analysts and testers, they fail in the validation process with the customer: it is difficult for a customer to understand formal specifications because they are based on mathematical foundations and notations. However having formalized a system, automated support is available for validating the model by execution.

Many have proposed the use of executable formal specifications for the construction of prototypes to validate software requirements with the users at an early stage through feedback [Fuchs, 1992]. Techniques like execution have been introduced to overcome the difficulty of using a non executable specification language, allowing the specifier to either test or rapidly implement his specification document. Several researchers have reported success in executing subsets of Z translating them to languages such as PROLOG or LISP [Ozcan et al., 1998, Hazel et al., 1997].

Although specification execution can provide immediate feedback during the process of writing a specification and reduce the errors made at the early stages of the development process, seems yet to be more useful to developers rather than to users. This is due to the fact that the execution is still based on the underlying specification notations and the system is still described in a way that users can not understand.

Visualization techniques have been used in many computing areas. However, in spite of their success, little research has been reported in the area of requirements visualization. Most of the reported works are oriented towards the validation of requirements on specific domains, as for example real-time systems (IPTES [Pulli et al., 1991] and ENVISAGER [Gonzalez and Urban, 1991]), and do not address a wide range of problems as formal specification methods do. Moreover there is only one fixed graphic representation of requirements, for example nets, limiting in consequence the expressive power of the visual presentations. This could lead to poor expressive presentations that make difficult understanding.

Among the few works reported two of them can be remarked: VIZ [Ozcan et al., 1998] and POSSUM [Hazel et al., 1997]. Both systems enable the developer to validate visually specifications in Z. Technology provided by VIZ allows software developers to choose an appropriate representation of objects used in an executable formal specification and create animations of these objects in an interactive fashion. However, the system only supports the construction of simple presentations. On the other side, POSSUM facilitates the construction of complex presentation using Tcl/Tk, but it does not provide assistance in the construction of the presentations. Moreover, both systems only supports 2D presentations and does not take advantage of current 3D graphics technologies. In contrast, we attempt to fully exploit visualization techniques and also assist the developer in building the presentation.

## 3 Visualization

Lets first state the notion of visualization, which is defined by Card [Card et al., 1998] as follows: *"the use of computer-supported, interactive, visual representations of data to amplify cognition"*, where cognition is the acquisition or use of knowledge (see figure 1). So, the purpose of visualization is insight, not pictures. The main goals of this insight are discovery, decision making and explanation. That is the user may perform distinct types of visualization processes: exploratory (when the user does not know what is looking for), analytical (the user knows what is looking for in the data, trying to determine if it is there) or descriptive (when the phenomenon represented in the data is known, but the user needs to present a clear visual verification of it). Information visualization is useful to the extent that it increases our ability to perform these and other cognitive activities.

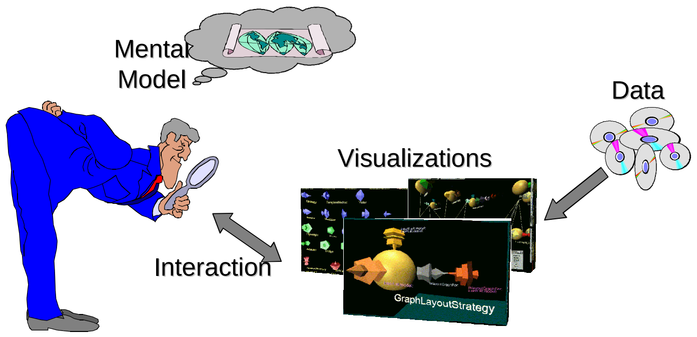


Figure 1: Visualization Process

### 3.1 3D Graphics

At the beginning most of the visualization systems display 2D graphics, but nowadays, more and more applications use 3D graphics in their visual presentations. Using this kind of presentations provides several advantages. The first and, perhaps the most clear one, is a greater information density than two-dimensional presentations as a consequence of a bigger physical space [Robertson et al., 1993]. Also, they help to have a clear perception of the relations between objects by integration of local with global views [Mackinlay et al., 1991] and by composition of multiples 2D views in a single 3D view [Koike, 1993]. Moreover their similitude with the real world enables us to represent it in a more natural way than 2D. This means that the representation of the objects can be done according to its associated real concept, the interactions can be more powerful and the animations can be even more real.

On the other hand, several problems arise, as intensive computation and more complex implementation than two-dimensional interfaces. These problems can be lighten using powerful and specialized hardware and several tools like 3D toolkits as JAVA3D [Sowizral et al., 1998] or 3D modeling languages such as VRML [ISO, 1997].

In general, 3D presentations should not be used in all visualizations, they should be used only when it is possible to take advantage of their benefits and avoid their weakness [Mullet et al., 1995]. 3D presentations are not essential, however a good utilization could be very helpful.

## 4 ReqViZ3D Approach

Our main objective is the visualization and animation of requirements to achieve a more effective requirements validation process. The approach proposes the use of visualization as well as formal specifications. Before describing the approach lets state what a validation means [Lano, 1996]: *"Validation of a description **D** against a description **C** means checking that D satisfies the properties specified in **C**, where **C** is the informal or semi-formal description."*

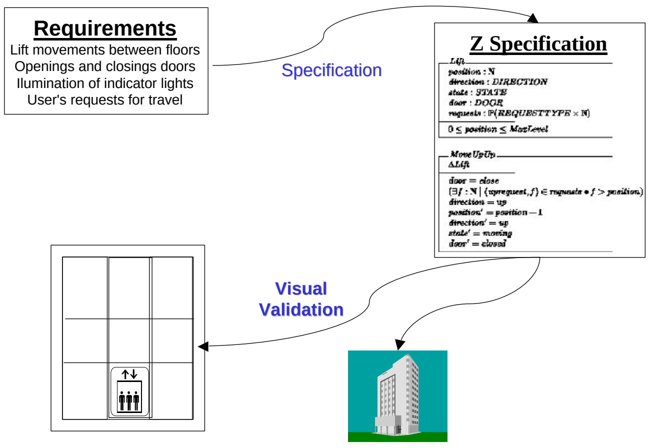


Figure 2: Requirements Validation Process

In the context of requirements validation, the check consist in ensuring that the specified system (**D**) is the system that the clients wants, where **C** is an informal set of the clients expectations. Figure 2 resumes the key ideas behind this project. First, we express requirements formally in Z. A formal specification makes it possible to unambiguously denote the meaning of the system requirements. After that, we define suitable graphic representations of the specification concepts and validate visually the specification with the user. Therefore, knowing that the requirements specification conforms to the user needs, it is a much more reliable base for developing the system.

The formal approach adopted can be classified as a light one, in the sense that no formal reasoning (theorem proving) is carried out to check if the properties of the specified system respond to the informal requirements and the emphasis is focused on the execution of the specification [Jones, 1996, Hörl and Aichernig, 2000]. Using formal methods in a lighter way is both a key to using them on large-scale applications and a way of penetrating fields outside the safety-critical area, where formal methods are mainly used and a detailed application can be justified because of the danger of loss of life [Jones, 1996].

We have decided to formalize requirements in Z. This is mainly because the experience gained in the past years from case studies has proven that a large variety of specifications problems may be successfully addressed in Z and set theory forms an adequate basis for building the more complex data structures which are needed in specifications [Potter et al., 1991]. However, it should be noted that also other formal specification languages could have been used.

In order to present an animated presentation, to validate requirements, the formal specification is executed. The execution of the specification allows the user to walk through a specification using different scenarios that are shown by visual presentations. The animation displays the behavior of the specified system and provides a means of dynamic testing. As a result of the approach:

- Misunderstandings between clients and developers are detected.
- New services arise and obscure ones are clarified.
- Inconsistencies in the specification are detected.
- The developed system is much closer to the needed system.
- The development effort is reduced.

In the next subsections each part of the approach is discussed in more detail: from system specification in Z, execution of specifications and visualization using a simple example.

## 4.1 Formal Specifications in Z

Z is a formal specification language that uses mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved.

The main construction in Z is the schema. A schema enables us to decompose a specification into small pieces. In Z, schemas are used to describe both **static** and **dynamic** aspects of the systems. The static aspects include the states it can occupy and the invariant relationships that are maintained as the system moves from state to state. On the other hand, the dynamic aspects include the operations that are possible, the relationship between their inputs and outputs and the changes of state that happen. In order to clarify this ideas a simple and widely discussed LIFT SYSTEM [Evans, 1994] example is presented:

*"A lift controller system has to service requests coming from the buttons placed on the floors of a building. The lift is moved by the controller in a direction satisfying the pending requests until no more requests are found; in this case the lift changes direction to service other new or pending requests. "*

First, we introduce a schema to describe the system state which corresponds to the static part of the system:

MaxLevel == 5
DIRECTION ::= up | down
STATE ::= moving | stopped
DOOR ::= open | closed

*Lift*

$position : \mathbb{N}$
$direction : DIRECTION$
$state : STATE$
$door : DOOR$
$requests : \mathbb{P}\,\mathbb{N}$

$0 \leq position \leq MaxLevel$
$state = moving \Rightarrow door = closed$

*InitLift*

$Lift'$

$position' = 0$
$direction' = up$
$state' = stopped$
$door' = open$
$requests' = \varnothing$

The lift can be defined by its position, direction, state and door state. The direction of the lift can be *up* or *down*, while the state indicates if the lift is *moving* or *stopped*. The lift door opens when the lift arrives at a floor and it is closed while the lift is *moving*. Possible requests are *up* or *down* requests. The invariant states that the movement of the lift is restricted to an interval of valid floors and asserts that while the lift is moving the door must be closed.

We can now start defining the system operations, that is the dynamic aspects of the system. *MakeRequests* schema adds a new request to the requests set. Operation schema *MoveUpUp* defines the operation of moving the lift up if up requests are present above the lift (also similar operations are defined for the other directions, not reported for conciseness):

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{MakeRequests}} \\ \hline
\Delta Lift \\
f? : \mathbb{N} \\ \hline
requests' = requests \cup \{f?\} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{OpenDoor}} \\ \hline
\Delta Lift \\ \hline
door = closed \\
(\exists f : \mathbb{N} \bullet f \in requests \land f = position) \\
state' = stopped \\
door' = open \\
requests' = requests \setminus \{F\} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{MoveUpUp}} \\ \hline
\Delta Lift \\ \hline
door = close \\
(\exists f : \mathbb{N} \mid f \in requests \bullet f > position) \\
direction = up \\
position' = position + 1 \\
direction' = up \\
state' = moving \\
door' = closed \\ \hline
\end{array}$$

In an operation schema we can identify a declaration part and a predicate part. The declaration part defines the inputs and the outputs of the operation as well as the system state schemas over which its operates. The declaration $\Delta Lift$ tells us that the schema is describing a state change in *Lift* schema. The declaration names ended in a question mark define the inputs of the schema. The part of the schema below the line, that is the predicate part, defines conditions that constrain the values declared in the declaration part.

## 4.2 Execution of formal specifications

Formal specification languages such as Z have been developed to precisely and concisely define the characteristics and specifications of a software system. However, formal specification languages fail in establishing a very important property for an immediate reflection of the consequences of the specifications and for an early validation: the executability of a specification [Fuchs, 1992].

Z was not conceived for execution, since its aim is to define abstract properties of the system being built and not the design decisions or the implementation details used in the system. Z specifications are declarative and the developer can declare non-computationally entities, such as infinite sets or non-computable functions and specify properties and operations on them.

Therefore, in order to execute a formal specification of Z, the notation of Z must be restricted to a subset almost directly executable. This means that Z is restricted forbidding the declaration of non-computationally entities and adapting it to the capacities of executables languages that, on the other side, are less expressive that non-executables ones, because their functions must be computable and their domains must be finite.

At this time several problems arise that must be faced according to the chosen method of translation and the target language. In general, most problems derive from trying to match different levels of abstraction. Any acceptable solution has to balance declarativeness versus efficiency in the sense that we want not only an executable form of a very high-level specification, but also a reasonable efficient execution to test the specification [Breuer and Bowen, 1994].

Due to the mathematical and logical foundations of the formal languages the declarative or functional languages seem to be the most suitable ones. For example, a straightforward way to animate Z documents seems to be the mapping of Z specifications into PROLOG as practice shows that most predicates found in Z documents have an easy implementation in terms of PROLOG clauses. A logic programming language is a very interesting choice for translating a specification language as Z which is based on first order logic. The conceptual gap between a logic programming language (which is a subset of a first order logic) and an specification based on logic is significatively less than a specification based on logic and an imperative language.

In fact, it is possible to take a subset of Z for generating PROLOG code. This point of view is compatible with the assertion that a considerable part of Z has executable semantics [Breuer and Bowen, 1994]. In particular, the approach we adopted is similar to the approach proposed by Sterling [Sterling et al., 1996]:

- The semantics of a subset of Z on which the transformation is based, is clear.
- The transformation of the subset is almost direct.
- The expressivity of the subset is powerfull enough for many applications.

Each of these assertions demands a deeper discussion. However, as the focus is the applications of logic programming in software engineering, these assertions will not be discussed, showing instead, which one is the subset and how the transformation can be carried out by using an example. In the next sections the process for translating Z schemas to PROLOG is explained.

### 4.2.1 State schemas

A state schema consist of declarations of variables and predicates that constraint their values. In order to translate a state schema, a PROLOG clause will be created whose name is the same of the schema, the arguments of the clause will be the state variables and the invariant of the schema as the body of the clause. An additional argument is also added for storing global declarations of the specification:

**Schema(GlobalDec,StateVar1,...,StateVarN):-invariant.**

For example, figure 3 shows the translation of the *Lift* schema. The clause *getContain*, which is used to access the values of global declarations, enable us to obtain the value of the constant *Max_Level*.

### 4.2.2 Operation Schemas

An operation schema models a state transition relating the values of the variables before the execution of the operation with the values after the execution. Also, it defines inputs and outputs for the operation. For translating an operation schema a PROLOG clause is created whose name is the same of the schema and the inputs and outputs of the operations as the arguments of the clause. Also, two additional parameters are needed (PROLOG structures) for holding the state of the system before and after

*Lift*

$position : \mathbb{N}$
$direction : DIRECTION$
$state : STATE$
$door : DOOR$
$requests : \mathbb{P}\,\mathbb{N}$

$0 \leq position \leq MaxLevel$
$state = moving \Rightarrow door = closed$

$\Rightarrow$

***Prolog***

```
z_Lift( Z_GState,Z_position,Z_direction,Z_state,Z_door,Z_requests ):-
0<=Z_position,
getContain(Z_StateG,'Root','Z_MaxLevel',Z_MaxLevel),
Z_position<=Z_MaxLevel,
implies(state = moving, door = closed).
```

Figure 3: Translation of a State Schema

the execution of the operation. Other two parameters are included for maintaining global declarations and logging and tracing the execution of the operations (information used to animate visualizations). Finally the body of the clause will be the assertions of the Z schema, that is, pre and post conditions of the schema operations. Also, after the execution of an operation schema the invariant must still remain true, so in order to verify that fact a call to the clause of the state schema is needed.

The translation procedure for operation schemas is the following:

**Schema( globalState(GlobalDec,GlobalDec'),state(before),state(after),Arg1,...,ArgN):-**
**predicates,**
**call to invariant.**

***Prolog***

*MakeRequests*

$\Delta Lift$
$f? : \mathbb{N}$

$requests' = requests \cup \{f?\}$

$\Rightarrow$

```
z_MakeRequests(globalState(Z_GState,Z_GNewState),
   z_Lift(Z_position,Z_direction,Z_state,Z_door,Z_requests),
   z_Lift(Z_position,Z_direction,Z_state,Z_door,Z_requests'),Z_f):-
uni(Z_requests,[Z_f],Z_requests'),
z_Lift(Z_GState,Z_position,Z_direction,Z_state,Z_door,Z_requests'),
addChangeOp(Z_GState,z_MakeRequests,Z_GNewState).
```

Figure 4: Translation of an Operation Schema

Figure 4 illustrates this translation procedure using the schema *MakeRequests* of the previous example. The operation *addChangeOp* registers in the global state that the operation was executed.

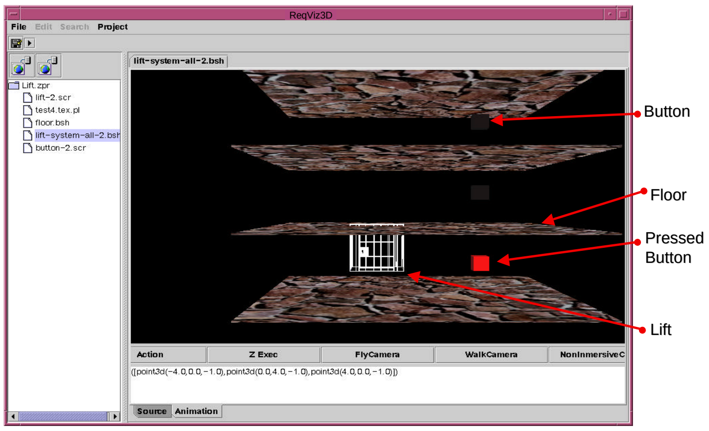
## 4.3 Visualization

Once the system is specified in Z the developer defines the graphical representation of the requirements for visualizing and animating the specification concepts in a 3D world, as figure 5 shows, and so validated by the users. In this example the user can press the buttons of each floor an see how the lift services user requests to go up or down. When the button is pressed is lighted on and when the lift services the request is lighted off. For building the visualization REQVIZ3D provides a graphical specification language to define the geometry and behavior of the 3D graphics objects. An object specification is composed by three main parts: *geometry definition*, *actions* and *recognized events*. For example, the next script defines the lift graphic object.

%—————-geometry———————
geometry([def(open,file('models/DBLDOORO.3DS')),def(closed,file('models/DBLDOORC.3DS'))]).

This section defines the different shapes that can be used to present a graphic object. In particular in this example, defines all the alternative geometries of the lift, that is one when the lift is closed and another when it is opened. Also each geometry can be named in order to be identified and accessed.

The next section defines the behavior of the lift (*open*, *close* and *goto*). These actions are defined in terms of a set of predefined actions (translate, move,..). For example the action *open* call the *switch* action in order to show a graphic of the lift opened.



```
([point3d(-4.0,0.0,-1.0),point3d(0.0,4.0,-1.0),point3d(4.0,0.0,-1.0)])
```

Figure 5: Lift System Visualization

%——————actions————————-
action(open, [ switch(open) ]).
action(close,[ switch(closed) ]).
action(goto(Floor,From),[ call(Time is abs((From-Floor)*1000.0)), moveTo(time(0,Time),[point3d(0.0,Floor,0.0)])]).

Finally the events section defines the reactions of the lift in response to changes in the execution of a Z specification using a change-propagation mechanism that ensures consistency between the specifications and visualizations based on implicit invocation [Krasner and Pope, 1988]. The mechanism maintains a registry of the dependent components within the specification. Changes in the state of the model trigger events that are propagated to the visualizations. Using this mechanism the Z animator announces different events (as table shows) about the state of the execution of a specification.

| **Event** | **Description** |
|---|---|
| *StartOperation* | Begin of the execution of an operation |
| *EndOperation* | End of the execution of an operation. |
| *FailOperation* | The execution of an operation failed. |
| *StateChanged* | The state of the system changed. |

The first event that is propagated when an operation is executed is the *start operation* event. It may be possible that the execution of the operation fails, so the *fail operation* event is announced. In contrast, if the operation is successfully executed, an *end operation* event is propagated. At last, if the operation changed the state of the system an *state changed* event is triggered. For example, when the lift is closed the presentation must be updated (switch to *closed* geometry).

%——————-events——————————-
event(stateChanged,goto(X),[value('position',Pos),oldValue('position',OPos),goto(Pos,OPos)]).
event(stateChanged,openDoor,[open]).
event(stateChanged,closeDoor,[close]).

## 5 Examples

This section presents a brief description of other two examples developed with ReqViz3D as figure 6 shows. The examples developed were a vending machine and an automatic teller machine. The first example is about a vending machine that sells cans. A client inserts coins in the coin slot.

After that, if the required amount of money was inserted, the client obtains a can by pressing the eject button. The machine can expend a limited number of cans.

The second example is an automatic teller machine which provides these basic services: deposit, withdraw, transfer, balance and user authentication. In order to use the ATM machine the user inserts its card and is prompted for a password. The password is validated and if it is correct the client can make different transactions selecting operations by touching the screen buttons and entering values using the keyboard. The user receives a ticket for each operation and, in the case of extracting money, takes it.

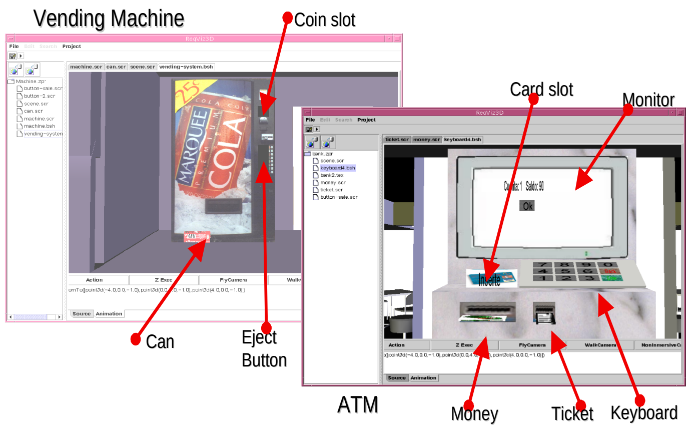


Figure 6: Examples

## 6 ReqViZ3D Tool

In essence, this tool takes a specification as input and generates a visualization as output, through which users can validate requirements. REQVIZ3D was developed in JAVA. In order to animate a Z specification it is translated to PROLOG and executed. As we developed REQVIZ3D in JAVA, a way to integrate JAVA and PROLOG was needed. This integration was done using JAVALOG [Amandi et al., 1999]. JAVALOG is a PROLOG interpreter written in JAVA designed to allow easy integration between JAVA and PROLOG mixing Logic/OO paradigms. Also, trying to take advantage of 3D visualizations we developed the View subsystem on the top of JAVA3D. Figure 7 presents a global system view of REQVIZ3D that defines a blueprint of the overall structure of the application and corresponds to the architectural model *Model-View-Controller*. This model prescribes the division of an interactive application in three parts, the *Model* that represents the application functionality, the *View* responsible for the output interface and the *Controller* responsible for the input handling.

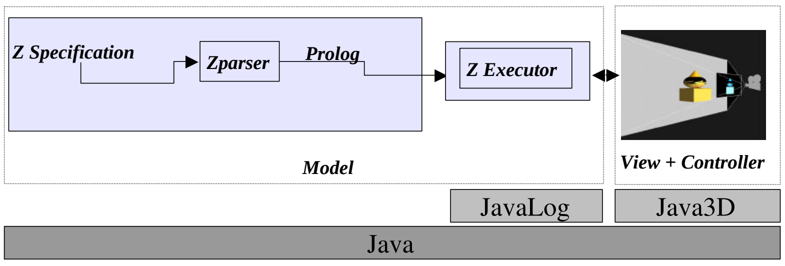


Figure 7: Global System View

## 7 Conclusions and Future Extensions

The main contribution of this work is the utilization of visualization techniques to reduce the communication gap between the costumer and the developer: *the system is described in a way that users can understand*. As a consequence of validating requirements in the earlier stages of the development process the total effort to develop a system is reduced.

Also a prototype tool to visualize requirements was developed. This tool assist the developer in several stages in the development process: from requirements specification in Z and definition of graphical objects, to animation and execution of requirements in a 3D world.

Three dimensional graphics were used in the construction of the visualizations. Their similitude with the real world enables us to represent it in a more natural way than 2D. This means that the representation of the objects can be done according to its associated real concept. However, the construction of 3D graphics presentations is a difficult and time consuming task, besides it requires specific knowledge and even artistic skills. Therefore a future extension is the automatization of the presentation extending the ideas presented in several works about the automatic generation of presentations [Mackinlay, 1986, Zhou, 1998]. Several systems were presented, as an example, showing that the use of visualization techniques were very useful in analyzing the dynamics of them.

At last, the work combines an informal approach (visualization) with a formal light one, resulting in a more effective technique. In that sense, a light application of formal methods can be an economical way to improve the quality of specifications without using formal proofs.

Others future extensions include supporting also OBJECTZ as specification language, provide a basic library of 3D graphics components and develop new examples about software architectures.

## References

[Amandi et al., 1999] Amandi, A., Zunino, A., and Iturregui, R. (1999). Multi-paradigm languages supporting multi-agent development. In *MAAMAW'99*, pages 128–139.

[Breuer and Bowen, 1994] Breuer, P. and Bowen, J. (1994). Towards Correct Executable Semantics for Z. In *Proc. 8th Z Users Workshop (ZUM)*, pages 185–212. Springer-Verlag.

[Card et al., 1998] Card, S., MacKinlay, J., and Shneiderman, B., editors (1998). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers.

[Evans, 1994] Evans, A. S. (1994). Specifying and verifying concurrent systems using Z. *Lecture Notes in Computer Science*, 873.

[Fuchs, 1992] Fuchs, N. (1992). Specifications are (preferably) executable. *IEEE Software Engineering Journal*, 7(5):323–334.

[Gonzalez and Urban, 1991] Gonzalez, J. P. D. and Urban, J. E. (1991). Language aspects of ENVISAGER. an object-oriented environment for the specification of real-time systems. *Computer Languages*, 16(1):19–37.

[Hazel et al., 1997] Hazel, D., Strooper, P., and Traynor, O. (1997). Possum: An animator for the SUM specification language. In *Proceedings: 4th Asia-Pacific Software Engineering and International Computer Science Conference*, pages 42–51. IEEE Computer Society Press.

[Hörl and Aichernig, 2000] Hörl, J. and Aichernig, B. K. (2000). Validating voice communication requirements using lightweight formal methods. *IEEE Software*, 17(3):21–27.

[ISO, 1997] ISO (1997). Vrml97, international specification. Technical report, ISO.

[Jackson and Wing, 1996] Jackson, D. and Wing, J. (1996). Lightweight Formal Methods. *IEEE Computer*, 29(4):22–23.

[Jones, 1990] Jones, C. B. (1990). *Systematic Software Development Using VDM*. Prentice-Hall International, Englewood Cliffs, New Jersey, second edition. ISBN 0-13-880733-7.

[Jones, 1996] Jones, C. B. (1996). Formal methods light: A rigorous approach to formal methods. *Computer*, 29(4):20–21.

[Kelly et al., 1992] Kelly, J. C., Sherif, J. S., and Hops, J. (1992). An analysis of defect densities found during software inspections. *The Journal of Systems and Software*, 17(2):111–117.

[Koike, 1993] Koike, H. (1993). The role of another spatial dimension in software visualization. *ACM Transactions on Information Systems*, 11(3):266–286.

[Krasner and Pope, 1988] Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of OOP*, 1(3).

[Lano, 1996] Lano, K. (1996). *The B Language and Method: A guide to Practical Formal Development*. Springer Verlag London Ltd.

[Mackinlay, 1986] Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141.

[Mackinlay et al., 1991] Mackinlay, J. D., Robertson, G. G., and Card, S. (1991). The perspective wall: Detail and context smoothly integrated. In *Proceedings of ACM CHI'91*, pages 173–179.

[Meyer, 1995] Meyer, B. (1995). On formalism in specifications. *IEEE Software*, 2(1):6–26.

[Mullet et al., 1995] Mullet, K., Schiano, D. L., Robertson, G., Tesler, J., Tversky, B., Mullet, K., and Schiano, D. J. (1995). 3d or not 3d: More is better or less is more? In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Panels*, pages 174–175.

[Ozcan et al., 1998] Ozcan, M. B., Parry, P. W., Morrey, I. C., and Siddiqi, J. I. (1998). Visualisation of executable formal specifications for user validation. *Lecture Notes in Computer Science*, 1385.

[Parry et al., 1998] Parry, P., Ozcan, M., and Siddiqi, J. (1998). The application of visualization to requirements eng. Technical report, Computing Research Centre, Sheffield Hallam University.

[Potter et al., 1991] Potter, B., Sinclair, J., and Till, D. (1991). *An Introduction to Formal Specification and Z*. Prentice Hall, New York.

[Potts, 1991] Potts, C. (1991). Expediency and appropriate technology: An agenda for requirements engineering research in the 1990s. *Lecture Notes in Computer Science*, 550.

[Pulli et al., 1991] Pulli, P., Elmstrom, R., Leon, G., and de la Puente (1991). IPTES - incremental prototyping technology for embedded real-time systems. Technical report, ESPRIT.

[Robertson et al., 1993] Robertson, G., Card, S. K., and Mackinlay, J. D. (1993). Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71.

[Sowizral et al., 1998] Sowizral, H., Rushforth, K., and Deering, M. (1998). *The Java 3D API Specification*. Addison-Wesley.

[Spivey, 1988] Spivey, J. M. (1988). *Understanding Z*. Cambridge Tracts in Theoretical Computer Science 3. Cambridge University Press. ISBN 0-521-33429-2.

[Sterling et al., 1996] Sterling, L., Ciancarini, P., and Turnidge, T. (1996). On the Animation of Not Executable Specifications by Prolog. *Int. Journal on SE and KE*, 6(1):63–88.

[Zhou, 1998] Zhou, M. X. (1998). Automated visual discourse synthesis: Coherence, versatility, and interactivity. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems (Summary)*, volume 2 of *Doctoral Consortium*, pages 76–77.