Requirements Engineering in the Solution Domain

6

Never tell people how *to do things.*
Tell them what *to do,*
and they will surprise you with their ingenuity.
George Smith Patton, general, 1885–1945

6.1 What is the Solution Domain?

The solution domain is the domain in which engineers use their ingenuity to solve problems. The primary characteristic that differentiates the solution domain from the problem domain is that, invariably, requirements engineering in the solution domain starts with a given set of requirements. In the problem domain requirements engineering starts with a vague objective or wish list. The extent to which the input requirements for the solution domain are "well formed" depends on the quality of the people within the customer organization that developed them. In an ideal world, all the requirements would be clearly articulated, individual testable requirements.

As indicated in Chapter 2, the solution is very rarely arrived it in a single step (see Figure 6.1).

At each level there is modelling and analysis done first to understand the input requirements and second to provide a sound basis for deriving the requirements for the next level down. The number of levels of design is dictated by the nature of the application domain and the degree of innovation involved in the development. No matter how many levels are necessary, it is always vital to understand how many solution details – the "how" – should be introduced at each step.

At every level in the solution domain, engineers must make decisions that move towards the final solution. Each of these decisions, by their very nature, reduces the available design space, *i.e.* they preclude certain design options, but it is impossible to make progress in the absence of decisions. Engineers are always very strongly tempted to go into too much detail too soon. This temptation must be avoided, in order to allow creativity and ingenuity to work together to produce innovative solutions that could never be achieved in the presence of the constraints imposed by premature design decisions.

Typically the first level of system development in the solution domain is to transform the stakeholder requirements into a set of system requirements. These

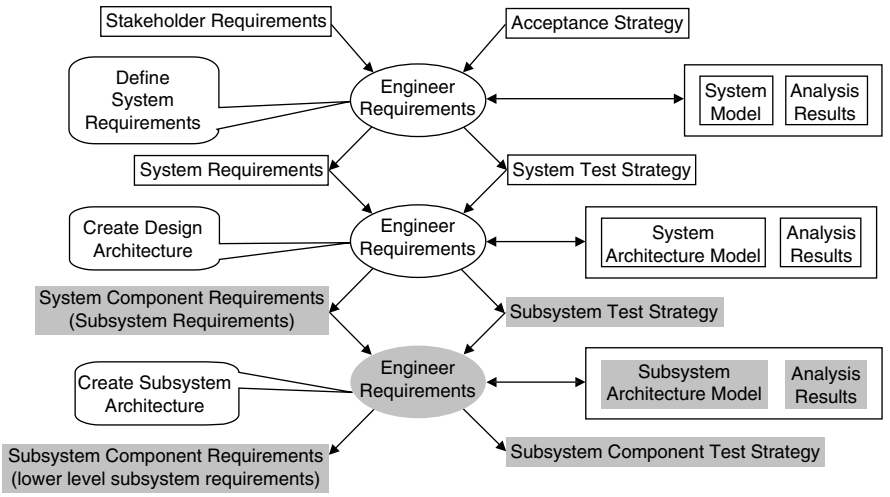

Figure 6.1 Possible instantiations of the generic process.

must define what the system must do in order to solve the problems posed by the stakeholder requirements. This first level is illustrated by the top instantiation of the generic process in Figure 6.1.

The issue of premature design detail is especially problematic at the first step. The system model indicated in Figure 6.1 must be created at a level of abstraction that enables the functionality of the system to be defined without going into unnecessary detail.

The next step on from defining the system requirements is to create an architectural design as indicated by the second instantiation of the generic process in Figure 6.1. This must be expressed in terms of a set of components that interact to generate the emergent properties identified by the system requirements. The derived requirements from the architectural design process (Figure 6.1) define the requirements that the component suppliers must satisfy for each component.

Development proceeds by further levels of design that move further towards implementation detail.

This chapter concentrates on the transformation from stakeholder requirements to system requirements because it is the most problematic in most developments, because typically too much detail is added too soon.

6.2 Engineering Requirements from Stakeholder Requirements to System Requirements

The full instantiation of the generic model for this transformation is shown in Figure 6.2.

As with all instantiations, the process commences by agreeing the input requirements, which, in this case, are the stakeholder requirements. The agreement process must not assume that the input requirements have been produced according to the guidelines given earlier in this book. Instead, it is necessary to

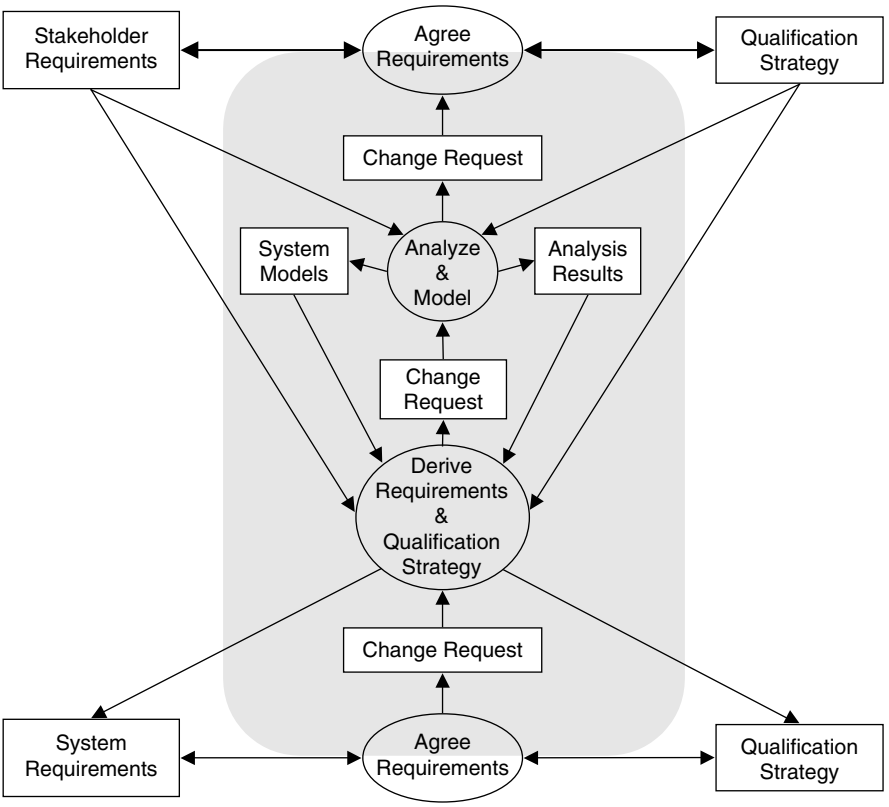

Figure 6.2 Instantiation of generic process to create system requirements.

consider the requirements and the associated qualification strategy on their merits and apply the review criteria for stakeholder requirements with rigour and thoroughness.

6.2.1 Producing the System Model

To avoid the tendency to go into too much detail, engineers should always work in the context of a model (see Figure 6.1) that is sufficiently detailed for the purpose of defining requirements in terms of what should be done rather than how. The level of detail that should be provided in derived requirements depends on the level of development at which requirements engineering is being done, but the maxim is always "do not add more detail than is necessary". The temptation to go into detail is always greatest at the top level where stakeholder requirements expressed in problem domain terms are being translated into high-level system requirements that indicate what the system must do to solve the problems posed by the stakeholders. The difficulty arises because of the need to work at an abstract level. The creation of an abstract system model, which will provide the framework for the system requirements, always causes problems. At all levels below this, development work progresses in the context of a design architecture.

Engineers are much more comfortable with this level of detail, because they can become involved with determining how the system will work. Even at these levels, care must be exercised to ensure that the amount of detail imposed is appropriate. Consequently, the architecture models should be expressed in terms of components that work together, but care should be taken to ensure that components are defined in terms of what they are required to do rather than how they should achieve it. In other words, components should be specified as "black boxes" whose internal details are of no concern provided that they achieve their overall purpose as defined in the requirements.

The next sections of this chapter concentrate on the preparation of system models for the derivation of system requirements. Following this, the ways in which the same approach is applied at more detailed levels is explained.

6.2.2 Creating System Models to Derive System Requirements

The system model must be created at an appropriate level of abstraction such that it encompasses:

- internal functionality that the system must exhibit – this must concentrate on *what* the system must do rather than on *how* it should be done to avoid pre-empting the design;
- functionality necessary to enable the system to interact with other systems in its environment;
- functionality necessary to enable people to successfully interact with it;
- functionality to prevent the system from malfunctioning owing to the presence of other systems (threats) in its environment. (Note that some of these systems may not be deliberately threatening, *e.g.* electromagnetic emissions from neighbouring equipment.) This "safeguard" functionality must also prevent the system from interfering in an adverse way with the environment.

The way in which these types of functionality interact with each other and with elements in the system's environment is expressed diagrammatically in Figure 6.3. It is clear that the context of the system within its environment must be defined with respect to:

- the existing systems with which the new system is required to cooperate;
- the types of people who are intended to interact with the system;
- the threats that the system must defend against;
- the adverse effects that must be prevented.

The functionality can be represented in a number of ways, for example:

- operations or methods on classes in class diagrams;
- message sequence charts;
- state transition diagrams;
- function flow block diagrams;
- processes in data flow diagrams.

In practice, it will be necessary to use several models in order to cover the many different aspects required. Each model contains information of a defined set of

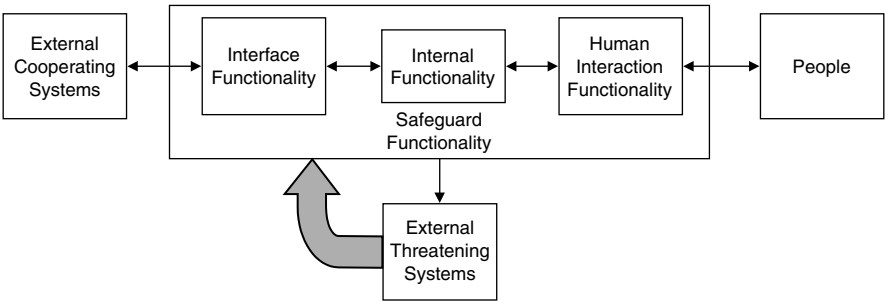

Figure 6.3 System context and types of functionality.

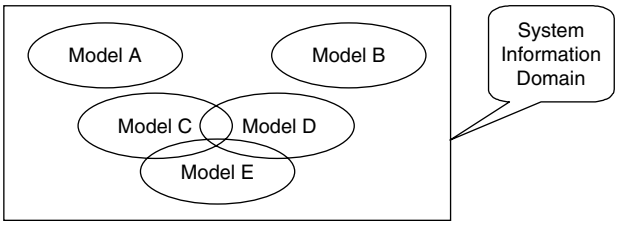

Figure 6.4 Scope of system models.

types and each modelling technique carries its own semantics. The information for some models may be separate from information in other models. On the other hand, the same information may appear in more than one model. In the latter case, it is essential that, when information is changed, the change is reflected in all other models in which that information occurs. Ideally this would be achieved automatically by linking the modelling tools. If this is not the case then extreme care should be exercised to ensure that any change is applied *identically* in all relevant models. The Venn diagram in Figure 6.4 indicates that some models represent islands of information whereas others may have common information presented in different forms. Figure 6.4 also indicates that there may be some system information that is not present in any model.

Internal Functionality

This is the primary element of the creation of the system requirements, because it is the main focus of defining what the system will do. It is necessary to create a structure or model that can be the basis for creating the system requirements. This model must have the capability to express some form of decomposition of the system into modules or high-level components such as subsystems. The use of terms such as "module" or "component" tends to make developers think more in terms of implementation rather than specification. This is generally considered to be bad practice, especially in software-based systems. In general systems, the need to move to a more physical model is not considered to be particularly problematic, since the application domain will generally be of a more physical nature.

As an alternative to terminology that may induce premature implementation, there is an increasing tendency (some would say "fashion") to use the term "object" as the decomposition element, especially for software-based systems, since objects can refer to items in the problem domain. This discipline helps to prevent the premature descent into solution terms. Functionality can then be introduced as methods or operations on objects and as interactions between objects.

The use of this object-oriented approach can also make the creation of traceability from the system requirements to the stakeholder requirements an easier task, because the same objects tend to be visible in both the problem domain and the solution domain.

In addition to stating what the system must do, the system model may also be required to indicate the intended behaviour of the system. There are a number of ways in which to represent this type of information. Models in this area usually represent the fact there are a number of concurrently active "actors" that interact in some way. Examples of this sort of notation are message sequence charts and behaviour diagrams. Message sequence charts have long been used in the telecommunications industry. Behaviour diagrams originated in the US ballistic missile early warning system (BMEWS) in the 1970s and have been implemented in tools such as RDD-100 from Ascent Logic Corporation and CORE from Vitech Corporation.

Behaviour can also be modelled using state transition diagrams or statecharts. State transition diagrams have the limitation that they can only model a sequence of states and the item being modelled can only be in one of these states at any one time. State transition diagrams cannot represent hierarchical states directly. Separate diagrams must be used for each level in the hierarchy and, in some cases this means that there may be a set of active diagrams at certain times. Such sets of diagrams can be difficult to understand. To avoid this complexity, it is better to use statecharts because they have been developed to handle state hierarchies directly. They also address parallel states.

In any system it is necessary to consider whether there is information to be handled. Some systems, *e.g.* insurance company systems, require that information must be gathered and retained for use over a number of years. In other systems, *e.g.* radar data processing systems for air traffic control, there is some information that has a long lifetime, *e.g.* flight plans, whereas the current position of an aircraft in flight, by its very nature, is soon out of date. Hence the information requirements must be examined to ascertain:

- the longevity of the information, *i.e.* for how long is the information relevant, and for how long must it be retained?;
- the freshness of the information, *i.e.* how up to date must it be (seconds, minutes or hours)?

It is also very relevant to know the amount of information that may be involved. This can have a profound effect on the design of the system.

Interface Functionality

It is necessary to define the nature of the interactions required with any other system. Interactions may involve the movement of information or material

between the systems. The movement may be in one direction or both, and there may be limits on the capability that can be transferred. It may be necessary to provide temporary storage (*e.g.* data buffer or warehouse) for items that are held up. There may be time response requirements on the speed with which either system must react to a stimulus from the other.

The nature of interfaces varies significantly. However, there must always be a baseline reference that indicates what each party undertakes to do or provide as part of the interface. These obligations are frequently documented in an interface control document (ICD). Where the interactions are controlled by national or international standards, the standard becomes the interface control document to which all relevant parties can work. Where the interface is less well defined, the obligations (*i.e.* interface requirements) must still be written down and agreed. Control of these requirements is notoriously difficult because there is often no organization with a clear mandate to control the interface. Consequently, each party to the interface tends to have its own version of the document and, worse, each party tends to have its own interpretation of it.

It is usual for interface documents to be controlled by the organization that has responsibility for the system that encompasses the two (or more) systems that interact. Such an organization is difficult to define when a new system is being developed. Often the existing system(s) will have been developed earlier and the interfaces may not be properly documented. Further, the development organization may well no longer have any responsibility for that system, having handed it over to a higher level customer or other operating authority.

Care must be exercised to ensure that all interface obligations are accurately reflected in derived requirements at the appropriate level and, as far as possible, the interface control authority is clearly defined.

Human Interaction Functionality

The crucial issue for human interactions with a system is to know what interactions are going to be required. The context in which the users will work is also important. This can have an impact on the way they can work. For example, users working in a standard office environment will be warm and able to work conveniently without gloves. Other users may have to operate the system in harsh environments such as extreme cold weather or hazardous situations where protective clothing will be necessary. In these circumstances, the design of the displays and keyboards must take note of these aspects.

Safeguard Functionality

The environment in which a system must operate will also have a significant influence especially with respect to safety and security. For example, in a banking system it is necessary to provide assurance that information and money will not be given to unauthorised people. In a car (system) it is necessary to be assured that the car will stop when the brake pedal is operated.

There may also be other systems operating in the environment of the system that may be competing with the system being developed. This competition could

be commercial competition as in online banking for example. Here the need for any system to be evolved rapidly can be of prime importance.

Other "competing" systems include those that could interfere with the correct operation of a system by, for example, making radio transmissions that confuse the system or overload sensitive receivers. An example of this is the worry that the use of mobile telephones on board aircraft in flight could interfere with the aircraft's navigation systems.

System Transactions

It is worthwhile revisiting the use scenarios that were developed for the system from the stakeholders or, if none exist, to create a set of relevant scenarios. These can be applied to the system model(s) to make sure that they are possible within the system being specified (see Figure 6.5). Working through these "system transactions" provides reassurance that elements of system functionality have not been lost by a blinkered approach to object modelling or functional decomposition. (Note that this use of the term "system transaction" is different to the use of the term within the CORE method described in Chapter 3.)

The system transactions shown in Figure 6.5 as user system transactions are those derived from the use scenarios. Figure 6.5 also indicates that there can be other transaction derived from the way in which the system being developed must interact with external systems.

System transactions encourage system engineers to stand back and take a "holistic" view of the system. It is all too easy to concentrate on the detail and forget the big picture, *i.e.* how do the detailed parts work together to achieve the overall aim?

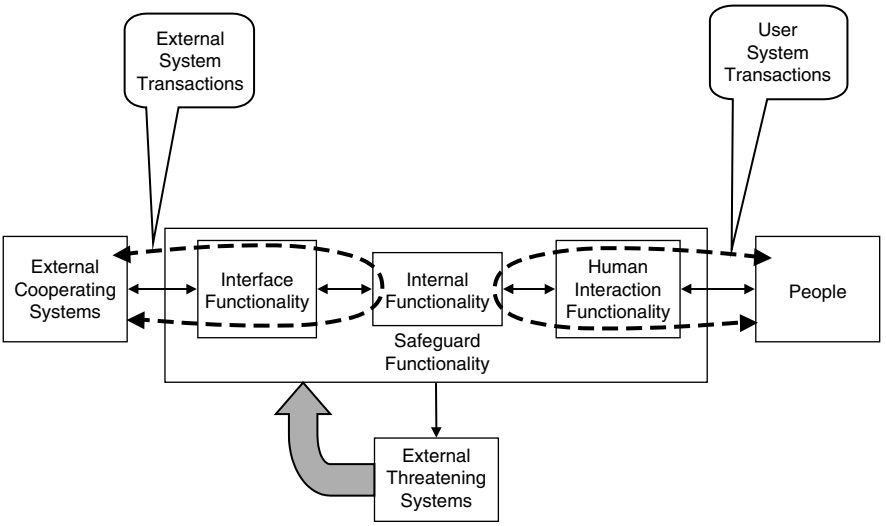

Figure 6.5 System transactions.

Modes of Operation

Different functionality may be required in some circumstances. A typical example for information-based systems is the need to be able to provide training for staff without compromising the integrity of the data held in the system. Other examples include fallback modes of operation following a failure or, in military systems, different modes depending on the current state of alertness. These may be related to the use scenarios in the stakeholder requirements.

Additional Constraints

In addition to the constraints already mentioned, there are additional aspects that should be considered. Perhaps the most important are those concerned with safety and certifiability. In these areas, additional requirements can be introduced and these will certainly have a strong influence on the means adopted for qualification. The relevant authorities will have to be convinced that a system is safe to use or to be deployed or, in the case of an aircraft, that it can be given a certificate of airworthiness.

A further set of additional constraints are introduced by the need to manufacture the system. It may be necessary to use an existing facility or the design may have to be changed in order to reduce the cost of manufacturing.

6.2.3 Banking Example

In this example of a management information system, the primary concern will be to model the information that must be handled, but it is clear that there are many other areas that should be addressed. Several system models are therefore likely to be used, one focusing on the information, others focusing on the flow and security of information.

Figure 6.6 shows a model that provides an alternative abstraction for the bank example. It identifies the types of locations where equipment might be sited and thus from where transactions may be initiated.

Internal Functionality

The primary internal functionality is concerned with supporting the services provided by the bank such as current accounts, deposit accounts, loans and investment portfolios. To support these services, the system must be able to collect, update and retain information. Of vital importance here are the types (or classes) of information (*e.g.* accounts, customers), the relationships that exist between them (*e.g.* how many accounts is a customer allowed to have?) and the longevity, freshness and volume of each type.

It is important to determine how information is collected, disseminated and manipulated.

A further important aspect of a banking system is the number and location of sources of information and/or transactions. These will include branches, automatic teller machines and credit card point of sale machines.

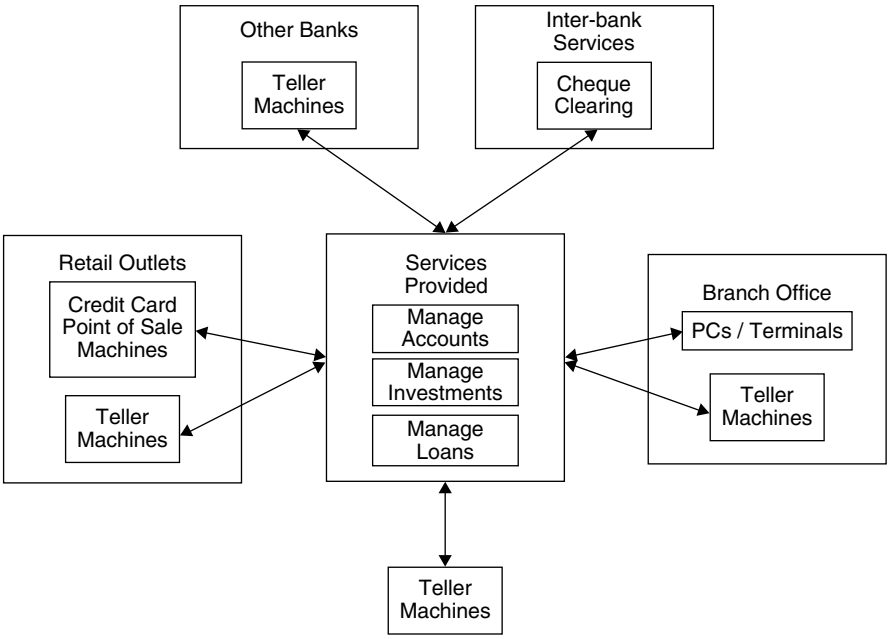

Figure 6.6 An abstract model for the bank example.

From a performance point of view, it is important to understand the likely loading that the system must be able to cope with, such as the number and mix of transaction types. This will clearly vary from day to day and from hour to hour within each day. There may be constraints imposed by existing infrastructure such as landlines or other communication mechanisms.

Interface Functionality

The primary interfaces for this type of system are to other banks for fund transfers and use of their teller machines.

Banks also have existing systems for clearing cheques, etc, that are jointly created amongst several banks.

It is highly likely that a banking system will make use of telecommunications services from external providers.

Human Interaction Functionality

Information systems generally have to cope with a wide variety of user types. For a bank the following list covers many of them:

- *General public* – Must be able to use automatic teller machines and, increasingly, online facilities via the web without any prior training, *i.e.* the user interfaces must be intuitive.

- *Counter staff* – Must be able to use the system quickly in order to provide a fast and efficient service to customers queuing up. These counter staff will require training and the most important aspect of this type of interface is that it should be "slick" when the staff have been trained.
- *Managers at various levels* – Some managers may not be as computer literate as the counter staff (although, of course, some may have been promoted after becoming proficient as counter clerks). The facilities to be provided for the managers may include some of the counter staff facilities, but will include more summary types of information derived from looking at a wider set of information than a single account. These may include statement summaries or branch or area business summaries. Note that these types of information demand that information is retained over a period of time so that trends and other historical information can be deduced and/or presented.
- *Policy makers and marketing staff* – Require different capabilities, perhaps introducing the capability to start new business products.
- *System maintainers* – Must be able to update system facilities. Ideally they should be able to do this while the system is fully operational, but in practice they may take down all or part of the system (usually for a brief period in the middle of the night) in order to guarantee integrity of data.

Safeguard Functionality

Security in banking systems is of paramount importance. The key element is the need to protect the integrity of the information that is at the heart of the business.

Obvious mechanisms used include the personal identification numbers (PINs) on credit or debit cards and encryption for transfers between branches, teller machines, etc.

Other areas that must be considered are the need to keep the systems working in the presence of computer faults, power failures or communication failures. These categories of functionality are related to the perception of risk. The degree of protection that can be afforded to mitigate the risks depends critically on the exposure that is perceived.

Finally, and most importantly, it is necessary to consider threats from hackers, embezzlers or others with fraudulent intent. The software must provide adequate protection to safeguard the bank and its customers from these threats.

System Transactions

Each type of user is likely to be a stakeholder in this category of system. Therefore, it is likely that there will be a set of use scenarios for each type of user. For the bank customers these include regularly used facilities such as withdrawals, deposits and transfers, whether made in person or done automatically as direct debits, salary payments, etc. There will also be other transactions used less frequently such as negotiating a personal loan or a mortgage.

For each type of user it is worthwhile considering the load that will be imposed, so that the response time can be estimated. Of course, this will not be a fixed time, but will depend on the current loading and this, in turn, will depend on the time of day and day of the week.

Increasing use of web-based facilities must add a further dimension to load prediction.

Modes of Operation

The predominant mode of operation will be the normal mode. However, there may be additional modes to cover training, backup and recovery operations and system evolution.

6.2.4 Car Example

The second example addresses a more physical type of system, but it is interesting to see that the same categories of information are still present, although in an entirely different form.

The big issue in this example is whether the system model is a physical model and to what extent it can become abstract. It is unlikely that a new car is going to be radically different in architecture from previous models – it will still have a wheel at each corner, an engine, a gearbox, suspension, a windscreen, etc. For this reason, the system model for a car may well make reference directly to the physical objects of the architecture as indicated in Figure 6.7. The arrows on this diagram indicate "some influence" in the direction of the arrow. The driver presses the brake pedal and the brake pedal activates the brakes. The connections between the body and the parts fastened to it are shown as double-ended arrows to indicate that there is a dependency in both directions, *e.g.* the engine is fastened to the body and the body has mountings to take the engine.

However, where aspects of the new car are likely to be rather different – such as in an electronic vehicle control system – remaining more abstract will present

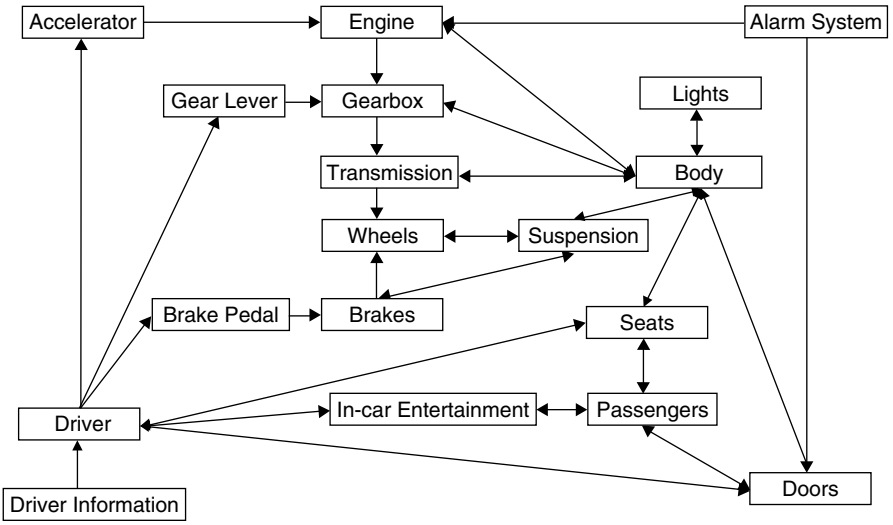

Figure 6.7 An abstract model for a car based on physical objects.

advantages in determining the best solution. To the extent that the functionality of a car is fairly well understood, what is required is to quantify the functionality. For example, it is clear that a car must be able to move people and their luggage or other items from one place to another. The key questions that should have been addressed in the stakeholder requirements are:

- How many people?
- How much luggage?
- How comfortable will the car be?
- How fast will the car travel?
- How quickly will the car accelerate?
- How much will it cost?
- What information will be provided to the driver?
- What in-car entertainment facilities will be provided?
- What safety features will be necessary?
- Where will the car be used?

Internal Functionality

The key requirements that must be addressed at the functional level include:

- *The acceleration rate of the car*. This requires a balance between the engine power, the overall weight of the car, the wind resistance of the body and the drag induced by the wheels.
- *The range of the car*. This requires a balance between the fuel efficiency of the engine, the fuel capacity, whether a manual or automatic gearbox is used and the way in which the car is driven.
- *The comfort level of the car*. This will influence cost and weight of the car plus people of different stature may perceive the end result differently.

Note that these key aspects are not independent. This is typical in a systems engineering situation. It is these interactions that tend to move the model to a more abstract level. For example, the above factors will be different depending on the type of engine and fuel used. Fuel types include petroleum, diesel and liquefied petroleum gas (LPG). The fuel efficiency and weight of engine, fuel and fuel tank are different in all three cases. Consequently, it is necessary to determine:

- whether to make a selection at all at this point, or
- whether to keep all options open or
- whether to provide a customer selectable option for one, two or three of these types.

The nature of the design will be significantly affected by the decision(s) that are taken. It may be that multiple options are evaluated, each more detailed than the overall model. Alternatively some options, for example LPG fuel, could be eliminated right at the start.

Interface Functionality

One might expect that a car is going to be isolated in terms of its need to interface with other systems. Increasingly this is not the case. One trivial example is that a car will usually have a radio receiver and this entails conforming to certain standards of demodulation in order to decode the transmitted signals.

As sophistication increases, so there are wider sets of standards that must be conformed to. For example, cars that have GPS navigation must understand how to receive and decode the satellite signals on which this system depends. Cars that can provide traffic information to drivers must be able to interface with the relevant information providers. In future, it is possible to envisage that the navigation system may well be influenced by the traffic information and hence a further (internal) interface will be introduced.

For modern cars, the way in which they are serviced is an important consideration. Frequently cars are required to retain information about events during their operational use so that the service technician can access it to aid in diagnosing problems or to guide him to check or change relevant items that are either faulty or nearing the end of their useful life. This is an example of a test system that is partly installed in the operational system (*i.e.* the car) and partly installed in the garage where the maintenance operations are undertaken.

Human Interaction Functionality

Many aspects of the "user interface" of the car are set by conventions that have evolved over the years. For example, the relative positions of the foot pedals (accelerator on the right, brake to the left and, if present, clutch to the left of that) are identical all over the world.

Other aspects, such as left-hand or right-hand drive and position of indicators and windscreen wipers, have local conventions in different geographical areas.

On the other hand, for entertainment systems, navigation systems and other less common systems there are, as yet, no agreed conventions and therefore the designers are free to provide an interface of their choice. As with most electronic systems, there is a need to make the interface easy to use (or even possible to use) for anybody who needs to use it. This is a challenge, because the only explanation that can be provided is a user guide. It is not possible to send drivers and passengers on training courses and it is not appropriate to make any assumptions about the educational level or experience of those who may need to use it.

Safeguard Functionality

The primary safeguard functionality in cars is to ensure the safety of the car and its occupants. A further, increasingly important, area of functionality is the prevention of theft.

Safety functionality starts with the braking system. It is essential that effective braking is available to the driver at all times. Dual-circuit hydraulic brakes that provide redundancy such that braking is still provided after any single hydraulic failure is one way of providing this. The system model could include the implementation directly; alternatively, the model could just include the need for braking.

In the latter case, the fact that braking must still be available in the event of a single hydraulic failure must be added outside of the model.

Note that this discussion has tacitly assumed that braking will be effected using hydraulics! Some aspects of detailed design can be included especially where there is a well-established precedent, or the decision can be taken in response to a business objective introduced into the input requirements by the developer organization.

Other areas of safety include ABS braking and the provision of air bags to cushion the impact of a collision. Again these can either be explicitly included in the model or the designer can be given freedom to invent alternative solutions.

The starting point for security is the provision of locks on doors. This can be enhanced by the provision of an alarm system and engine immobilizer. The limiting factor here is the cost of introducing the extra functionality and the amount that a customer is prepared to pay for it. However, there are other factors such as the facilities provided by competing cars and the attitude of insurance companies. Both have a strong influence not only on the functionality that must be provided, but also on the way its inclusion is justified.

System Transactions

There are many possible transactions for a car. All are based on journeys but with specific objectives or characteristics, for example:

- Driver, shopping trip in town – leave parking bay, travel, park, secure vehicle, unlock vehicle, load vehicle, leave parking bay, travel, park, unload, secure vehicle.
- Driver, motorway trip.
- Driver, airport trip (with luggage).
- Driver, trip with accident.
- Passenger – get in, fit belt, travel, undo belt, get out.
- Garage technician – repeatedly service, with major/minor intervals.
- Owner – buy, depreciate, sell/dispose.
- Salesman – repeatedly attempt to sell, ended by selling, warranty period.

Each of these may add new requirements such as luggage capacity or maintenance facilities. Therefore, it is important to consider them all and understand how the implied requirements of each are addressed. Of course, this does not mean that all of them will be satisfied. It may be that some are rejected because they are too expensive or are not considered to be relevant for the market being considered. Alternatively, the transactions may cause different models to be created for different markets.

Modes of Operation

One could imagine a car in which the prevailing terrain could influence the way in which the car operates. For example, in mountainous terrain, the gearbox could automatically select lower ratios and the engine management system would take

into account the amount of oxygen in the air and consequently alter the mixture of petrol and air to take account of this. Alternatively, these could be options that could be selected either at the time of purchase or when driving.

A further important mode of operation is the maintenance mode, in which the engine management system is downloading the collected information for analysis by the maintenance system and technician.

A more extreme mode could be to join a motorway "train" composed of a set of cars all travelling at the same speed with minimal spacing. The cars would then be controlled as a group and local driving facilities would be disabled.

6.2.5 Deriving Requirements from a System Model

Create a Document Structure for the Requirements

As indicated earlier, the system model may be composed of many independent and potentially overlapping models. It is possible to start deriving requirements from any of these models as has already been alluded to in the previous sections covering the banking and car examples. However, the challenge is to find a structure into which all of these derived requirements can be placed such that every requirement has an obvious place in that structure and that any empty sections are empty by design rather than by accident. The structure is referred to as a "document structure" in Chapter 4.

It is recommended that one of the models be chosen as the primary source for generating the document structure. The model selected should be the one with the widest scope, since the system requirements must cover the complete system and not one small area. In practice, the decision is usually obvious. For data-oriented systems such as the banking example, the data model is often the best focus, since every function is concerned, to some extent, with establishing, disseminating, updating or safeguarding the data. For more physical systems such as the car example, it is often best to use a model derived from the physical structure of the system (assuming one exists), because most of the requirements refer to one or more of these elements.

Derive or Allocate Requirements

Once the structure has been agreed, it is possible to collect all the requirements that have been derived and to place them in the structure. It may be possible to allocate some input requirements directly to the document structure. Where this is the case, it frequently means that the input requirements are too detailed, *i.e.* too close to the implementation.

All the rules for writing good requirements outlined in Chapter 4 should be observed when formulating requirements. Remember that the golden rule is to write each requirement as a single testable statement. As each requirement is formulated, it is necessary to establish traceability back to the one or more input requirements that the newly derived requirement satisfies wholly or partially.

When considering testability, it may be worthwhile thinking about the criteria that will determine whether a test is considered successful or not. These acceptance criteria should be documented with each requirement. Sometimes the criteria

can be embodied as a performance clause within the text of the requirement. An alternative is to write the criteria in a separate attribute alongside the requirement.

As testability and performance are being considered, it is vital to consider how the testing, or other demonstration of successful implementation, will be organized. This leads naturally into the issue of qualification strategy and the identification, in outline, of the set of trials tests and inspections that will be necessary.

In this context, it is also essential to consider the test harnesses or special test equipment that will be required. These may require separate development and, in some cases, can become separate projects in their own right. A further consideration in this area is the notion of built-in tests and the provision of monitor points. Built-in tests are increasingly important, especially in safety-related areas. For example, in the car example, most electronic systems will have a built-in test that is performed when the car engine is started up. Monitor points are places where significant information can be made available that would otherwise not be visible. A simple example of this is an oil pressure gauge on cars. An information example for the banking system could be a display screen showing the current transaction rates across the whole of the banking network.

The final set of requirements that should be considered is the set of constraints. These add no additional functionality, but control the way in which the functionality is delivered. At the systems requirements level, there may be some constraints that come straight from the stakeholder requirements. For example, the space that a system can occupy may be limited or the stakeholders may have insisted that a pre-existing system is used as a subsystem in the new system.

Some other sources of constraint are:

- ***Design*** *decisions – e.g.* the decision to have a dual hydraulically operated braking system.
- ***The application*** *itself – e.g.* that the equipment must be able to cope with the vibration generated by the car when it is in motion.
- ***Safety*** *– e.g.* how can the developers convince the authorities that the car will not constitute a hazard to other road users?
- ***Manufacturing*** *– e.g.* can the car be manufactured using the existing facilities at a reasonable cost?

6.2.6 Agreeing the System Requirements with the Design Team

The final step in the creation of the system requirements is to agree the requirements with the team who will be responsible for developing the design. This uses the agreement process described in Chapter 2 and therefore no further explanation is necessary.

6.3 Engineering Requirements from System Requirements to Subsystems

The logical next step on from the creation of the system requirements is to produce a design architecture whose components are the major subsystems of the

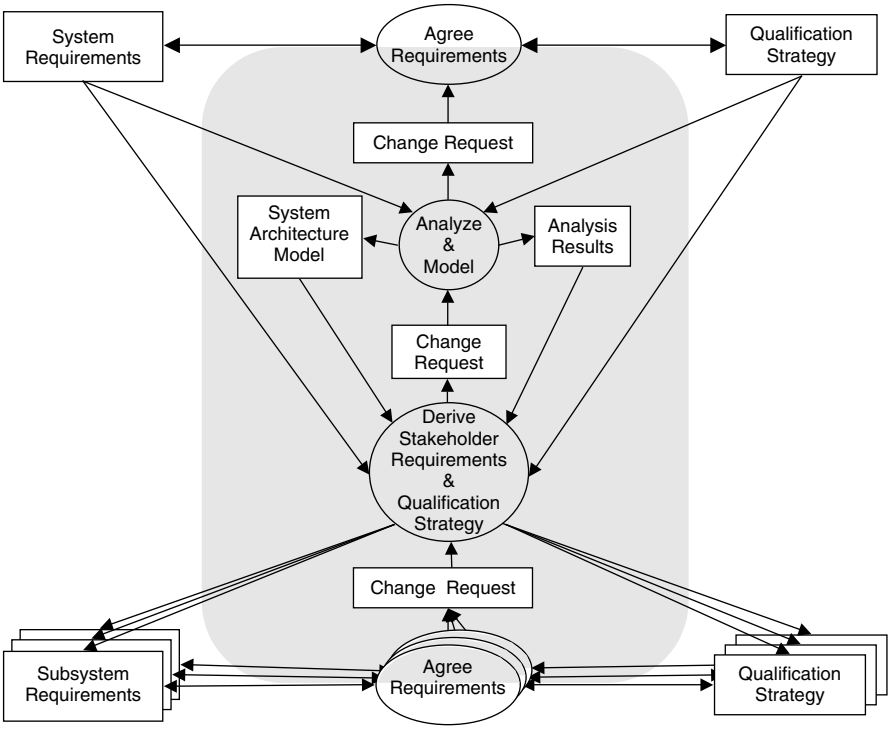

Figure 6.8 Instantiation of generic process to create subsystem requirements from system requirements.

proposed system, as shown in Figure 6.8. As usual, the process starts off by agreeing the input requirements with the customer. The review criteria for system requirements must be used as the basis for the agreement process together with the general criteria presented in Chapters 2 and 4. The requirements should be free from implementation bias unless there is a specific need to constrain the design. In the latter case the requirement must be explicitly stated as a constraint. All too frequently constraints are assumed because "that is what the customer asked for". It is always good practice to challenge any design constraint, especially if the constraint is implied rather than explicit. Sometimes requirements are expressed in design terms owing to laziness and because engineers have a tendency to go into too much detail too soon.

The analysis work necessary to support the agreement process helps to educate the designers about what is intended and starts them thinking about possible solutions.

6.3.1 Creating a System Architecture Model

An architecture model identifies the components of the system and the way in which they interact. The designer must understand how the components work together to develop the emergent properties of the system, *i.e.* to indicate how they satisfy the input requirements. The designers must also be able to predict

whether there are any emergent properties that are definitely not required, such as catastrophic failures or other safety or environmental hazards. There may, however, be emergent properties that a given design generates that, although not actually requested by the customer, may be perfectly acceptable. These additional capabilities must be discussed with the customer. They may give rise to changes in the input requirements to request them, or the customer may request that they are inhibited. Finally, the designers may find that it is impossible to satisfy the requirements at all or at reasonable cost.

It is only when a design architecture has been generated and explored that these possibilities come to light. Once a design exists it is possible to predict the cost and development time for a system with much greater accuracy than earlier. Hence it is possible to enter a round of negotiation with the customer to hone the input requirements by the customer making concessions where problems or cost dictate the need.

In many circumstances, it is worthwhile considering two or more alternative designs and then investigating the relative merits of each. Again this can lead to further negotiation (trade-off) with the customer to determine the most appropriate options in terms of cost versus benefit.

When an agreed architecture has been established, each component must be described in terms of its internal functions and its interaction obligations with other components and with external systems.

6.3.2 Deriving Requirements from an Architectural Design Model

From the descriptions of the components, requirements can be derived. The requirements must address the functionality that the component must provide, the interfaces that it must use or provide and any constraints to which the component must conform. These constraints may come directly from the overall system constraints (*e.g.* a particular electronic technology must be used for all components), or they may be derived from them (*e.g.* the overall weight limit for the system has been divided amongst the components). The component (*i.e.* subsystem) requirements are essentially the system requirements for that component when it is viewed as a system in its own right.

As each requirement is derived, it should be traced back to one or more of the input requirements that it partially or fully satisfies.

The strategy for testing each component must also be determined. This will not be the first occasion that testability has been considered. Testability is one of the most important aspects of the design and must be considered as the design is being created.

6.4 Other Transformations Using a Design Architecture

As the development proceeds from one level down to lower levels, so each level introduces its own architectural design model (see Figure 6.1). At each level the process followed is as described in the previous section. Thus the next level down from the creation of subsystems is to create the components of each subsystem and so on.

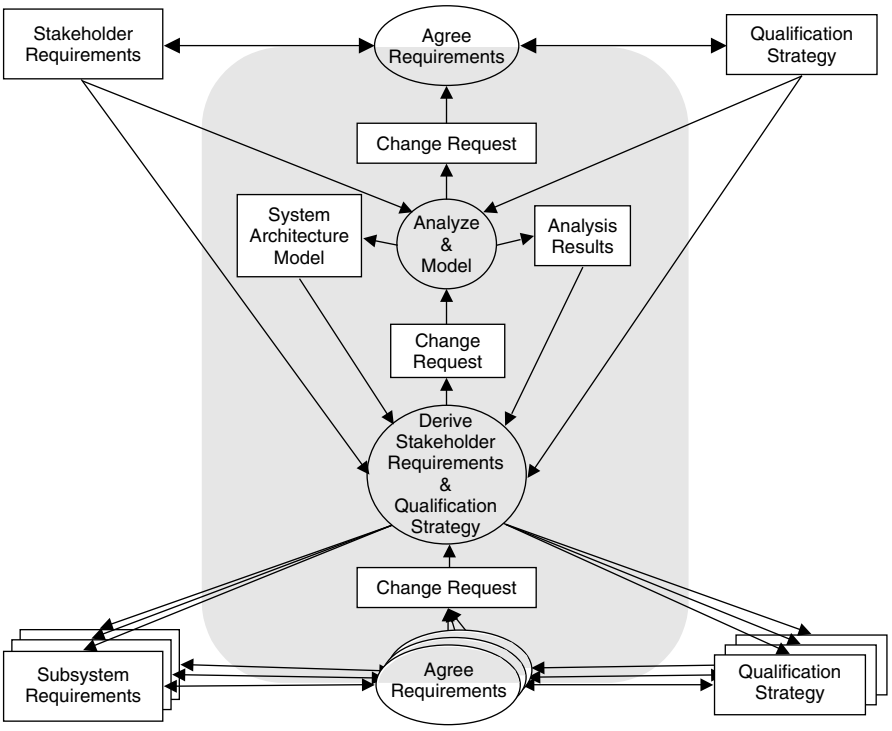

Figure 6.9 Transforming stakeholder requirements directly to subsystems.

There is one special case in which an architectural model is used that is an exception to this rule. This is indicated in Figure 6.9, which shows that a design architecture and subsequently subsystem requirements are created directly from the stakeholder requirements. This is only possible where the system architecture model is known in advance. Examples of this include some of the physical systems already considered, *e.g.* cars and aeroplanes. Another important group of applications are those in the telecommunications industry. Here the overall design architecture is mandated in the telecommunications standards that govern the application domains. It is a moot point whether the input requirements to such a process which are often taken directly from the standard are really stakeholder requirements or are, in fact, system requirements. Whatever interpretation is placed upon these requirements, during the transformation it is usual to make direct allocations from the input requirements to the subsystem requirements. Again this suggests that such standards are providing requirements at a detailed level.

6.5 Summary

In this chapter, the nature of the solution domain has been described and the way in which requirement engineering is applied to transform stakeholder

requirements to system requirements and thence to subsystem requirements and components requirements has been explained.

Two different examples have been used to explore the types of functionality that must be used to define requirements in the solution domain. It has been shown that, in addition to the required internal functionality, additional functionality must be added to cope with external cooperating systems, human interactions, to safeguard the system from external threatening systems make the system safe to use. The latter aspect may also involve additional constraints on the means of qualification in order to convince the relevant authorities.