The Characteristics of Data in Data-Intensive Safety-Related Systems

Neil Storey[1] and Alastair Faulkner[2]

[1] University of Warwick, Coventry, CV4 7AL, UK
N.Storey@warwick.ac.uk
[2] CSE International Ltd., Glanford House, Bellwin Dr, Flixborough DN15 8SN, UK
agf@cse-euro.com

Abstract. An increasing number of systems now use standardised hardware and software that is customised for a particular application using data. These data-driven systems offer flexibility and speed of implementation, but are dependent on the correctness of their data to ensure safe operation.

Despite the obvious importance of the data within such systems, there is much evidence to suggest that this does not receive the same attention as other system elements. In many cases the data is developed quite separately from the remainder of the system, and may not benefit from the same level of hazard analysis, verification and validation.

This paper considers the use of data in data-driven safety-related systems and suggests that in such systems it is appropriate to consider data as a distinct and separate component with its own development lifecycle. The paper then considers the architectural design of data-driven systems and the problems of validating such systems.

1 Introduction

While all computer-based systems make use of some form of data, some applications make more extensive use of data than others. Many safety-related systems are created by assembling standardised hardware and software components and tailoring them for a particular application through the use of *configuration data*. This approach is often adopted when using pre-existing hardware and software components, such as COTS products, allowing similar components to be used in a wide range of situations [1]. Data is also used to adapt custom-designed systems to a range of similar applications – for example, to configure a control system for a particular plant. Where large quantities of data play a major role in determining the behaviour of a system we describe the arrangement as a '*data-driven system*'.

In many cases, much of the data used by a data-driven system is *static* configuration data that is used to describe the environment in which the system is to operate. For example, an Air Traffic Control (ATC) system is an example of a data-driven system that uses large amounts of static data to describe the terrain within a section of airspace, the location of airfields within it and the characteristics of the

different types of aircraft. Here a standard ATC package may be tailored for use in a number of locations simply by changing the data describing its location.

While many systems make extensive use of static configuration data, most also depend heavily on *dynamic* data. For example, the ATC system mentioned above also uses large amounts of dynamic data to describe the position and movement of aircraft within its airspace.

It should be noted that the data within data-driven systems is not restricted to that used by the hardware and software elements of the system. In our ongoing example of an ATC system, data relating to aircraft movements is used not only by the automated systems but also by the air traffic controllers. Whether the data takes the form of information stored within a computer, or information written on a piece of paper (as in the case of the flight strips used in ATC) the correctness of the data will affect the safe operation of the system. Therefore, when considering data we must include not only the information stored electronically within the system, but also the information stored in other ways, such as within manuals, documents and the minds of the human operators! Many semi-automated systems rely on the flexibility of operators to deal with unusual or fault conditions. Clearly their ability to do this will be compromised if the information supplied to them is incorrect.

While we have used a single example (an ATC system) to illustrate the role of data within safety-related installations, data-driven systems are used in a wide range of situations, in applications as diverse as railway control and battlefield management. It is clear that in such applications the data plays a major role within the system, and the correctness of this data is essential to assure overall safety. Within the literature, and within the various standards and guidelines in this area, data is almost always considered as an integral part of software. These documents often give useful guidance (or requirements) for the development and testing of the *executable* aspects of software, but invariably say nothing about the development of the *data*. Unfortunately, data has very different characteristics from executable code and requires different techniques. At present, little guidance is available on appropriate development methods for data and anecdotal evidence suggests that data is often being largely ignored during system development [2].

In data-driven systems the data often represents a major part of the system and its generation and maintenance often represent a substantial part of the cost of the system. For this reason, it would seem appropriate to identify data as a distinct entity in such systems, and to consider it separately from the executable software. This would help to ensure that data is treated appropriately in such systems and would simplify the task of providing detailed guidance that is specific to data.

2 The Case for Considering Data Separately from Software

One reason that is often given for *not* considering data as a distinct entity is that it is an integral and essential part of software. All programs make use of both static data (in the form of constants) and dynamic data (as values within calculations) and the programs would not function without them. These elements are indeed integral to the software and this paper is *not* suggesting that such data should be considered

separately. However, some systems make use of large quantities of data that is often developed separately from the executable software. This is particularly true when a standard package is customised for a specific application using data. While the data set for the first installation may well be developed in parallel with other aspects of the system, in subsequent installations the data is usually developed quite independently. In such cases, later versions of the data are often produced with little or no input from the original development team [3].

Another reason for considering data to be separate from the hardware and software elements of the system is that it has very different characteristics. One of these differences is that the hardware and software parts of a system generally remain the same during the life of the system (unless they are modified during maintenance or as a result of upgrading) while the data element is normally subject to considerable change. Data-driven systems invariably use status data that represents the time-varying characteristics of the application and this is clearly dynamic in nature. However, the apparently static configuration data is also subject to modification as the plant or the environment changes. Indeed, in many cases a major reason for using a data-driven approach is to allow the system to easily adapt to changes in its environment. Unfortunately, the changing nature of data causes particular problems when considering the continuing safety of a system.

Data also differs from software in other ways, such as: the way it is developed; its functional characteristics; the fault mechanisms that affect it; and the verification and validation methods that are required. The use of data may also represent an alternative and distinct implementation method.

3 Selecting Implementation Strategies

When performing the top-level or architectural design of a system, engineers will partition the required functions between system resources. Some functions may be performed using a human operator; others may use mechanical arrangements while others will use electrical or electronic circuits. In many cases a computer system will provide some of the functionality and under these circumstances it is normal to partition the requirements between the hardware and the software. However, in many cases the use of data offers a third distinct implementation and functions may be partitioned between hardware, software and data, as shown in Fig. 1.

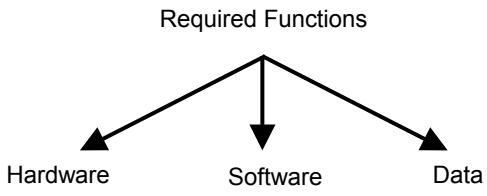

Fig. 1. Partitioning of System Functions

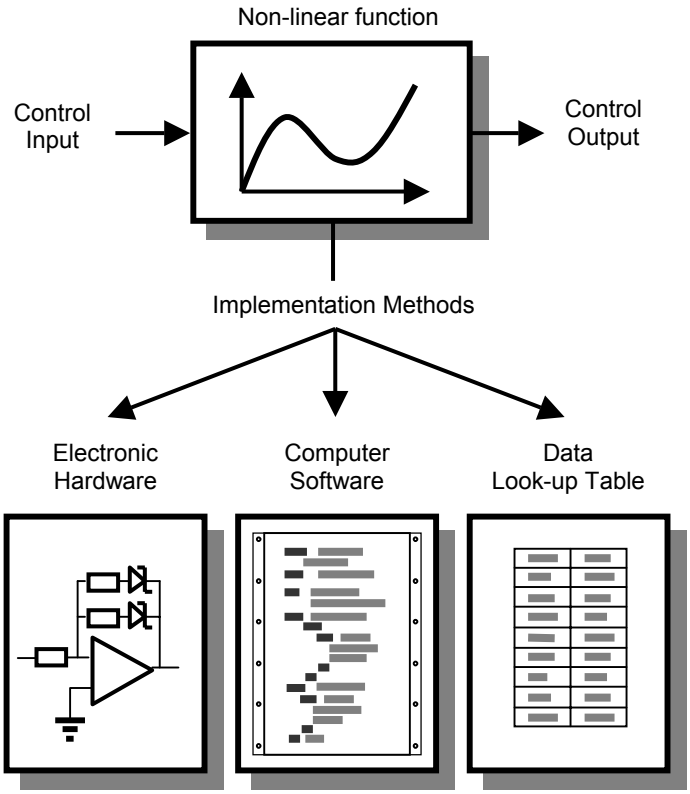

Fig. 2. Alternative implementation methods

An example of this approach is illustrated in Fig. 2, which shows part of a control system that is required to implement some form of non-linear translation of an input value. This function could be performed in a number of ways and the figure illustrates three possible approaches. The first is to use some form of non-programmable hardware, perhaps based on a non-linear curve-fitting circuit. The second and third approaches each use a computer-based technique. The former implements the non-linear function using a mathematical model that is executed within software. The latter approach stores the relationship between the input and the output within a data table. It should be noted that any computer-based approach required computer hardware in order to operate. However, where a particular function is achieved using a model implemented within a program, we would normally describe this as a *software* implementation. Similarly, when a function is achieved using a data table we can describe this as a *data* implementation even though computer hardware and computer software are required for its execution.

The reason why it is important to distinguish between software-based and data-based approaches is that these have different characteristics. When a designer assigns functions to a particular component, he or she needs to consider the characteristics of that element and how these will influence the performance and cost of the overall

system. The general characteristics of hardware and software are well understood and many designers are very experienced at partitioning between these two resources. However, the characteristics of data are less well understood and there is much evidence to suggest that the implications of data use are not always considered in sufficient detail [2].

4 A Survey of Data Development Methods

In order to investigate current development methods for data-driven systems, a series of structured interviews were carried out, with representatives from a range of industries. The survey produces a great deal of useful information and demonstrated, as expected, that data is treated very differently from other system elements. Key points raised by the survey, were:

- Data is often not subjected to any systematic hazard or risk analysis.
- Data is often not assigned any specific integrity requirements.
- Data is often poorly structured, making errors more likely and harder to detect.
- Data is often not subjected to any form of verification.

Perhaps the most striking result to come from the survey was the total lack of any uniform approach to the development or maintenance of data. In many cases engineers had not considered data in any detail and had no specific strategy for dealing with the particular problems that it presents.

5 A Data Development Lifecycle

Many of the problems identified above stem from the fact that data is often ignored during the hazard and risk analyses stage of system development. If data-related hazards were identified at an early stage then the conventional methods of requirements capture and requirements traceability would ensure that these hazards were appropriately dealt with.

One way of ensuring that data is treated appropriately throughout the development process is to consider it as a separate entity having its own development lifecycle. This would not only ensure that the various stages of the process where applied to the data, but would also simplify the task of specifying requirements and giving guidance on the process of data development. International standards such as IEC 61508 [4] give detailed guidance on many aspects of the development of software and suggest a range of techniques that might be appropriate for each stage of the work. However, this standard says almost nothing about the methods or tools that would be appropriate for the production of data. The use of a separate lifecycle for the data within a system would simplify the task of giving such guidance.

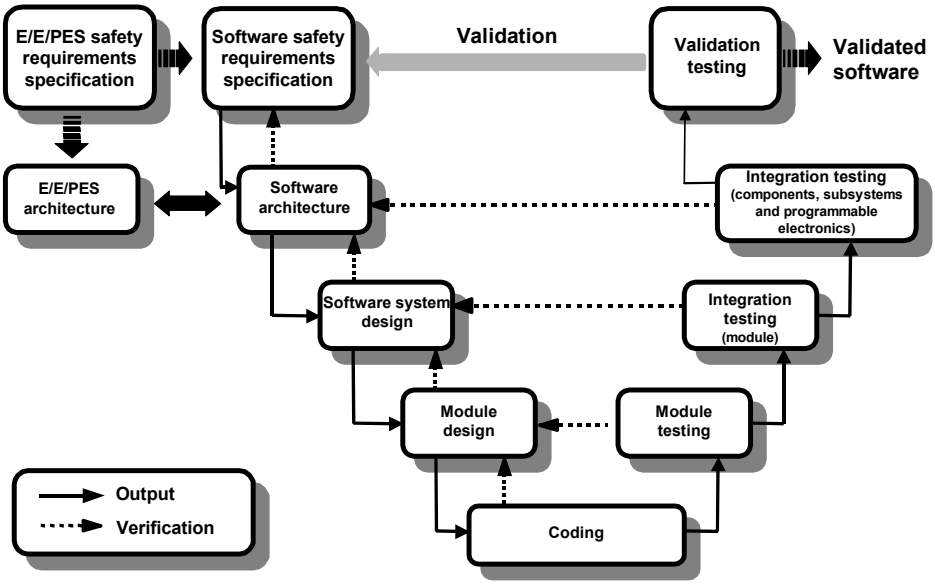

Fig. 3. A software development lifecycle model from IEC 61508

A wide range of lifecycle models is available and engineers differ in their preferences in this area. IEC 61508 uses a generic 'V' lifecycle model to describe the development of software and this is shown in Fig. 3. The top left-hand corner of this diagram represents the system level analysis of the system and the partitioning of the system to form an appropriate architecture. Those aspects of the system that are assigned to software are then developed as described in the remainder of the diagram.

It could be argued that the model of Fig. 3 could be used directly for the production of data, or that a separate lifecycle model is unnecessary since this model already includes both executable software and data within its various elements. While this might be true, the model of Fig. 3 does nothing to emphasize the role of data within the system and does not highlight any aspects that are directly related to data. Given the identified lack of attention being given to data, it seems appropriate to take some 'affirmative action' by defining a unique lifecycle for data.

In such a model, the 'module design' associated with software development is replaced by 'data structure design' in the case of data development. This phase represents a key element in the production of data-driven systems and is required to format the data in such a way that it may be verified at a later stage. Many current systems store data in a completely unstructured way making any form of verification impossible. This phase should also consider whether any form of fault detection or fault tolerance is necessary to achieve the required data integrity requirements.

The 'coding' associated with software is replaced by the process of 'data generation or collection' required to populate the various data structures.

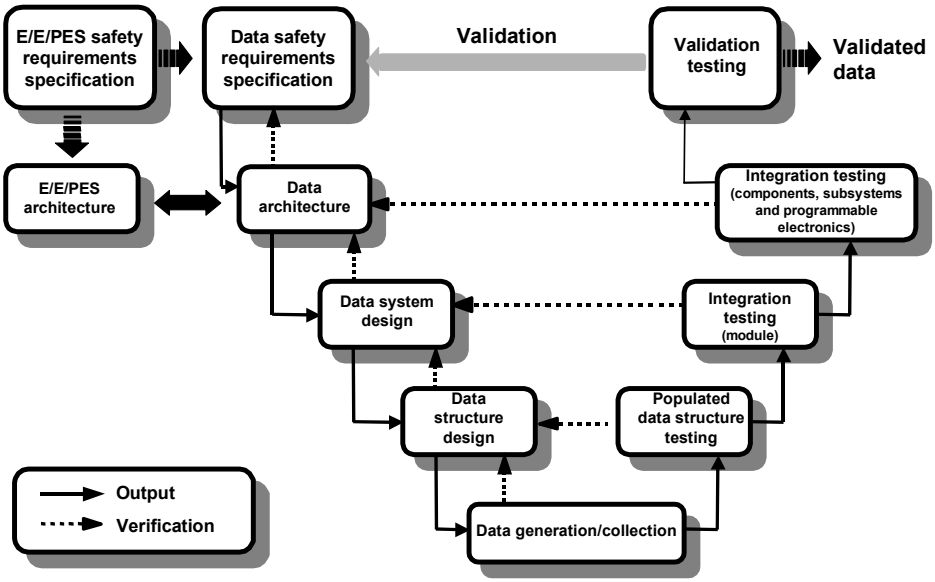

Fig. 4. A lifecycle model for data development

A possible development lifecycle for data is shown in Fig. 4. This is based on the software model of Fig. 3 with appropriate changes to reflect the different nature of data. As with the development of all safety-related components, key elements of the model are the various verification and validation activities.

6 Verification and Validation in Data-Driven Systems

Within data-driven systems we are concerned with both the verification and validation of the data, and with the validation of the overall system.

While few standards have anything significant to say about data, one standard is specifically concerned with this topic. This is RTCA DO 200A [5], which is concerned with the processing of aeronautical data for use in air traffic control. However, this standard is concerned only with the communication and manipulation of data before it is presented to its target system. It does not look at the generation of data or the use of appropriate data structures. This standard assumes that the data supplied to the communication chain is already validated, and relies on 'end-to-end' verification of the chain to guarantee the integrity of the communication process. The actual production of aeronautical data is covered by a companion standard RTCA DO 201A [6], but this standard is primarily concerned with data requirements (in terms of such factors as accuracy, resolution and timeliness) and says very little about the selection of data structures or data validation. Thus very little published information is available on data validation and we must look elsewhere for guidance.

The difficulty of validating a system increases with the integrity requirements of that system. IEC 61508 states target failure rates for the safety functions within systems of different safety integrity levels (SILs) and these are shown in Table 1.

Table 1. Target failure rates for systems of different safety integrity levels from IEC 61508

SIL	**High demand or continuous mode** (Probability of a dangerous failure per hour)	**Low demand mode** (Probability of failure on demand)
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

It is generally accepted that it is not possible to demonstrate that a system meets these target failure rates by testing alone (except perhaps for low SIL systems) and so confidence must be gained from a combination of dynamic testing, static testing and evidence from the development process.

Over the years the various safety-related industries have gained considerable experience in assessing the safety of software-driven systems. A wide range of static code analysis tools are available to investigate the properties of software and standards such as IEC 61508 give detailed guidance on which development techniques are appropriate for systems of each SIL. Using these techniques, together with appropriate dynamic testing, it is possible to have reasonable confidence that: the target failure rates have been achieved; that the system has an appropriate integrity; and that the hazards associated with the safety function have been identified and dealt with.

The situation with regard to data-driven systems is very different. Logic would suggest that the target failure rates for data-driven systems should be similar to those of software-driven systems of equivalent SIL. However, few static tools are available to investigate the correctness of data, and little guidance is available on the development techniques appropriate for system systems. Given this situation it is difficult to see how a system developer can reasonably demonstrate that their data-driven system will satisfy the requirements of IEC 61508.

Another problem associated with data-driven systems relates to their dependence on changing data. When a safety-related system is modified by making changes to its *software*, it would be normal to revalidate the system to ensure that the changes have not affected its safety. This would suggest that when using a data-driven system, the system should be revalidated whenever changes are made to the *data* being used. In practice this is impractical since some elements of the data are changing continually. This would seem to suggest that the original validation of the system should prove that the system is safe for *any* combination of data. In fact this is rarely the case and the safety of the system is crucially dependent on the data used. This leaves us with the problem of how we validate this data.

We noted earlier that data-driven systems use many different forms of data. Dynamic data is often continually changing and it will never be feasible to revalidate the system each time a value alters. Safety must be achieved either by ensuring that the data is correct by validating it before applying it to the system, or by validating the data in real-time within the system. In many cases a combination of these two techniques is used. For example, an operator may be responsible for checking the data that is input to the system, and then some form of reasonability testing will be performed to check these entries. In such cases it is likely that the overall safety of the system will depend on the skill of the operator and the ability of the system to detect errors in the input data.

In many situations dynamic data is supplied to a data-driven system from another computer network. This allows the possibility that errors in this external system, or in the communication channel, will lead to incorrect or corrupted data being supplied. Again, validation must be done in real-time to ensure safety.

When using dynamic data it may be possible to validate the data in real-time to ensure that this is correct. However, a second requirement is the validation of the overall system to ensure that it is safe for any valid combination of input data. In large systems this may be extremely difficult given the vast size of the input domain.

The situation for static data is somewhat different. In many cases static data takes the form of configuration data that defines the environment in which the system will operate. Here the nature of the data dramatically changes the behaviour of the system and different data sets will define very different systems. Under these circumstances it will normally be impossible to design the system such that it is safe for any combination of configuration data, and it will be necessary to validate each distinct implementation of the system.

When a safety-related system is modified, the amount of work required to revalidate the system depends very much on its design. A well thought-out modular approach may allow a particular module to be redesigned and revalidated with some confidence that the effects of these changes on the remainder of the system will be minimal. While the complete system will need to be revalidated, it will generally not be necessary to revalidate each of the other modules. In order for this to be true the designer must achieve ‘isolation’ between the modules such that the operation of one module does not interfere with the operation of another.

When designing data-driven systems we should aim to partition the configuration data as a separate isolated module such that it may be changed without requiring all the other modules in the system to be revalidated. If this is done successfully this will dramatically reduce the effort needed to validate each instantiation of a data-driven application. Under these circumstances it is likely that the first instance of a new system will require a considerable amount of validation effort, but future implementations will require considerably less work. However, it is essential that full details of the design and validation of the system are available to those working on later installations to allow them to effectively validate the system [3].

It is worth noting that despite the clear advantages of a well-structured, modular approach to configuration data, the survey suggests that few companies adopt such a strategy. It also suggests that validation of data is limited or sometimes non-existent.

7 Large-scale systems

As data-driven systems become larger they tend to make more extensive use of data, and the identification and management of *data* integrity becomes a significant factor in the demonstration of *system* integrity. Larger systems often form part of a hierarchy of computer systems that share data. The various elements of this hierarchy will invariably use the data in different ways, and will impose different requirements on it. Where data-intensive systems are linked to distributed information systems, the same data may be used by a range of machines for very different purposes. Under these circumstances the requirements of the data (including the integrity requirements) will vary between these machines. For example, in a railway system, data that represents the current position of the trains is used by signaling control systems (where its use is safety-related) and also by passenger information systems (where it is not).

Implicit in the development or implementation of a data-driven system is a description of the data model and the data requirements. The data model, in common with other system components, should be developed to the same integrity as the overall system. Unfortunately, experience and anecdotal evidence suggests that this is not commonly the case. Development of the data model is complicated by the fact that many control systems have peer, subordinate and supervisory systems.

Fig. 5 identifies a number of layers within a system and proposes a method of categorising these layers based upon their nature and their role within the hierarchy of the system [7].

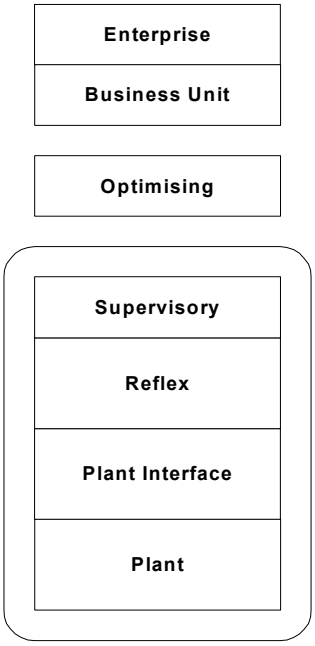

Fig. 5. A layered model for a hierarchy of systems

1. The '*plant'* layer represents single instances of elements of the plant infrastructure, the physical equipment;
2. The '*plant interface'* layer represents the interface to the plant infrastructure elements. This layer converts information from sensors (including feedback from actuators) into abstract representations such as electrical signals or data, and also produces signals to drive the various actuators;
3. The '*reflex*' layer is the lowest layer at which the measured status is interpreted and control (or protection) actions are carried out. These actions may be based upon information (which may include stored information), any demands upon the system and some set of rules. In this reflex layer the rules and information completely determine the control action. In principle all activities in the reflex layer can be automated. Where a protection system does not require the intervention of an operator these protection systems are described as reflex actions. These reflex actions are in essence rule based. Safety-critical functions often require a fast response and therefore often make use of reflex actions;
4. The '*supervisory'* layer represents a more complex level of control. This complexity may be a result of large-scale operation, integrating a number of dissimilar functions, or of interpreting complex or ambiguous data (or of some combination of these). The distinction between the reflex and supervisory layers is the judgement or knowledge that must be applied, particularly in degraded or emergency situations. Supervisory systems are characterised by the need to support the judgement of the operator doing the supervision. Predominantly the supervisory layer is downward looking, viewing the performance of the lower levels;
5. The '*optimisation'* layer represents the most sophisticated control layer. At its most developed the optimisation layer should maximise the use of resources for the delivery of the service. The optimisation layer should respect the performance and safety constraints of the underlying system. The information demands on the optimisation layer are high, requiring a full understanding of the underlying system, the planned service and contingency plans. The full understanding of the underlying system includes the performance capabilities and constraints of the various layers of the system;
6. The '*business unit'* layer represents the divisional responsibility of the delivery of service by the organisation. This layer normally plays little part in the real-time operation of the operational parts of the system, being more concerned with the medium term maintenance (including competencies) and development of the infrastructure, and the subsequent future delivery of the planned service. The business unit will become involved in the short-term operation of the system in response to a serious incident that causes substantial impact on the delivery of the service; and
7. The '*enterprise'* layer represents the corporate entity; responsible for the planning and execution of large-scale changes to the infrastructure; responding to changes in legislation; setting and maintaining standards, procedures and competency requirements.

In this work the authors propose that the supervisory layer should be the highest layer at which a safety function should be implemented. This boundary is depicted in Fig. 5 by the box surrounding the plant, plant interface, reflex and supervisory layers. The optimisation layer should take into account knowledge of current and possible future operational conditions. These operational conditions may be restrictions on the use of the plant due to planned or unplanned maintenance. Optimisation is therefore required to respect the performance and safety constraints of the system. Clearly, optimisation should only employ safe functions; that is, the optimisation of the execution of the planned service should not be capable of compromising the safety of the system.

The implementation of large-scale control systems requires a framework in which to express the role played by respective system components and provide a mechanism by which large-scale system safety may be argued. The layered model may be used to represent an organisation where several systems are used in the provision of a single function or service. This is illustrated in Fig. 6, which represents the coupling between the constituent components of an organisation.

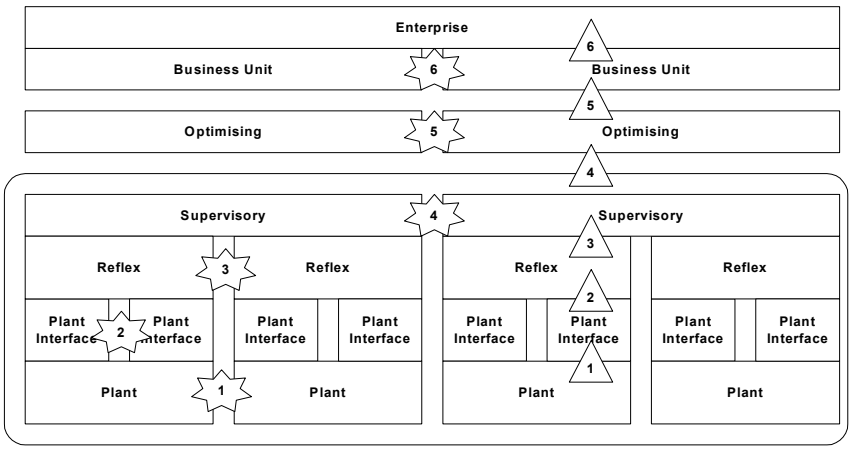

Fig. 6. An organisation employing a hierarchy of systems

The functional decomposition or partitioning of a design may be described in terms of its 'coupling'.

In this paper *coupling* is taken to be the degree of interdependence between elements of the design. A common design goal should be to minimise coupling. Low coupling, between well-designed modules, reduces the overall design complexity, creating products that lend themselves to analysis.

In a systems context, coupling is not restricted to the elements of a single design. The system will interact with its environment and quite possibly with other instantiations of the same or similar systems, or with subordinate and supervisory systems. Therefore the concept of coupling should be extended to consider vertical coupling between the system and subordinate and supervisory systems, and horizontal coupling between other instantiation of the same or similar peers systems.

Within this paper we are primarily concerned with coupling related to the exchange of data either through a shared (static) description of the infrastructure, or through dynamic data passed across the system boundary. In Fig. 6 vertical coupling is identified by the numbered triangles, and horizontal coupling is shown by the numbered stars.

A system possessing low coupling should not only possess the desirable design properties of resilience and stability, but should also have the characteristic that changes to one component should have limited effects on other components. The benefits of these properties are self evident for both good software and good systems design, and are described in many texts. However anecdotal evidence suggests that the same cannot be said of the data used by data-intensive systems. While data differs in many respects from hardware and software, it is likely that broad parallels may be drawn in terms of the desirable properties of good data design.

8 Discussion and Conclusions

Growing commercial pressures to use standardised hardware and software are leading to an increased use of data-driven systems. These are often complex in comparison with conventional computer-based systems and very often form part of a hierarchy of computers. This complexity makes data-driven systems challenging to design. There are also indications that the data at the heart of these systems is not receiving the attention it deserves. For this reason this paper suggests that data should be identified as a separate component within a data-driven system. This would highlight the importance of data to the correct operation of the system, and would simplify the process of giving guidance in this area.

An informal survey of engineers working in this area suggests that data is often largely ignored within the development process. Data is often not subjected to hazard analysis and is not assigned a specific integrity level. Perhaps for these reasons, data is often poorly structured and is not subjected to any form of verification.

Given finite economic resources, all data cannot be treated equally and other more realistic strategies are required. One such strategy would be to develop data integrity requirements that allow the targeting of development resources by a classification of risk, based upon failures due to data errors or data faults. This pragmatic approach develops the position advocated by standards such as IEC 61508 for hardware and software.

The key to successful data management lies in the use of well-designed data structures that permit and ease verification. Good design practice requires the design of components which have low coupling. Such components are usually modular, with interfaces that are resilient to changes in design. Isolation of data modules is also important since this can dramatically reduce the effort required for system validation.

References

1. McDermid, J.A.: The cost of COTS. IEE Colloquium - COTS and Safety critical systems London (1998)
2. Storey, N., Faulkner, A.: The Role of Data in Safety-Related Systems, Proc. 19th International System Safety Conference, Huntsville (2001)
3. Storey, N., Faulkner, A.: Data Management in Data-Driven Safety-Related Systems, Proc. 20th International System Safety Conference, Denver (2002)
4. IEC: 61508 Functional Safety of electrical / electronic / programmable electronic safety-related systems, International Electrotechnical Commission, Geneva (1998)
5. RTCA: DO 200A Standards for Processing Aeronautical Data, Radio Technical Commission for Aeronautics, Washington (1998)
6. RTCA: DO 201A Standards for Aeronautical Information, Radio Technical Commission for Aeronautics, Washington (2000)
7. Faulkner, A.: Safer Data: The use of data in the context of a railway control system", Proc. 10th Safety-critical Systems Symposium, pp 217-230 ISBN: 1-85233-561-0, Southampton, UK (2002)