Ampersand

Chapter 2

Ampersand

An Approach to Control Business Processes

1 Introduction

This chapter shows how to use business rules for specifying both business processes and the software that supports them. This approach yields a consistent design, which is derived directly from business rules. This leads to software that complies with the rules of the business.

Compliance

The approach, called Ampersand, is specifically suited for business processes with strong compliance requirements, such as financial processes or government processes that execute legislature. Features of Ampersand are: rules define a process, no process modeling is required, *compliance* with the rules is guaranteed, and rules are maintained by systematically signalling participants in the process.

A case study completes this chapter.

2 Rules

Ampersand lets you design information systems to control business processes. It is based on rules. But what exactly are rules? Merriam-Webster's dictionary contains over a dozen different definitions. Some of those are in agreement with this book:

– A prescribed guide for conduct or action.
– An accepted procedure, custom, or habit.
– A regulation or bylaw governing procedure or controlling conduct.

For practical purposes, however, you will need a definition that you can use when you design information systems and business processes. We need a definition that lets you tell a rule apart from other statements. The following definition provides the means to say whether a statement is a rule or not.

Business rule

A *business rule* is a verifiable statement that some stakeholders intend to obey, within a certain context.

Here is an example of a rule:

In our club, a coat of any guest shall be in the cloakroom, as long as the guest is in the club.

Let us analyse this statement, to better understand what we mean by a business rule.

Scope

a Rules have a *scope*.
The context of the stakeholders is our club in which this rule is valid. We call this the scope of this rule. A rule may or may not hold beyond its scope. People may or may not wear their coats outside of the club.

Stakeholder

b Rules have *stakeholders*.
Anyone involved in a rule is called *stakeholder*. For example, to ensure that guests put their coats in the cloakroom, there may be a bell boy to take the coats in and hand them out again, or there may be a staff member who sees to it that people who try to smuggle their coats inside are intercepted. Or a simple notice may inform the dear guests to leave their coats, or else. Whenever a rule has no stakeholders, then apparently no one is interested if the rule is obeyed or not. Such a

rule has no business merit, and we do not consider it to be a business rule. Stakeholders who "live by the rules" are working to satisfy rules, each in his or her own role.

Verifiable

c Rules are *verifiable*.
If there is a guest inside the club who holds a coat, we have a violation of this rule. We call a rule verifiable if its violations can be spotted unambiguously and objectively. That violation could be a signal to someone to take action. A floor manager might summon the offender to take his or her coat out to the cloakroom. Or, a staff member might take the coat from the guest and put it away in the cloakroom. Or even the guest himself might take some action. He might toss his coat out of the window, ensuring that it is beyond the scope of this rule. Technically, that would be an acceptable thing to do, unless there are rules in place that forbid littering the street... Whatever actions happen, the situation should be restored to where the rule is complied with.

For a better understanding, let us look at some counterexamples. The following statements are *not* rules:

– *Our club is transparent to the outside world.*
Whether this statement is true or not is open for discussion, depending on what you think "transparent" means. For this statement to be a rule, we must be able to determine objectively whether it is true or not. Within the context of this book, this statement is not a rule because it is not verifiable.

Concrete

Rules must have the property of being *concrete*.
– *Club members get up in the morning and go to sleep in the evening.*
This is not a rule if none of the stakeholders really cares. If nobody is willing to maintain the truth, we have no rule.

Relevant

Rules must have the property of being *relevant*.
– $E = mc^2$
This is a law of nature, which is considered true in any scope and without the intervention for any stakeholder whatsoever. No one will check to see if the rule is obeyed or violated, and certainly will no stakeholder put in an effort to undo violations. Even if we would consider this to be a rule, there is no need to maintain what mother Nature maintains for us. Such irrelevant rules and laws of nature are out of our scope, they are not business rules.
Business rules must represent agreements that people care about.
– *Peter Lee Jones has visited the club this morning.*
This statement can be either true or false, so it is a verifiable statement. And once we have established its truth, it will never change. Therefore, we call this a fact rather than rule, even though theoretically, there is no reason why facts should not be rules.

Rules usually assume a number of things tacitly. That is also the case in our example. The rule sounds: "In our club, a coat of any guest shall be in the cloakroom, as long as the guest is in the club." It assumes a number of things, such as:

– There is a club, which we call "our club". To avoid uncertainty, we had better remove the reference to "our" club, and supply the exact name of the club.
– Coats have owners, especially guests can be owner of a coat.
– Coats can be in the cloakroom. This also implies that there is a cloakroom.
– Guests stay in the club for a certain period of time.

If one of these assumptions is not true, the rule is meaningless. Requirements engineers, who write rules on behalf of stakeholders, must be aware of these tacit assumptions.

Also, the exact phrasing of a rule is really important. Rephrasing can cause problems, because there may be implicit assumptions underlying the statements. The following examples show how seemingly innocent rephrasings can unwillingly change the intended meaning:

- In our club, guests must put their coats in the cloakroom.
  This rule does not prevent a guest from taking his coat into the club. He or she can take the coat out, right after putting it in the cloakroom. To avoid this and similar situations, it is good practice not to prescribe actions, but to describe a state.
- In our club, the coat of each guest must be in the cloakroom, as long as they are in the club.
  This rule assumes that every guest has precisely one coat. If this is not the case, then what?
- In our club, all coats must be kept in the cloakroom at all times.
  In this case, coats of members and staff are also kept in the cloakroom...
- In our club, the bell boy will put your coat in the cloakroom.
  This rule affects anyone entering the club. It does not say what to do with your coat when the bell boy is absent. Besides, it is not specific about "you". In principle, this rule also applies to the mailman who drops by to deliver some mail...

In order to check whether a statement is a rule, please ask yourself the following questions:

a Can I decide objectively at any moment in time, whether the rule is satisfied or not? If so, this statement is verifiable. As a double check, can I think of a situation that violates the rule?
b Where and in which situation(s) does this statement make sense? This gives you the scope.
c Can you identify who are affected by this rule? If so, these people (or groups) are your stakeholders.
d Is there an intention to keep this statement true? If so, which stakeholder(s) take which action(s) to maintain this rule? If none of the stakeholders have such intention, your statement is not a rule.

3 Rules in Business

Define
Business process

Business rules can be used to manage and control business processes. In this sense, business rules actually *define* the *business process*. This yields compliant systems and compliant processes. This chapter explains the principle, which can be summarized as: signal violations (in real time) and act to resolve them. This drives a series of events to comply with all business rules. It lets us conclude that business rules are sufficient as an instrument to design compliant business processes and information systems.

Whenever and wherever people cooperate to work together, they coordinate their work by making agreements and commitments. These agreements and commitments constitute the rules of the business. A logical consequence is that these rules must be known and understood by all who have to comply. From this perspective, business rules are the cement that ties a group of individuals together to form an organization. In practice, many rules are documented, especially in larger organizations. Life of a business analyst can hardly be made easier: rules are known and discussed in the organization's own language, and stakeholders know (or are supposed to know) the rules and abide by them.

Maintain

The role of information technology is to help *maintain* business rules. That is: if any rule is violated, a computer can signal that and prompt people (inside and outside the organization) to resolve the issue. The Ampersand approach uses this as a principle for controlling business processes. For that purpose two kinds of rules are distinguished: rules that are maintained by people and rules that are maintained by computers.

A rule maintained by people may be violated temporarily, for the time required to fix the situation. For example, a rule might say that each benefit application requires a decision. This is violated from the moment an application arrives until a corresponding decision is made. Allowing the temporary violation gives a person time to make a decision. For that purpose, a computer monitors all rules maintained by people and signals them to take appropriate action. Signals generated by the system represent (temporary) violations, which are communicated to people as a trigger for action.

A rule maintained by computers need never be violated. Any violation is either corrected or prevented. If for example a credit approval is checked by someone without the right authorization, this can be signalled as a violation of the rule that such work requires authorization. An appropriate reaction is to prevent the transaction (of checking the credit application) from taking place. In another example the credit approval might violate a rule saying that name, address, zip and city should be filled in. In that case, a computer might correct the violation by filling out the credit approval automatically.

Procedure

Closed

Since all rules (both the ones maintained by people and the ones maintained by computers) are monitored, computers and persons together form a system that lives by the rules. This establishes compliance. Business process management (BPM) is also included, based on the assumption BPM is all about handling cases. Each case (for instance a credit approval) is governed by a set of rules. This set is called the *procedure* by which the case is handled (e.g. the credit approval procedure). Actions on that case are triggered by signals, which inform users that one of these rules is (temporarily) violated. When all rules are satisfied (i.e. no violations with respect to that case remain), the case is *closed*. This yields the controlling principle of BPM.

This principle rests solely on rules. Computer software is used to derive the actions, generating the business processes directly from the rules of the business. To compare: workflow management derives actions from a workflow model, that captures a procedure in terms of actions. Workflow models are specified by modelers, who take the rules of the business, and transform these into actions plus an appropriate but fixed order to achieve the desired result.

The new approach has two advantages: the work to draw up a workflow model can be saved and potential mistakes made by process modelers can be avoided. It sheds a different light on process models, whose role is reduced to documenting and explaining a process to human participants in the process. Process models no longer serve as a 'recipe for action', as is the case in workflow management.

The following section discusses an example, that illustrates this process control principle.

4 An example

Consider the handling of permit applications by a procedure based solely on rules. Each permit application is seen as a case to be handled, using the principle of rule based process control (section 3). First the business rules are given that define the situation. We subsequently discuss a scenario of the demonstration that is given with the generated software and the data model (also generated from the rules) on which that application is based.

The example consists of the following business rules.

a An application for a permit may be accepted only from one individual whose identity is authenticated.
b Each application for a permit must be treated only by authorized personnel.
c Every application must lead to a decision.

d An application for a particular type of permit may never lead to a decision about another type of permit.
e Every employee must be assigned to one or more particular areas.
f An employee may only handle applications from those areas to which (s)he is assigned.

First, we establish that each statement is indeed a business rule by showing that each rule is falsifiable. For that purpose, one example is given of a violation for each rule:

a An application for a permit from an individual with suspicious identity or credentials.
b An application that is handled by an unauthorized person.
c A permit application without a decision.
d An application for a building permit that leads to a decision about a hunting permit.
e An employee assigned to no area at all (perhaps an apprentice?).
f An employee assigned to Soho who handles an application from the East End.

As for the IT consequences, notice that violation d can be prevented by a computer, by consistently choosing the type of the permit as the type of the corresponding decision. This causes rule d to be free of violations all the time. Rule c may be violated for some time, but in the end a decision must be made. So the work is assigned to an employee who makes that decision. Rule e may also be violated for some time, but the employee cannot handle applications for the time being. Rules a, b, and f may be enforced by preventing all transactions that might violate the rule. Thus, a system emerges that complies with all these rules.

An application to controls this process has been built on a computer. The functional specification was generated by software that translates a set of (formalized) rules into a conceptual model, a data model, and a catalogue of services with their services defined formally. This specification defines a software system that maintains all rules mentioned above. The specification guarantees that many rules can never be violated, and the remaining ones such as c yield a signal as long as a decision on the application is pending. A compliant implementation was obtained by building a prototype generator that produces a database application according to the given specification.

An actual scenario of interleaved user and computer activities, used in demonstrations of information systems generated by business rules, proceeded as follows:

a An employee creates a new application for 'Joosten', who wants to have a 'building permit'.
b The system returns an error message for violating rule a. This means that an application for a permit from an individual whose identity is unknown is not accepted.
c The employee remembers he should have checked the identity of the applicant. He asks for identification and enters the applicant's passport number into the system.
d The employee can now record the new application. As far as this employee is concerned, he is done with the application.
e Next, an employee must be allocated for making the required decision. If an employee is chosen in violation of rules b or f, that transaction is blocked.
f The employee who makes the decision records it in the information system. The fact that this decision is about a building permit is copied (by the computer) from the application, without any interference from the employee.

Notice that this system may be criticized for picking an employee 'by hand'. This behaviour is a logical consequence of *not* having the rules in place for picking employees. One could argue that the system is incomplete, because there are too few rules. Adding appropriate rules will yield a process in which employees are assigned automatically. This illustrates how a limited (even partial) set of rules can be used already to generate process control. In practice, this means that process control may be implemented incrementally.

Automated data analysis tools can also use the business rules to produce specification artifacts such as an UML class diagram, or formal specifications of the software services required to maintain all rules. These deliverables are not shown here for the sake of brevity.

5 Control Principle

After discussing rule based design (section 3) and illustrating it with an example (section 4), let us discuss the consequences of rule based control of business processes in some more detail.

The principle of rule based BPM is that any violation of a business rule may be used to trigger actions. This principle implements Shewhart's Plan-Do-Check-Act cycle (often attributed to Deming) [31]. Figure 2.1 illustrates the principle. Assume the ex-

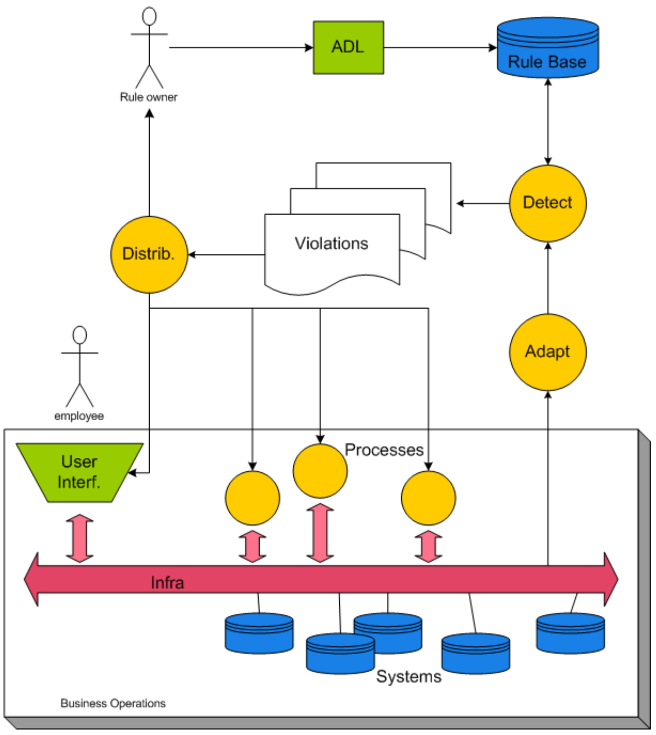


FIGURE 2.1 Principle of rule based process management

istence of an electronic infrastructure that contains data collections, functional components, user interface components and whatever else is necessary to support the work. An adapter observes the business by drawing information from any available source (e.g. a data warehouse, interaction with users, or interaction with information systems). The observations are fed to a detector, which checks them against business rules in a rule base. If rules are found to be violated, the detector signals a process engine. The process engine distributes work to people and computers, who take appropriate actions. These actions can cause new signals, causing subsequent actions, and so on until the process is completed.

Event The system as a whole cycles repeatedly through the phases shown in figure 2.2. The detector detects when rules are violated and signals this by analyzing *events* as they occur. The logic to detect violations dynamically is derived from the business rules.

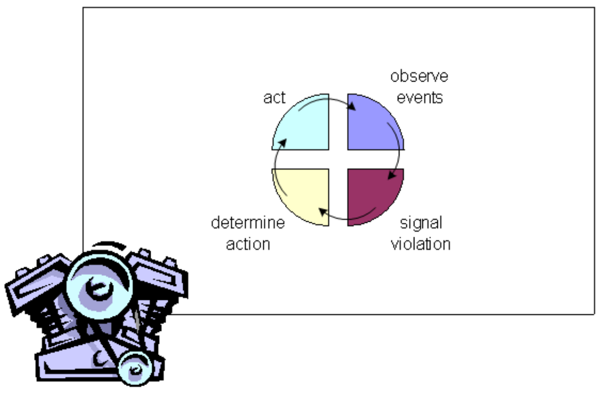


FIGURE 2.2 Engine cycle for a rule based process engine

Violation
Signal

This results in systematic and perpetual monitoring of business rules. Whenever a *violation* is detected, a *signal* is raised for the attention of some actor or actors Signals sent to specific actors (either automated or human) will trigger actions. These actions can cause other rules to produce signals by which other actors are triggered. So the actual order of events is determined dynamically.

6 Rule Management

Changing the rules is done by altering the contents of the rule base. The behaviour of the organization will change accordingly. A rule that is removed can no longer be violated, so the signals caused by that rule will cease to exist. New rules create new kinds of violations, which cause people to take new kinds of action. In this way, changing the rules has a profound impact on the processes governed by these rules.

For this reason, an organization must obviously manage their rules. Since the rules of the business change all the time, business processes change along with them. Documenting the rules is one thing, but a process needs to be in place to deal with rule changes.

Even rule changes are subject to rules. The process of legislature in a country is a good example; countries have rules that govern how new laws are made. In most organizations, simpler processes than that will be in place. By defining the "rule management process" as "the collection of rules that govern rule changes", you can use our approach to define that process, just like any other process.

7 Case Study: A Service Desk

This section demonstrates the Ampersand approach by means of a case study.

Practicable business rules
Invariant rules

Business rules can and will apply to people, processes, and overall corparate behaviour. If we restrict to rules that we want to capture in some information system (perhaps *practicable business rules* is a suitable name), then this book is primarily concerned with what we call the *invariant rules*, also known as integrity rules or constraints. These rules assert facts that must (need to, ought to, should) hold at all times. Several other categories of practicable business rules can be pointed out, but we will not be examining them in this book, mainly because those rules cannot be easily captured by way of Relation algebra and the Ampersand approach.

The case study is about an IT Service Desk that handles incoming customer calls about software and hardware problems. It illustrates a multi-step process involving components, problem solutions, and responses that may or may not be acceptable.

The process is driven by just a few business rules about the relations between incoming calls, the required responses, and business activities to provide those responses.

Our way of working is as follows.

a We start with the business rules. The customer's point of view first, and we add a few business rules that are indispensable from the point of view of internal processing.
b Based on these few rules, a small but coherent business process engine can be generated. We discuss the abstract structure of this engine. It consists of a conceptual model, and of a set of rules now rephrased as exact formulas.
c Next, we demonstrate how this engine works in practice, by tracing one call from start to finish. We show how the process is driven by alternating between performing some business activities, and signalling rule violations, until no violations remain.

Of course, it is just a small example and we cannot elaborate on the details, exceptions or extensions. However, you are welcome to try and expand the example. This will give you a feel of our Ampersand method, and also about the versatility of the business rules approach in general.

7.1 THE BUSINESS RULES

From the customer's point of view, we can establish one all-important rule:

1 Every call must get an acceptable response.

No more, no less.

This rule ensures that every process instance, once initiated, will get to be concluded. In effect, this one rule governs the entire process, driving it from start to finish. The rule is violated as long as there exists some call with no response at all, in other words: there is work to do. But even if some response is available for a call, the rule may be violated. For instance, the response may be recorded in the system, but nobody thought to tell the customer. Or, the client is informed but the client does not accept it because it does not solve the problem. The wording of the rule is very precise: it requires that the response must be acceptable.

Switching to the point of view of internal processing, we have several more rules:

2 Every call is entered by exactly one client.

3 Every call involves at least one hardware- or software component.

4 Every response describes at least one problem solution that applies to at least one component involved in the call.

Let us briefly explain these rules. The requirement 2, that every call should originate with exactly one customer, is quite natural. And it ensures that the company can send an invoice to each customer, for services rendered; a process however that is beyond the scope of our example.

Rule 3 states that a call must involve some hardware- or software components. This also is rather obvious: if no components are involved, then why put in a call? Later on, we will see that both rule 2 and rule 3 establish a property of one particular relation, a type of business rule that is generally named "multiplicity rules" or "cardinality".

The fourth rule states that every response must describe at least one problem solution. Not just any problem solution: it must apply to some component or other that is involved in the call. The idea is that a problem solution can be useful only if it applies to at least one of the components involved in the call. Notice that this rule is a simplification of reality. Call analysis, assessing which components are involved, and determining the problems and appropriate solutions can be quite a difficult job. Again, we consider this part of the process beyond our scope. We simply assume that the job gets done somehow, and that all knowledge about components and problem solutions is recorded somehow, somewhere.

7.2 SPECIFICATION OF THE RULE ENGINE

By inspecting the few rules above, we can see there are five concepts involved: Client, Call, Response, Component, and Problem Solution.
These concepts are involved in six relationships, such as:

- Call is *placed_by* a Client, and
- Problem Solution is *described_in* a Response.

Of course, all kinds of refinements and extensions can be conceived. But for this case study, we are satisfied with our five concepts and six relationships. A suitable way to understand and discuss these is by drawing a diagram, such as figure 2.3. This

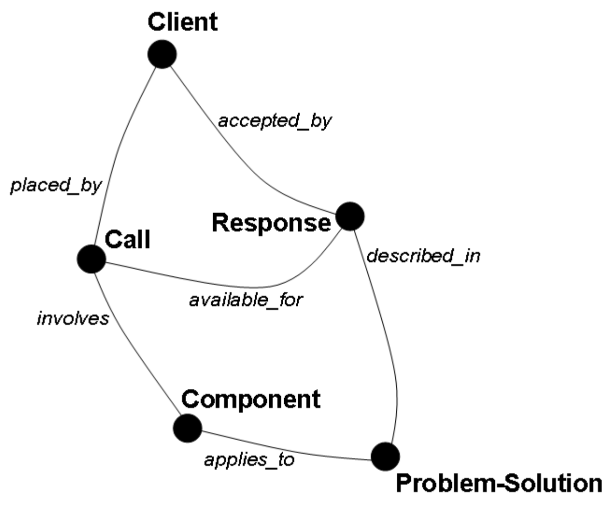


FIGURE 2.3 Diagram of the conceptual model

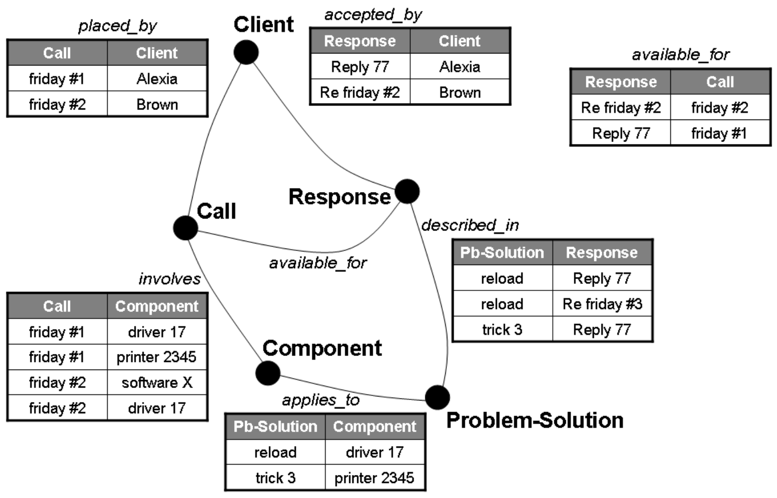
diagram gives a clear picture of the concepts and relations involved in our rules. But please beware that the picture does *not* show any rules at all. It merely depicts the structure, a conceptual model consisting of concepts and relations.

To get at the actual rules, we need to delve somewhat deeper. In fact, there is some mathematics involved to rephrase the rules in perfect detail. Perfect meaning: precise and unambiguous. So much so that a computer can read the stored data for all the concepts and relations, and calculate each and every violation of the rules. The mathematical rephrasing of rules into exact formulas will be extensively discussed later in the book. For now, we will use the four rules as phrased above.

The conceptual model and the rules together specify exactly what is needed to run the business. In effect, they constitute the functional specifications of a business process engine that will maintain these rules, and signal any violations. Non-functional specifications concern important aspects of information systems design that we do not capture in our business rules. For instance, security demands, scalability properties, response times, user interface requirements etc.

7.3 THE RULES AT WORK

Let's play the part of a helpdesk employee. As you enter the Service Desk workspace on a friday morning, you notice that two calls have already been handled this morning. All relevant facts about these calls have been recorded correctly. Those facts may be inserted in the previous diagram, as shown in figure 2.4. You may check



placed_by

Call	Client
friday #1	Alexia
friday #2	Brown

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #3
trick 3	Reply 77

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17

applies_to

Pb-Solution	Component
reload	driver 17
trick 3	printer 2345

FIGURE 2.4 Initial data on record. No violations

that indeed, all four rules are complied with. There are no violations to be resolved, no work to be done. You may notice a component named "software X" for which at present, no problem solutions are known. This however is no cause for alarm, because a lack of solutions violates none of our rules.

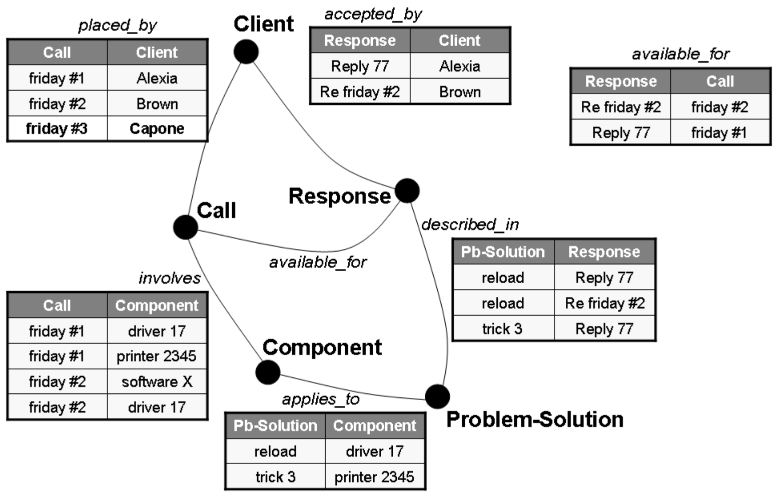
Then, a new call comes in from the client named Capone, and your work begins.

Step1 In the relation *placed_by*, you enter the client name and the call identifier, which is "friday #3". In figure 2.5, the new fact is inserted in the correct place. For your ease of reading, the new fact is placed at the bottom of the table, and highlighted. For the business process engine however, positioning nor highlighting have any significance. As the data is being inserted into the computer, the engine will check the rules, and detect two violations:

- rule 1 says that every call must have an acceptable response, but no response is available for "friday #3".
- rule 3 says that every call is related to at least one hardware- or software component, but "friday #3" is unrelated.

Notice that there is no priority among these violations, there are no rules that tell you what to do first. Rules that dictate the order in which activities must be executed are sometimes called imperative rules. In practical situations, rules about the particular order of work may be convenient, as it keeps track of what can or needs to be done. But even more often, you will find that the particular order is unimportant, the real importance is what can or needs to be done.

In our case example, you need to find out which components are involved in this call. Let us assume that you find that two components are involved: the "software X" and that "driver 17", again. And you should also prepare some response for this particular call. So you enter three new facts into the computer:



placed_by

Call	Client
friday #1	Alexia
friday #2	Brown
friday #3	**Capone**

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #2
trick 3	Reply 77

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17

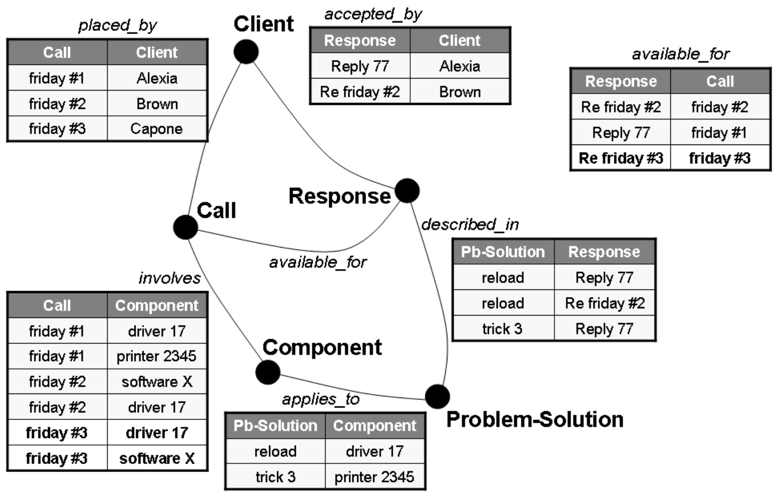
applies_to

Pb-Solution	Component
reload	driver 17
trick 3	printer 2345

FIGURE 2.5 Step 1: new call placed. Violations

– the call "friday #3" involves the component "software X",
– the call "friday #3" involves the component "driver 17", and
– response "Re friday #3" is available for the call "friday #3".

Step 2 You enter the data into the system, while realizing that it does not matter whether you do it one at a time (in any arbitrary order), or all at once. Once you



placed_by

Call	Client
friday #1	Alexia
friday #2	Brown
friday #3	Capone

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1
Re friday #3	**friday #3**

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #2
trick 3	Reply 77

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17
friday #3	**driver 17**
friday #3	**software X**

applies_to

Pb-Solution	Component
reload	driver 17
trick 3	printer 2345

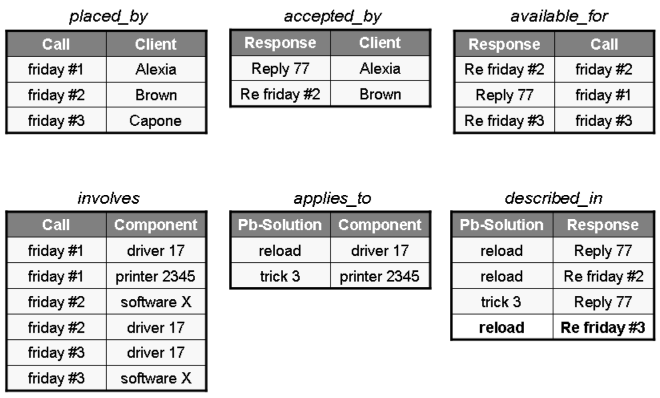
FIGURE 2.6 Step 2: call analysis. Other violations

have entered all three tupels, you find that some of the previous violations have been remedied, but now other violations emerge (figure 2.6).

– rule 1 is still being violated. A response is available for "friday #3", but as yet, the client has not accepted it.
– rule 4 says that every response describes at least one problem solution. Moreover, that problem solution must apply to at least one component involved in the call.

The available response is empty, as it describes no problem solution.

Step 3 The helpdesk employee (you) now notices that component "driver 17" is involved, and a problem solution is already known for that one: "reload" the driver. So you *describe_in* your response that particular problem solution (figure 2.7).



placed_by

Call	Client
friday #1	Alexia
friday #2	Brown
friday #3	Capone

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1
Re friday #3	friday #3

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17
friday #3	driver 17
friday #3	software X

applies_to

Pb-Solution	Component
reload	driver 17
trick 3	printer 2345

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #2
trick 3	Reply 77
reload	**Re friday #3**

FIGURE 2.7 Step 3: try one solution

Incidentally: this figure depicts only the tables and the tuples populating them, but no more. This way of representing the information may be harder to understand for people. But of course, it does not affect the workings of the process engine in any way.

When you record this tuple, the engine will calculate that rule 4 is no longer violated. This rule concerns four relations (*available_for*, *involves*, *applies_to* and *described_in*). Notice how new tuples were entered in three, but not all four relations. In this case, the violation is resolved because a suitable tuple already existed in the fourth relation, *applies_to*. Thus, a suitable response is now available.

Step 4 As yet, the client has not accepted a response, and rule 1 is still being violated. You, or the computer, may infer from the data that only one tuple in relation *accepted_by* seems to be missing. Indeed, automatically inserting the fact "Response Re friday #3 is *accepted_by* Client Capone" would eliminate the final violation.

But this is a bad idea. The computer cannot make a real-world decision like deciding whether a response is acceptable or not. And indeed, when asked, the client clearly states that reloading driver 17 does not solve the issue. Hence, this fact may not be entered into the *accepted_by* relation, neither automatically nor manually. Contacting the client has not really changed the situation of step 3. The recorded facts are still as shown in figure 2.7, and the violation of rule 1 is still there.

Step 4 revisited So you need to come up with another response that is better than the one available now. Let us assume that you invite somebody else, a back-office employee, to examine the nature of the call and the components involved. She notices that "Software X" may be the cause of the trouble. A reinstallation of that component may solve the issue, and so she adds a new tuple into the relation *applies_to*. She informs you of the addition, and the situation is now as in figure 2.8.

placed_by

Call	Client
friday #1	Alexia
friday #2	Brown
friday #3	Capone

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1
Re friday #3	friday #3

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17
friday #3	driver 17
friday #3	software X

applies_to

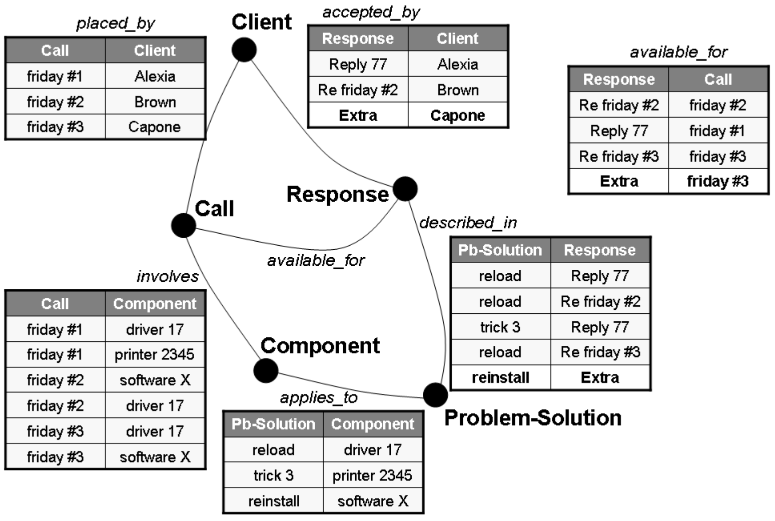
Pb-Solution	Component
reload	driver 17
trick 3	printer 2345
reinstall	**software X**

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #2
trick 3	Reply 77
reload	Re friday #3

FIGURE 2.8 Step 4: other solution

Step 5 This new problem solution alone, does not remedy the violation of rule 1. But now that it is available, you can easily compose a new response for the client. Your new response gets the name "Extra" as its identifier, and you record several facts for it. The new problem solution is *described_in* it, you make it *available_for* the call, and you inform the client of it. And luckily, this time it works: the response "Extra" is *accepted_by* the client "Capone". Moreover, when you insert all this information into the three corresponding relations, the computer detects no more violations. All business rules are complied with, which is to say that the process terminates (figure 2.9).



placed_by

Call	Client
friday #1	Alexia
friday #2	Brown
friday #3	Capone

accepted_by

Response	Client
Reply 77	Alexia
Re friday #2	Brown
Extra	**Capone**

available_for

Response	Call
Re friday #2	friday #2
Reply 77	friday #1
Re friday #3	friday #3
Extra	**friday #3**

described_in

Pb-Solution	Response
reload	Reply 77
reload	Re friday #2
trick 3	Reply 77
reload	Re friday #3
reinstall	**Extra**

involves

Call	Component
friday #1	driver 17
friday #1	printer 2345
friday #2	software X
friday #2	driver 17
friday #3	driver 17
friday #3	software X

applies_to

Pb-Solution	Component
reload	driver 17
trick 3	printer 2345
reinstall	software X

FIGURE 2.9 Step 5: no more violations. Process terminates

Summary From a business perspective, the following scenario has been executed:

– The client placed a new call.

– The system signalled violations of two rules.
– The helpdesk workers then performed various business activities, and recorded the data obtained in these activities.
– Every time, the system used the latest data to check compliance to the rules and generated new signals for each rule violation.

After a number of steps, and having entered all the relevant data, all the rules were complied with, no violations persisted, and no signals for work needed to be raised. This means that this, and in fact all calls have been processed successfully. Our case is complete.

7.4 POINTS OF INTEREST

This case study is rather simple. It describes the essence of a primary process: how a call for service is answered. Several interesting observations can be made even in this simple case:

Declarative versus imperative rules Our rules are declarative in nature. They describe which states are okay: every rule is satisfied, there are no violations. Or not, some rule is violated, and there is work to be done. But the rules do not say how, by whom, or in which work sequence the activities need to be executed. Imperative rules dictate exactly what must be done how, when and by whom. It is well possible to write imperative rules, but we advocate against it because such rules will produce process flows that cannot be easily adapted to new demands. The declarative rules do not enforce any particular sequence of work. It lets the workers free to decide whichever activity can or should be done.

Application supports the workers The information system generated from rule requirements should determine whether the data being entered, combined with the data already present, violate the rules. The system can then either prevent the new data from being entered (the data is rejected for being wrong). Or the system should signal violations to the workers so that corrective actions can be taken. But the system should not enforce a particular order of entering the data about real-world events. Ampersand's concept of business process does not presuppose any event sequence. The only provision on the process is that, in the end, all rules are satisfied and no violations remain.

Processing adds data As the business process is being executed, and violations are being resolved, data is constantly being added. Less common is that data already recorded in the system has to be changed. Changing some data may raise new and often unexpected violations. As a consequence, rework may be required for cases that are currently being processed or even for completed cases. In exceptional cases, you may even have to delete some data. But deletions may have even greater consequences than mere changes. For instance, an all-too-easy way to deal with a call, is to delete the fact that it was placed. Such an act would frustrate the very purpose of the business process engine.

Violations and activities are not the same All violations can and should be remedied. This is achieved by executing activities (and recording the data into the system). However, it is wrong to assume that a single violation corresponds to one bit of data or one particular activity. The case shows examples where a single new fact results in several violations. Also, one violation may require multiple changes or additions of data items for elimination. We even have an activity that resulted in no data at all (the client was informed of the response "Re friday #3" but this response was not acceptable). The implication is that a computer may calculate the violations, but it cannot always determine the appropriate actions to eliminate them. It often requires some judgment to decide what is wrong, and what must be done to remedy the rule violations that the computer signals. For that, we will always need people with a sound judgment of reality.

Extensions Our specifications contain five concepts, six relations, two multiplicity rules, and two composite rules. Many extensions are conceivable. It is important to realize that whenever you add something new to these specifications, be it a concept, a relation or a rule, you must do so for a sound purpose based on business goals. To illustrate the point, let us consider some extensions.

– A rule like "a Response is *accepted_by* at most one Client" seems appropriate. In fact, every relation should be inspected to determine its multiplicity constraints; as will be explained in the next chapter.
– We stated that the client was informed of the response "Re friday #3" but this response was not acceptable. Apparently, a relation "Client is *informed_of* Response" might be added to model this part of the business. And whenever you add a relation, you should also think about rule(s) that might apply to it. Here, you probably want to control the business process by ensuring that "IF a Response is *accepted_by* a Client, THEN that Client is *informed_of* that Response". This kind of composite rules will be discussed in chapter 4.
– Another extension may be to discern between "Problem" and "Solution", that are now lumped together into one concept. You might want tot replace this by two concepts plus a relation. This kind of improvement, and many others, will be covered in chapter 5.
– Still other options are to record which employees are handling the calls, or to add a rule that calls may be placed only by clients who have negotiated a service contract. In the end, the business must decide which rules serves their purposes best. And it is up to you, as a designer, to capture those rules in the specifications in the best possible way.

This concludes our demonstration how a business process is controlled by way of business rules. And even though our case study included only four rules, the business process being driven is far from trivial.

The computer brings each signal to the attention of some designated actor(s) who must take action to remedy the violation. The actor will then enter new information into the system (or sometimes adjust the data already in the system). The computer will then check all the data, old and new, and signal the latest violations. This process is repeated until no violations remain. As a consequence, when all rules are satisfied, there is no more work to be done, and the case can be closed.

The approach taken by Ampersand is to drive the process by alternating between the business actions (choosing and executing business activities, and recording the data), and system actions (inspecting all the rules and signalling rule violations), until no violations remain. Moreover, the Ampersand approach ensures that in the end, all the rules will be complied with. This concludes our demonstration how compliance to the business rules provides us with a sound way to control the processes of the business.

8 Consequences

Controlling business processes directly by means of business rules has consequences for designers, who will encounter a simplified design process. From a designer's perspective, the design process is depicted in figure 2.10. The effort of design focuses on requirements engineering. The main task of a designer is to collect rules to be maintained. From that point onwards, a generator (G) produces various design artifacts, such as data models, process models etc. These design artifacts can then be fed into an information system development environment (another generator la-

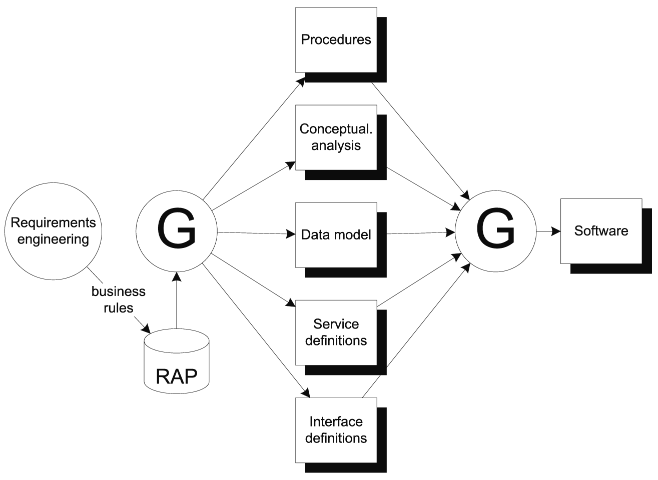


FIGURE 2.10 Design process for rule based process management

belled G). That produces the actual system. Alternatively, the design can be built in the conventional way as a database application. The rule base (RAP, currently under development) will help the designer by storing, managing and checking rules, to generate specifications, analyze rule violations, and validate the design. For that purpose, the designer must formalize each business rule, using a supportive tool (Ampersand). He must also describe each rule in natural language for the purpose of communication to users, patrons and other stakeholders.

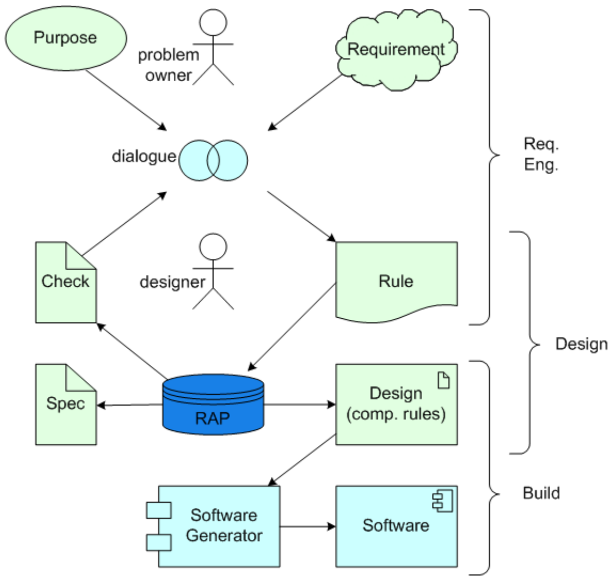


FIGURE 2.11 Design process for rule based process management

From the perspective of an organization, the design process looks like figure 2.11. The focal point of attention is the dialogue between a problem owner and designer. The former decides which requirements he wants and the latter makes sure they are captured accurately and completely. The designer helps the problem owner to make requirements explicit. Ownership of the requirements remains in the business. The designer can tell with the help of his tools whether the requirements are sufficiently complete and concrete to make a buildable specification. The designer sticks to the principle of one-requirement-one-rule, enabling him to explain the correspondence of the specification to the business.