Guidelines to Formal System Studies

by J.-R. Abrial

November 2000

Draft Version 2

Guidelines to Formal System Studies

J.-R. Abrial

Consultant

While the problematics of the *development process* for software system has now acquired a certainly stability in form, the same thing cannot be said about "System Studies". This document is intended to present a, so-called, *formal approach* to such studies and to elaborate a number of *guidelines* aimed at putting it into practice[1].

It is made of three parts. In the first one, the approach is presented in a rather informal way, while the second one contains some precision concerning its more formal setting. In the third part, we indicate how this approach could be put into practice with the present version (3.5) of **Atelier B**.

Informal Presentation

System Studies

Under this term we usually set, in a relatively casual way, all the reflexions and decisions situated *before* the drawing up of the specification documents of the individual *parts* of a *complete* (and complex) system. These parts concern the various, possibly distributed, components of the software (if any), the physical equipment and, finally, the necessary communication between them.

These system studies are thus mainly aimed at elaborating the *architectural decomposition* of a system. They are also aimed at showing that the system being considered is *viable*. Besides, they can end up in the determination of a certain number of fundamental parameters fixing the main *dimensions* of the system.

The Classical Approach

Most of the time, in industry, such studies are made by very competent domain-expert people, whose main source of inspiration simply comes from the *previous system* they have made (its fine points as well as its weaknesses).

Such studies are sometimes "validated" by means of a number of *simulations*, consisting in building some behavioral models of the system in project, simulations which are then executed under the form of various monitored sessions. The results of the monitoring are analyzed in order to determine whether the system (or rather its model) works according to certain predefined criteria.

[1] Many of the ideas in this document have been inspired by various work done around *Action Systems*.

Rationale for a Formal Approach

As a complement to the above approach, our thesis is that these system studies can be greatly enhanced by using the main ingredients that have proved to be quite successful in software specification and design, namely: formal (mathematical) description, refinement techniques, decomposition, and mechanized proofs.

The reason for the above belief comes from the understanding that there is no fundamental distinctions to be made between the various *discrete automatons* that compose a system. Whatever its final implementation (as a piece of physical equipment, or a piece of software, or as a chip, etc), each component reacts to some outside impulses, is somehow connected to some others, is plunged within flows of data, and eventually executes a pre-defined "program", etc. Globally, they together form a vast entity whose dynamic behavior is *distributed* among various communicating parts.

The Complexity Gap

The overall rational study of such complex entities seems at first glance to be absolutely unfeasable: we do not know which string has to be pulled first in order to understand what is going on. More seriously, we do not know even how to characterize what the system "does". And the very initial requirements documents that are at one's disposal are not very helpful in that matter because they, quite often, correspond to some mental implementations. They should rather give the general properties of the intended system.

In software development, we had sometimes a notion of "input" (and thus of pre-condition) and one of "output" (and thus of post-condition), notions that helped us defining the first handle on which a formal development could be elaborated.

Here the situation is quite different in that we have a completely *closed system* containing a large number of concurrent components, each of them being subjected to many *transitions* occurring, most of the time, asynchronously. The need for new methods for studying systems at this level is badly needed: the use of informal approaches is clearly not sufficient.

Mathematical Simulation

The main idea of the proposed approach consists in studying our future system by means of one (or better several) *model*, on which we intend to apply certain correctness criteria. In this respect, this approach does not seem to be very different from the simulation technique which was alluded above.

As a matter of fact, the difference is not so much in the modelling itself than in the way we are going to exploit it. The criteria might be the same, but rather than analyzing the results of a number of simulation sessions and conclude that they confirm the selected criteria, we are going to *prove directly* on the model that it is indeed the case.

This change in the mode of exploitation of the "simulation" has a number of interesting consequences:

1. Rather than using a simulation language in order to build our model under the form of a simulation program, we are going to use directly a *mathematical notation*, which will

allow us to represent the model in a way that will be more convenient than a simulation program (for expressing the statements to prove and to prove them).

2. Rather than limiting ourselves to a single simulation program, as is usually the case, we can very well use a series of embedded models that are supposed to be *refinements* of each others. In this way, the various criteria to prove can be accumulated and proved immediately at their right level of abstraction: *the proof process accompany the model elaboration process*.

3. Once a model has reached a dangerous level of complexity (so that the proofs might become cumbersome), it will be possible to *decompose* it into separate models: the architecture is born.

Observations

A series of questions that might come to mind at this point concerns the notion of model: What is a model ? How to build one ? etc. Everyone has a vague notion of what a model is, but this is not sufficient to undertake a formal construction: we clearly need to give to this concept a very precise definition.

For building the model (at a certain level of abstraction) of a future system, the idea is to place oneself mentally *above* the system in question and try to imagine what we could *observe* from there.

Such observations might very well be quite “physical”: for instance, we could observe trains moving on a certain network, and sometimes people going in and out of them. A careful observation could lead to some laws that seems to be obeyed: trains never touch each other while in movement, people only go in or out of trains while these are stopped, etc.

State and Events

These observations are essentially made by means of two kind of notions: a notion of objects (trains, people, etc) and a notion of movement (of trains, of people, etc). The former constitutes the *state* of the model (this is a static notion), while the latter constitutes the *events* (this is a dynamic notion) that may occur and that we are able to observe.

The properties of the state are the, so-called, *safety* properties, whereas the properties of the events are the, so-called, *liveness* properties. Proofs might be performed to validate these two kinds of properties, which have no reason, a priori, to be coherent.

Notice that the physical objects that are represented in our model at this level corresponds to the observation of the reality : they are perfectly well defined objetcs although probably not with the accuracy we might have in further (refined) models.

The Parachute Paradigm

Once you have made some observations and correponding proofs at a certain level “above” the system, you might envisage to step down a little and thus be able to observe other interesting things: that is, a more precise state and more events, which you were not able to distinguish before because you were too high in the sky.

For instance, one might observe some traffic lights along the track as well as doors in the trains, and of course also some new events like the opening and closing of doors or the changes in the status of the traffic lights. Finally, new laws governing now trains, people, traffic lights and doors can be set up. These new observations should certainly not invalidate the previous ones, again they should rather make them *more accurate*.

From this level, you might step down one more time, and so on: this is the so-called *parachute paradigm*. What we have just briefly explained here is the way we can *refine* our models in order to have a gradually more precise view of all the facets of the system. What is important to stress at this point is that all the properties that are proved at a certain level *remain valid* along the path leading gradually to the ground.

From the Physical State to the Information and Communication State

By going further down, we start to observe things that are no so much physical in nature as was the case when we were very high up in the sky: we are now able to observe the behavior of some particular objects called *data*.

This is because we start investigating some parts of the system that are supposed to "intelligently" *control* others. Clearly, such parts were not "visible" at the beginning of the modelling process. In order to play their rôle, such control parts have to *memorize* a certain synthesis of the overall physical situation. This memory forms the, so-called, *information* part of the state.

At a given moment, such a memorized picture reflects a physical reality that is not exactly the same as the one that takes place in the physical world at that very moment: the "reality", which is represented in the memory, belongs to a certain (usually close) past. This is due to the fact that the composition of the memorized picture is realized with the help of a number of *sensors* that are supposed to make some physical measurements, which are not instantaneous. Moreover, once determined, the information should be transmitted to the control parts through some communication *channels*, thus adding an extra delay between the physical situation and its reflected picture. Such channels carry the messages forming yet another part of the state, the *communication* part.

Conversely, the control parts of the system do elaborate some decisions aimed at effectively controlling the physical reality. Such decision are thus supposed to be feedbacked onto the physical parts of the system by means of symetric communication channels and eventually some *actuators*.

We are here at the heart of the system studies. Of course the overall construction we have just sketched in very general terms has to be built very gradually by means of a series of new refinement steps. At each such step, we have to prove that the control and communication parts in construction ensure the preservation of the laws, only elaborated in terms of the physical state at the very early stages of the developmement, despite the fact that these control parts can only "reason", as mentioned above, on a fuzzy picture of the reality.

Notice that all this is elaborated by using the same basic ingredients of our models, namely states and events. Also notice that, from the point of view of the modelling technique, no special distinction has to be made between those parts of the state that are *physical*, those that deal with the controling *data*, and finally those dealing with the *sensors*, the *channels* and the *actuators*.

Decomposition: Building the Architecture

In the previous section, we have extended the physical model, elaborated at the early stages of the development, with some control and communication parts. But, we still have a single model. Clearly, that model may now be quite large and thus difficult to develop further: it may have a state made of many parts "activated" by many events.

This is the right moment to envisage *decomposing* the model into several sub-models. A "natural" decomposition is clearly one where we have a sub-model for the physical and communication parts and one for the control part. But this is only one among many other possible decompositions: we could very well have several control parts as well as several physical parts comunicating with each other. It could also very well be envisaged to have several control parts forming separate models communicating with each other through some communication channels (this is just a distributed *network*).

For instance, in a train system, we could have some control parts replicated and embarked within trains and other control parts replicated and installed along the track, and perhaps, yet another control part at the extremity of some line, etc.

The rôle of the decomposition is clear. Once separated from the main body, a sub-model can be developped further independantly from the rest of the system, which become its, so-called, *environment*. Notice that this notion of is essentially *relative*: each sub-model (or group of sub-models) being the environement of others and vice-versa.

What is important to notice here is that we clearly favor a *decomposition* process over a *composition* one. This does not mean that we cannot compose existing parts: it only means that such a composition has to be made *under the supervision of a decomposition process*. This is the only way that ensures that some global laws are maintained while the composing parts are working together. Composing components in the absence of a decomposition process may lead to the formation of some unexpected behaviors of an individual component *due to the presence of others*.

Deamons

An important outcome of this global event-driven approach is that it allows us to easily take into account the unexpected arrival of all sorts of "bad" things such as failures, forgery of information contained in communication messages, long delays, etc.

Formalizing this is very easy by means of asynchronous events occurring spontaneously. These are the, so-called *deamons*. We are then in a good position to formally *prove* that the system works "correctly" in spite of them. And thus guarantee that it is able to resist to such *attacks* that may occur at any time.

Conclusion: Summary of the Guidelines

Here is a summary of the guidelines that have been described in the previous sections:

1. Start with a very simple physical model (define the state and the events).
2. Make this physical more accurate model by means of a series of refinement steps.
3. Introduce some of the control parts together with some communication parts.
4. Loop on points 2 and 3.
5. Decompose the model into sub-models.
6. Loop on steps 2, 3,4, and 5 for each sub-model.
7. At each step, feel free to introduce attacking deamons.
8. At each step, don't forget to do the corresponding formal proofs.

Formal Presentation

Mathematical Model

The formal part of a System Study is made up of a series of *mathematical models*. Each model is supposed to *refine* the previous one. This series demonstrates the progressive taking into account of the *global system* which we want to study.

The State

A model is first presented by means of a certain number of *constants* and *variables*. In practical terms, these constants and variables mainly show simple mathematical objects: sets, binary relations, functions, etc. Moreover, they are constrained by some conditions expressing the *invariant properties* of the model.

These variables and constants, linked by the invariant properties in question, describe the *state* of the dynamic system which is to be analyzed. They can represent "data" of very diverse nature. For example, they can have a certain counterpart in future software, but they can also correspond to the formalization of certain physical equipment, or even to messages transiting via a communication network.

The Events

Besides its state, a model contains a number of *events* which show the way it may evolve. These events are only supposed to be *observable*, they are in no way actions that can be "called". Indeed, we are not describing the realization here, all be it abstract, of a system. We are rather making a *mathematical simulation*, which allows us to reason about the future system we are going to construct. This reasoning is precisely what is going to allow us to analyze very early on the behaviour of our future system and to draw up a possible *architecture*.

Each event is composed of a *guard* and an *action*. The guard is the necessary condition under which the event may occur. In other words, once its guard hold, the occurrence of the event may be observed at any time (but it may also never be observed). As soon as the guard does not hold however, the event cannot be observed. The action, as its name indicates, determines the way in which the state variables are going to evolve when the event does occur.

It is possible for several events to have their guards held simultaneously. From this point of view, the model present a certain *external non-determinism*. Notice that in case several guards hold simultaneously, no two event cannot be observed to occur "together": events are *atomic*.

Lastly, we must observe that the events are, a priori, *asynchronous*. Possible synchronisms are only the consequence of the actions of some of them on the guards of others.

Practical Form of an Event

Practically speaking, an event is presented in the following form:

```
xxx  $\widehat{=}$
  ANY $x, y, \ldots$ WHERE
    $P(x, y, \ldots, v, w, \ldots)$
  THEN
    $S(x, y, \ldots, v, w, \ldots)$
  END
```

where the identifiers have the following signification:

- xxx is the name of an event,
- $x, y, \ldots$ denotes a certain number of variables local to the event,
- $v, w, \ldots$ denotes a certain number of state variables or constants of the model,
- $P(x, y, \ldots, v, w, \ldots)$ denotes a predicate,
- $S(x, y, \ldots, v, w, \ldots)$ denotes the action associated to the event.

In this case, the guard of the event corresponds to the following existential predicate:

$$\exists\,(x, y, \ldots) \cdot P(x, y, \ldots, v, w, \ldots)$$

In other words, the necessary (but insufficient) condition for the event xxx to take place with the current value of the state variables or constants $v, w, \ldots$ of the model, is that it be possible to assign to the local variables $x, y, \ldots$, of the event xxx, some values making the predicate $P(x, y, \ldots, v, w, \ldots)$ true. As can be seen xxx presents a certain latitude in the choice of possible values for the local variables $x, y, \ldots$. We can speak here about *internal non-determinism*.

The action presents itself in the form of the simultaneous assignment of certain state variables $a, b, \ldots$ to certain expressions $E, F, \ldots$ depending upon the state of the system and the local variables of the event (it is to be noted that those variables which are not mentioned in the list $a, b, \ldots$ do not change):

$$a, b, \ldots := E, F, \ldots$$

Sometimes, the event can have the simpler following form:

```
xxx  $\widehat{=}$
  SELECT
    $P(v, w, \ldots)$
  THEN
    $S(v, w, \ldots)$
  END
```

In this case, there are no variables local to the event, and the guard just correponds to a condition holding on the state variables of the model.

Consistency: Preservation of the Invariant

Once a model is built, one must prove that it is *consistent*. One has to prove the preservation of the invariant by each event of the model. More precisely, it must be proved that the action associated to each event modifies the state variables in such a way that the corresponding new invariant holds under the hypothesis of the former invariant and of the guard of the event. For a model with state variable v and invariant $I(v)$, and an event of the form:

```
ANY $x$ WHERE
   $P(x, v)$
THEN
   $v := E(x, v)$
END
```

the statement to be proved is thus

$$\boxed{I(v) \;\wedge\; P(x,v) \;\;\Rightarrow\;\; I(E(x,v))}$$

Model Refinement

The model structure we have described above is simple enough. However, it would be impossible to formalize a complete *real* system by means of such a single model because the state would be far too complicated and the number of events far too large. In order to master the modelization process, we shall use two complementary techniques: first the refinement, second the decomposition. In this section, we deal with refinement.

Refining a model consists in refining its state and its events.

A concrete model (with regards to a more abstract one) has got a state that should be related to that of the abstraction through a, so-called, *gluing invariant*. Sometimes the concrete state is a simple extension of the abstract one so that the abstraction relation is then just a projection. But in general the abstraction relation can be any relation that should however be defined on all the concrete states.

Each event of the abstract model is refined into a corresponding event of the concrete one. Informally speaking , a concrete event is said to refine its abstraction (1) when the guard of the former is stronger than that of the latter (guard strenghtening), (2) and when the gluing invariant is preserved by the conjoined action of both events.

Another frequent way of refining an event system consist in *adding new events*. This corresponds to observing the system with *a finer granularity* than in the abstraction. Such new events have an implicit (hidden) counterpart in the abstraction, namely, the event that does nothing.

The new events that are introduced at some level must obey a specific constraint: they must not monopoly the "control" for ever. In other words, they should not have the possibility to be fired indefinitely without letting the old events be executed from time to time. Practically, this means that, should the control be given exclusively (guards permitting) to the new events, then, at some point, the disjunction of their guards should become false.

A global constraint of a refined model with regards to its abstraction deals with deadlock. Since normally a system should run for ever, then so must it be the case for its refinement. But more generally, we shall state that a refined model *should not deadlock more frequently* than its corresponding abstraction.

Proofs of Correct Refinement

Suppose we have an abstract model with state v and invariant $I(v)$, and also a corresponding concrete model with state w and gluing invariant $J(v, w)$. If an abstract and corresponding concrete events are as follows:

```
ANY $x$ WHERE
   $P(x, v)$
THEN
   $v := E(x, v)$
END
```

```
ANY $y$ WHERE
   $Q(y, w)$
THEN
   $w := F(y, w)$
END
```

then the statement to prove is

$$\boxed{I(v) \ \wedge \ J(v, w) \ \wedge \ Q(y, w) \ \ \Rightarrow \ \ \exists x \cdot (\, P(x, v) \ \wedge \ J(E(x, v), F(y, w)) \,)}$$

This states that for each possible choice of the local variables of the concrete event there is a choice of the local variables of the abstraction that makes the gluing invariant (as modified by *both* events) true: indeed, the concrete event refines its abstraction. As can be seen the concrete guard is stronger than its abstract counterpart. In case of a *new* evnt of the form

```
ANY $y$ WHERE
   $Q(y, w)$
THEN
   $w := F(y, w)$
END
```

the statement to prove is simpler since that new event is only suppose to refine the non-event that does nothing. Formally

$$\boxed{I(v) \ \wedge \ J(v, w) \ \wedge \ Q(y, w) \ \ \Rightarrow \ \ J(v, F(y, w))}$$

Proofs of the Impossibility of Monopoly of New Events

Given a new event of the form

```
ANY $y$ WHERE
    $Q(y, w)$
THEN
    $w := F(y, w)$
END
```

we must prove that, under the invariant, the event decreases a certain (natural number) variant expression $V(w)$ that has to be exhibited. Notice that this variant expression must be the *same* for all the new events. This decreasing is thus a global property of the new events. This is because we want to prevent their entire population to take control for ever. Should the variant be distinct for, say, two events, then they could very well stop individually after some steps, but the other one could then take control in an endless ping-pong: this is clearly something we want to avoid. Formally

$$\boxed{I(v) \ \wedge \ J(v, w) \ \wedge \ Q(y, w) \quad \Rightarrow \quad V(F(y, w)) < V(w)}$$

Proofs of the Limitation of Deadlocks

For each abstract event of the form:

```
ANY $x$ WHERE
    $P(x, v)$
THEN
    $v := E(x, v)$
END
```

and for an abstract invariant $I(v)$ and a gluing invariant $J(v, w)$, the following statement must be proved:

$$\boxed{I(v) \ \wedge \ J(v, w) \ \wedge \ P(x, v) \quad \Rightarrow \quad \text{disjunction of the concrete guards}}$$

where the right hand side of the implication denotes the disjunction of the concrete guards. In other words, whenever an abstract event can be fired then it must also be the case for a concrete event (not necessary however the corresponding concrete event, it could be one of the new events). Notice that the guarding predicate $P(x, v)$ that is on the left-hand side of the implication now corresponds to an abstract event. This contrasts with all the previous proof statements.

Practice with Atelier B

The **B** language and, a fortiori **Atelier B**, does not support directly all the practicalities of the formal approach we have just presented. It is possible however to fully *simulate* them and thus exactly obtain the corresponding proof obligations.

The purpose of this section is to explain how this can be done. As **Atelier B** already provides all the proof obligations for the maintenance of invariants and for ensuring correct refinements, the two main points we are going to deal with here are (1) the handling of the new events (together with the specific proof obligation associated with them) and (2) the limitation of deadlock.

The Handling of Events

Events are encoded under the OPERATIONS clause as "normal" operations having no pre-conditions and no input or output parameters. They are thus introduced either by means of the key-word BEGIN when the guard is missing or the construct SELECT ... THEN when there is a simple guard, or finally with the construct ANY ... WHERE ... THEN in case of a quantified guard.

As it is not possible, at the present time, to introduce any new operations in a refinement, the new events introduced at some stage must already exist in the previous abstraction. And since a new event introduced at a certain level is supposed to refine skip, its abstraction is thus simply defined with the "body" skip. The burden is that such "dummies" should be present in all previous abstractions, and thus, in principle, known right from the beginning of the development. Practically, the situation is not so bad since adding (on the fly) these skip events to an abstraction does not cause any further proofs: the only price to pay is then just to add these dummy events, re-type-check and re-generate the proof obligations (the merging with the previous proof status will thus determine that no new proofs are necessary). You can press a siggle key, the "prove" key, and the type-checking and proof obligation generation will be performed automatically.

As explained above, each event introduced at some stage must be proved to decrease a certain variant expression (which is usually a natural number expression V). We remind the reader that this expression is the same for all new events introduced in a given refinement. The corresponding proof obligation would have been very easy to generate should we have the possibility to associate, within the **B** language, a certain *post-condition* to an operation. Suppose for a moment that it is possible by means of the following construct (easily exended to the other forms of events):

```
xxx  ≙
   ANY  y  WHERE
     Q
   THEN
     S
   POST
     P
   END
```

Here the post condition P stipulate that by the end of the "execution" of the event the predicate P should hold. Notice that P might contain some state variables, say w, denoting the value of the corresponding variable *after* the execution, but also some variables of the conventional form $w\$0$ denoting the value of that same variable w *before* the execution (this is a standard convention in **Atelier B**), formally

```
xxx  $\widehat{=}$
  ANY $y$ WHERE
    $Q$
  THEN
    $S$
  POST
    $P(w, w\$0)$
  END
```

In order to simulate such a construct within the present **B** language, we are going to replace this operation by the following one (where we have replaced $w\$0$ by the fresh variable wz that is "assigned" the value w before execution as indicated):

```
ANY $y$ WHERE
  $Q$
THEN
  LET $wz$ BE
    $wz = w$
  IN
    $S$ ; PRE $P(w, wz)$ THEN skip END
  END
END
```

The specific proof obligation generated "naturally" by **Atelier B** in that case is the following (where I, is the abstract invariant and J is the gluing invariant):

$$I \ \wedge \ J \ \wedge \ Q \ \wedge \ wz = w \ \Rightarrow \ [S]\, P(w, wz)$$

In the very case we are interested in, namely the decreasing of a certain quantity $V(w)$, this would solve our problem in a very simple way by using the predicate $V(w) < V(w\$0)$. Here is the operation we obtain from the example already studied above:

```
ANY $y$ WHERE
  $Q(y, w)$
THEN
  LET $wz$ BE
    $wz = w$
  IN
    $w := F(y, w)$ ; PRE $V(w) < V(wz)$ THEN skip END
  END
END
```

This yields the following proof obligation

$$I(v) \ \wedge \ J(v, w) \ \wedge \ Q(y, w) \ \wedge \ wz = w \ \Rightarrow \ [w := F(y, w)](V(w) < V(wz))$$

that is

$$I(v) \ \wedge \ J(v, w) \ \wedge \ Q(y, w) \ \wedge \ wz = w \ \Rightarrow \ V(F(y, w)) < V(wz)$$

yielding

$$\boxed{I(v) \;\wedge\; J(v,w) \;\wedge\; Q(y,w) \;\Rightarrow\; V(F(y,w)) < V(w)}$$

which is exactly the desired result.

Limitation of Deadlock

In order to obtain the proof obligation stipulated above for the limitation of deadlock, we proceed as follows. We introduce an extra boolean variable *ddlck*, and an extra event deadlock defined at each step:

```
deadlock  ≙
   SELECT
      ¬ disjunction of guards
   THEN
      $ddlck$ := true
   END
```

When this event is refined, the proof obligations generated "naturally" by Atelier B results in the following (where $I(v)$ is the abstract invariant and $J(v,w)$ is the gluing invariant):

$$I(v) \;\wedge\; J(v,w) \;\wedge\; \neg\,\text{disjunction of concrete guards} \;\Rightarrow\; \neg\,\text{disjunction of abstract guards}$$

yielding by contraposition

$$I(v) \;\wedge\; J(v,w) \;\wedge\; \text{disjunction of abstract guards} \;\Rightarrow\; \text{disjunction of concrete guards}$$

Since we have a disjunction in the antecedent of this implication, this statement can be decomposed in an obvious way, yielding a series of proof obligations corresponding to each abstract guard. This yields exactly the formal result presented above (where $P(x,v)$ is one of the abstract guards):

$$\boxed{I(v) \;\wedge\; J(v,w) \;\wedge\; P(x,v) \;\Rightarrow\; \text{disjunction of concrete guards}}$$