



---

## Will it work?

Jonathan Hammond, Rosamund Rawlings, Anthony Hall

### Publication notes

Published in RE'01, the proceedings of the 5th IEEE International Symposium on Requirements Engineering

Copyright © 2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

---

## Will it work?

Jonathan Hammond, Rosamund Rawlings, Anthony Hall

*Praxis Critical Systems*  
*{jarh,rmr,jah}@praxis-cs.co.uk*

### Abstract

*This paper describes experiences using Requirements Engineering (RE) to reduce the risk of large heterogeneous distributed systems not working in their intended environments.*

*Industry is creating ever-larger systems by integrating increasingly complex smaller systems. As a result, systems integration is becoming a major, or even dominant, risk in the production of systems such as an aircraft, railway or telecommunications infrastructure.*

*In this paper, we describe some practical techniques we use for the RE of such integrated systems. They aim to provide assurance, before development, that the final integrated system will achieve its overall requirements. We illustrate the techniques with case studies drawn from their industrial application.*

### 1. Introduction

For some years the rapid development of technology, particularly computers and telecommunications, has led to systems and projects of increasing size and complexity. In our experience Requirements Engineering (RE) is being increasingly applied to systems that are built by integrating complex subsystems, ie “systems of systems”.

The relationship between the subsystems’ requirements and the ultimate goals of the integrated whole are typically complex. Undesired interactions between subsystems can also be very difficult to identify. Consequently, systems integration is a major, or even dominant, risk in the production of systems such as an aircraft, railway or telecommunications infrastructure.

Although addressing such risks is perhaps within the domain of systems engineering, RE has a crucial rôle to play. We need to be able to demonstrate that the combination of subsystems’ requirements will satisfy the stated goals of the complete integrated system. This includes understanding the operating environment of the system.

It is well established that the cost of correcting errors can increase exponentially the later in the lifecycle that the errors are found. For example, Leffingwell and Widrig [10] cite as much as a 200:1 cost saving, resulting from finding errors in the requirement stage

versus finding errors in the maintenance stage, of the software lifecycle. So a failure to set the correct requirements for every subsystem can result in a very high penalty.

In this paper we describe some practical techniques we use for the RE of distributed systems. They aim to provide assurance, before development, that the final integrated whole will achieve its overall requirements.

The next section identifies some of the principles upon which our RE method REVEAL<sup>®</sup> [12] is based and uses them to define some essential terms.<sup>1</sup> Each of the subsequent three sections describes a relevant aspect of the REVEAL method and a case study showing how it benefits a large-scale development. Finally, we provide a summary and some conclusions.

### 2. Principles and definitions

We use the generic “World and the Machine” model introduced by Jackson [7], [8] as a basis for some of the REVEAL principles.

Machines are built in order to bring about some improvement to the World. It is this real world that we wish to influence by the construction of some new Machine. In our terms a Machine can be any socio-technical artefact that is to be constructed and need not be physical; eg it could be a process. Those aspects of the real world that are relevant to the improvement that we wish to achieve are termed the Application Domain. Whenever a Machine (M) is introduced to a World (W) it interacts with that World through a shared interface (I), see Figure 1.

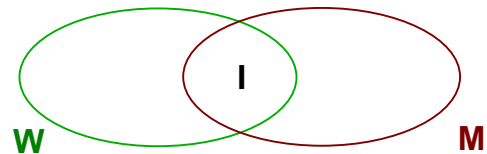


Figure 1: The World and the Machine

Items that might occur in the interface between the World and the Machine are those phenomena that are *shared* between the World and the Machine.

Phenomena in the interface are shared because they are in some sense visible to both the World and the Machine. For example, an operator may transmit some

<sup>1</sup> REVEAL is a registered trademark of Praxis Critical Systems Limited.

voice message to an airborne aircraft using a radio system. The voice message is shared between the Machine (the radio system) and the World. The Machine creates the voice message, but the World (in this case the aircraft's pilot) pays attention to it. It is through this shared phenomenon that the Machine seeks to influence the behaviour of the World. Other shared phenomena will be controlled by the World and responded to by the Machine.

The following definitions are based on those provided by Jackson [7].

*Requirements* ( $\mathcal{R}$ ) are a collection of statements about phenomena in the World ( $\mathcal{W}$ ) that we want the Machine to help make true.

A *Specification* ( $\mathcal{S}$ ) is a collection of statements that describes a Machine's external behaviour.

A specification statement refers only to shared phenomena in the interface ( $\mathcal{I}$ ).

A specification statement can only constrain shared phenomena that the Machine itself can control.

Note that a Requirement statement can also be a Specification statement, but does not have to be.

The phrases 'User Requirements' or 'User Needs' are often used to describe what we call 'Requirements', and 'System Requirements' may be used to describe what we call 'Specification'. We prefer to stick with Jackson's definitions as a basis.

Requirements are concerned with describing things that '*... we want the Machine to help make true*'. The implication is that it is not *solely* the Machine that brings about the Requirements. Other properties of the application domain independent of the Machine may also be required. Jackson calls these other properties *Domain Knowledge* ( $\mathcal{D}$ ). Note that Domain Knowledge is properties of the World that we know (or assume) to *be* true, and Requirements are properties of the World that we wish to *make* true (caused in part by the introduction of the Machine).

This recognition of the rôle of Domain Knowledge in the RE process leads Jackson to suggest the following relationship, which must hold if a Machine is to be introduced to a World to bring about some requirements.

$$\mathcal{D}, \mathcal{S} \vdash \mathcal{R} \quad \text{Satisfaction Argument}$$

The Satisfaction Argument (our label, not Jackson's) should be read as follows: using the relevant properties of the application domain ( $\mathcal{D}$ ), when combined with the specification of the behaviour of the Machine ( $\mathcal{S}$ ) to be constructed, it is possible to show ( $\vdash$ ) that the Requirements ( $\mathcal{R}$ ) will hold.

### 3. Understanding the application domain

The behaviour of the application domain ( $\mathcal{D}$ ) is the crucial connection between what a machine will do and how that will meet the requirements. For example:

$\mathcal{R}$  Allow road traffic to cross a junction safely, without colliding with traffic travelling in a different direction

is a requirement that can be met by a traffic light system. However, the specification of such a system includes statements about switching on coloured lights at particular times and in defined orders. How can lights physically stop one direction of traffic and allow other traffic to cross? The answer lies in  $\mathcal{D}$  statements such as:

$\mathcal{D}_1$  Drivers stop their vehicles at red lights.

$\mathcal{D}_2$  Vehicles can safely stop from 30mph in 10 secs.

$\mathcal{D}_3$  Drivers cross when they have a green light.

Only the combination of domain properties, such as driver behaviour, with the system specification satisfies  $\mathcal{R}$ . Of course, in this example, driver or vehicle behaviour is not guaranteed. What if brakes fail? However, by being explicit about the domain knowledge on which satisfying the requirement depends, informed decisions can be made about the acceptability of any risks of failure.

Many requirements errors arise not from mistakes in  $\mathcal{R}$  but from mistakes in  $\mathcal{D}$ . For example, Hooks and Farry [6] cite an aircraft technology program [1] where 49% of requirement errors were due to incorrect facts. Unfortunately there are many examples where failure to understand the environment of a system was potentially catastrophic. For example, Jackson describes [8] an accident where a plane overshot a runway on landing. This was due to the pilot being unable to apply reverse thrust because of an incorrect domain assumption on which an interlock depended.

Even if domain properties are accurately captured and a system initially meets its requirements, changes to the domain can mean that the requirements are no longer met. For example, the report into the Ariane 5 rocket failure [11] shows that software re-used from the Ariane 4 rocket received input values with different ranges. This was due to differing characteristics of the new rocket (ie the domain of the software had different properties) and led to the software crashing and subsequent loss of the rocket.

The 16 June 1995 New York Times reported that investigation of the 5 June 1995 NY subway crash (which killed the motorman and injured many people) indicated that the distance between signals was shorter than the stopping distance of a modern train. The emergency brake system tripped when the train ran a red signal (ie the system specification was met) but the

signal spacing was set in 1918 when trains were shorter, lighter, and slower. So the domain property that trains could stop in the space allowed after the signal was no longer true.

So describing the application domain (ie all relevant aspects of the World) is a crucial tool in ensuring that a system will achieve the stated requirements. One of the REVEAL principles is therefore:

Requirements Engineering begins with a description of the whole application domain.

One consequence of this principle is what should be shown on context diagrams. Conventional context diagram notations (eg Yourdon [14]) show how things in the real world interact with the system. But these diagrams are very constrained. You are only allowed to show things that directly interact with the system. You are not allowed to show any communications between these things unless they go through the system. Following Jackson [8] (expanded in [9]), REVEAL takes a broader view. It shows the whole application domain, including connections and properties of application domain entities that do not directly interact with the system.

### 3.1. Case study: e-commerce security

A project for a financial organisation illustrated the importance of the principle of domain description. We found that understanding relationships and processes, which *seemed* far removed from the system in question, helped us to discover the real requirements.

We were developing the user requirements for a certification authority (CA) – a crucial element in the trust which underpins e-commerce. In order to describe the requirements for the CA we modelled the whole business environment. Our model had five kinds of organisation whose business processes were supported, indirectly, by the CA. The model showed the flows of information between these organisations and indicated that just one of these organisations – the application developer – would interact with the CA system itself. Note that if we had drawn a conventional context diagram, the application developer would be the only organisation shown on our context diagram. Using REVEAL, we showed all five, including one called an application provider and one called an issuer, and the interactions between them.

We reviewed our context diagram with the client and they professed themselves satisfied with the work. They were aware, however, that the business environment was evolving rapidly and just one small thing bothered them – we might not have understood the rôle of the application provider correctly. This did not really matter – the application provider did not interact directly with the CA – nevertheless, we should go and talk to another part of the client business to check exactly what the relationships should be. At this point, a conventional approach might have said “But this is a

waste of effort. The application provider clearly isn't relevant as they don't interact with the CA. Why are we spending time on this irrelevance?”

However, we did show our context diagram to the relevant people. They took one look at it and said “But that's not how it will work at all”. Our model of the business interactions outside the CA was quite wrong and our misunderstanding about the application provider had been just a small symptom of this. That was not all, however. As a result of this reappraisal of the model, we discovered that the organisation that would need to interact with the CA was not the application developer at all, but the issuer. So not only had we misunderstood the relationships outside the CA, but because of this we had misunderstood the very nature of the CA itself. Had we not shown all these parts of the application domain, this error would not have been noticed. Following the conventional approach would have produced a plausible but wrong context diagram and would have gone a long way towards building the wrong system before finding the error. As it was, we were able to correct the error and build a new model of the domain within a day or two. The resulting requirements were used as the basis of a successful development and the system is now fielded and delivering certificates – to the right organisations!

## 4. Requirements and design

When a single requirement is to be satisfied by more than one subsystem, the detailed requirements and specifications for each subsystem must be derived from the overall requirement for the whole collection of systems. This is a *design* activity as defining interfaces between subsystems involves the internals of the Machine, see Figure 2.

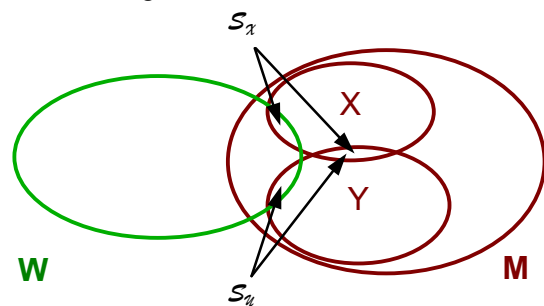


Figure 2: Subsystems

One conceptual system (the outer Machine ellipse) is being designed as a collection of smaller subsystems (the inner ellipses X and Y) plus a collection of interactions between them. These interactions are represented by the overlap common to both X and Y (ie the shared interface that is part of the subsystems' respective specifications  $S_x$  and  $S_y$ ).

The term “requirements allocation” is often used for this design process, but it is a misnomer. The requirements on the individual subsystems are derived

from the overall requirements, but they are often *not* the same as the overall requirements. In particular, the subsystems' specifications include not only their interactions with the domain (eg their user interfaces), but also their interactions with each other.

Consider a communications system, which allows pilots to talk to air traffic controllers, that consists of the following three subsystems:

1. an airborne transceiver in the aircraft;
2. ground stations;
3. a router for exchanging data between ground stations and air traffic controller terminals.

It is only the behaviour of all these subsystems combined, which can satisfy the requirement: "The system shall connect a pilot to the correct air traffic controller". Removing any of the three constituent subsystems will result in the requirement no longer being met.

So the requirement cannot be allocated unchanged to any of the subsystems. Instead, each constituent subsystem has derived requirements, such as being able to transmit or receive on a certain set of radio frequencies.

It is interesting to consider the context of just one of the subsystems, **X** say, shown in Figure 2. The other subsystem **Y** forms part of **X**'s application domain. Hence, information about **Y** is domain information (D) for **X**. So, relative to different subsystems, a single statement can be both a requirement/specification and domain knowledge. (Jackson discusses the same issue in [9] when considering subproblems.)

Moving from overall system requirements to requirements on individual subsystems requires a design activity. This design should produce:

- a definition of what the constituent subsystems are;
- an explanation of how the subsystems work together to meet the overall requirements.

A large system will need successive design activities to reach the point where the components are of a suitable size (and well-enough defined) to delegate to appropriate suppliers and/or implementation teams.

With a myriad of possible subdivisions of a large system into subsystems, someone (person or organisation) must take overall design responsibility. This means determining whether a given combination of subsystems is adequate and negotiating all the interfaces between them to obtain an efficient and cost-effective set of derived requirements.

Integrating system design with REVEAL is an iterative approach, which alternates definition of requirements with a design activity that adds structure to the system. The process steps are:

- a) define current level of requirements (eg Business, User, Performance);
- b) structure;
- c) refine requirements to obtain the derived requirements for the next level down.

The requirements at the lower level are derived (c) from the higher-level requirements (a), by taking account of the design decisions made in step (b).

As the process iterates, the balance between requirement and specification statements shifts. At the top-level requirements (R) are often not specifications at all. Specifications (S) of the systems that meet them are developed as the process progresses. Thus, by the time that responsibility is handed over to teams of developers, the requirements consist predominantly of specification statements. Such statements provide a clear basis for testing of components, and, provided they are rigorously derived, minimise integration problems. The number of iterations depends upon the problem and each stage uses notations appropriate to its content.

Scenarios are an ideal tool for exploring and describing dynamic behaviour. This behaviour might be how an application domain and system interact, eg to meet a particular requirement, and/or how subsystems interact to provide a system function. Notations such as UML activity or sequence diagrams [2] can be used to document them. Class and context diagrams can be used to record information that is more static, eg domain properties and the interfaces between entities.

Note that the above process and notations are easily applied to heterogeneous entities, as they are not dependent on the nature of the entities. For example, the context diagram for the CA (see section 3.1) contains elements for organisations, software and hardware.

The above system design approach readily establishes requirements traceability, a key component of standards for development of safety critical systems and essential for any large system. However, it is not enough to merely establish links. The adequacy of the more detailed requirements to meet the requirements from which they are derived has also to be established. This adequacy is recorded using rich traceability and is discussed below (in section 5).

#### 4.1. Case study: safety-critical protection system

We assisted a major manufacturer of safety-critical protection systems with the requirements and system design of a new system. This project shows how the above design principles can be used both to help meet the stringent engineering standards such systems are subject to, and to reduce integration risks.

The protection system is a generic product, which is configured to the particular installation. It interfaces to external user-interface systems, known as control systems, and to existing electro-mechanical equipment. It also directly supports a maintainer's interface. Configuration is particularly complex as the product is installed in different geographical locations and has to be configured accordingly.

The requirements and architectural design stages of the development used two iterations of the process described above (see Figure 3).

In the first iteration we captured the Business Requirements (step a) in a short document (about 60 requirements) for agreement with the marketing department. They were unambiguous, but not detailed. For example, they stated that the product would be capable of working with a particular external control system.

From these requirements the main functional areas (control processing, maintainer's facilities, interfaces, configuration, and suchlike) were identified (step b). We then developed a System Specification that identified the functionality that was required in each of these areas (step c).

Information about the application domain was gathered, analysed and recorded (the domain knowledge  $\mathcal{D}$ ). Then the Business Requirements were expanded and refined onto requirements in each of the identified areas.

The development of the System Specification was accomplished by working closely with application experts. A mixture of natural language and UML modelling [2] (class, state, and sequence diagrams) was used, both to show how the system interacts with its application domain and to ensure that the specification was precise and self-consistent.

A difficult problem in requirements engineering is how much detail is appropriate at a particular stage. The rule we used was 'if it might affect more than one subsystem then we must define it fully in the system specification'. If something would clearly affect only one subsystem (eg a user interface) then details (eg design of the screen layouts) could be deferred to later in the process. The System Specification contained 4 – 5 times as many requirements as its 'parent', the Business Requirements. It was thus a significant, but manageable and verifiable, step along the refinement process.

Up to this point the system was considered as a single 'black box', the structuring being only into functional areas – not into subsystems. This is particularly important – premature decomposition into subsystems confuses system specification and design. This makes it easy to lose sight of complete system behaviour, due to the distraction of subsystems' behaviours and interactions. The result is likely to be subsystems that do not work properly together, for example because functionality has been duplicated or omitted. Even where the decomposition is correct, it can be hard to establish the Satisfaction Argument because the gap between the high level requirement and the subsystems' behaviour is so large.

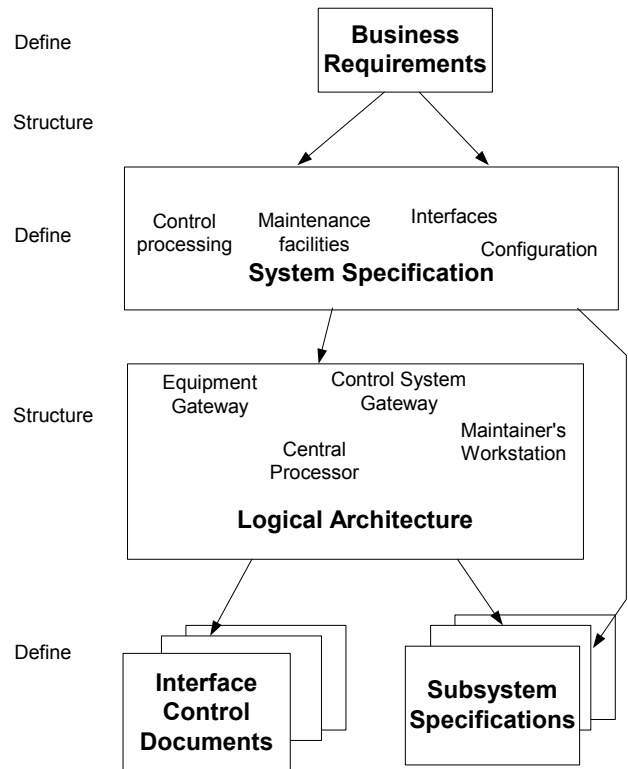
The next structuring stage identified the subsystems, allocated each subsystem a rôle, and defined the way in which the subsystems worked together to deliver the functionality. Some early conceptual work on the system provided a starting point for identifying the

subsystems, which was reinforced by considering the physical distribution of the external entities. Dividing functionality between the subsystems was based on the requirements of each external interface and safety issues, eg ensuring clear separation between the UI and core protection functions.

The dynamic application-level behaviour was the principal area of design at this structuring stage. Other documents defined the static (physical) view of the subsystems and defined the communications architecture at the transport level and below.

The Logical Architecture document used sequence diagrams to capture the interactions between the subsystems, and class diagrams to define the information needed by each subsystem to perform its rôle.

The final stage of the system design was the definition of the subsystems and the interfaces between them, producing a full set of subsystem specifications and interface control documents.



**Figure 3: Requirements and architectural design process**

## 5. Rich traceability

Traditional traceability links items from one artefact to the corresponding items from a previous artefact, or to the source of the items. For example, a specification statement may be linked to all the requirements that it helps to satisfy. However, there is no explanation or justification of the links and so their semantics is often unclear. Typically all a link between two items can suggest is that changes in one may require changes to the other.

Basic traceability can be enhanced in a variety of ways. For example, contribution structures [4], [5] extend requirements traceability by making explicit the social structure that gave rise to requirements.

REVEAL's rich traceability focuses on how an item is satisfied, by enhancing the use of simple links with additional structure and justification. For a given statement (eg a requirement) its rich traceability consists of an argument that explains:

- which statements (eg requirements, specifications or domain knowledge) combine to satisfy it;
- how/why the particular combination achieves the given statement.

Often supporting evidence is referenced, such as the results of modelling, simulations etc, or engineering calculations. For example, for the aircraft communications system (described in section 4) calculations concerning the power of the airborne transceiver and ground stations are needed to demonstrate that signals reach the ground.

This communications example uses rich traceability with multiple systems (ie a system of systems). However, rich traceability can also be used with a single system to justify that its requirements are satisfied by the system specification and appropriate domain knowledge. In this latter case rich traceability plays the rôle of the turnstile ( $\vdash$ ) in the Satisfaction Argument.

One advantage of rich traceability is that the explicit justification makes it easier to detect over- or under-engineering (ie necessity and sufficiency). Any over-engineering manifests itself as specifications (or subsystem requirements etc) that are not used 'completely' in the arguments provided by the rich traceability. This includes quantitative specifications that are more stringent than required, such as unnecessarily quick performance. Any under-engineering means that it is not possible to construct a valid justification for at least one statement.

Rich traceability (for an example see Figure 4 in the next section) has some similarities with the AND/OR goal graphs of the KAOS [3] goal-directed language and method. Rich traceability is more informal and is not usually used to support mathematically formal reasoning, although it does not preclude it. Rich traceability also includes, as part of the tracing graph, explicit justification of how or why a statement is achieved.

We have found rich traceability to be particularly useful when justifying that subsystems work together to achieve overall requirements. One reason for this is that rich traceability makes it easy to relate information about heterogeneous subsystems. A large-scale application of this is given by the following case study.

## 5.1. Case study: railway modernisation

The UK West Coast Route Modernisation (WCRM) is Europe's biggest and most complex railway construction project, with a projected investment level of £5.8 billion (1999 prices). The West Coast route links London to the major cities Birmingham, Liverpool, Manchester and Glasgow and is the UK's busiest route. The WCRM programme (the Machine, **M**) encompasses track renewal, major works to remodel junctions, renew and strengthen overhead power lines, and a new signalling system.

The WCRM is being undertaken by Railtrack, the manager of the complete UK rail infrastructure. Railtrack is responsible for everything the trains travel on, over, through, across, to and from. However, Railtrack is not responsible for the trains themselves. So for the WCRM trains are part of the World (**W**) and not **M**.

We have been working with the WCRM Systems Integration team to apply REVEAL to help:

- develop the requirements for major subsystems, eg signalling, overhead power, track and management centre;
- justify that subsystem requirements combined with domain knowledge are sufficient to achieve the overall WCRM requirements.

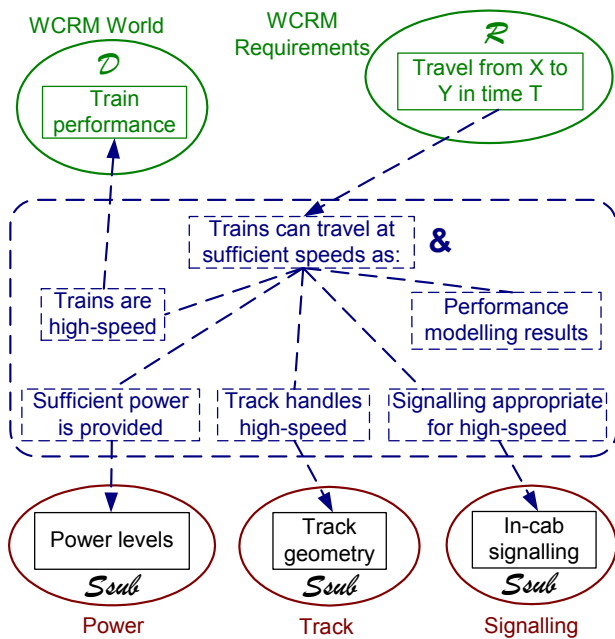
Rich traceability is the key technique that supports the justification of how overall requirements will be met. Examples of types of overall WCRM requirements are:

- reduced journey times (eg London to Glasgow in under 4 hours from 2005 onwards);
- increased capacity (eg  $X$  more trains per hour);
- enhanced safety for the public and rail workers.

Clearly, the relationship between a given journey time and requirements on the provision of power or signalling etc is not straightforward. Rich traceability enables this relationship to be broken down in a logical structure. For example, Figure 4 depicts a simplified rich traceability fragment for a WCRM journey time requirement.

The rich traceability (all the dashed objects) explains how domain knowledge and specifications of individual subsystems (*Sub*) combine to achieve the journey time. In practice, individual rich traceability items (eg 'Sufficient power is provided') often reference several requirement, specification and/or domain knowledge statements. This is not shown in Figure 4 because of space restrictions.

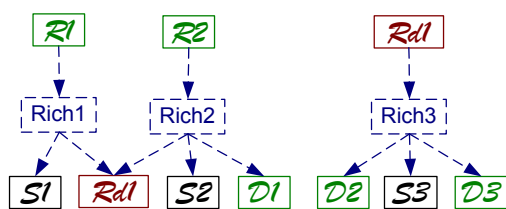
The **&** indicates that all the referenced statements must hold simultaneously. Note that detailed evidence of the sufficiency of the subsystems' performance, as defined by their specifications, is provided by modelling results.



**Figure 4: Example rich traceability fragment**

WCRM rich traceability includes references to a variety of performance, RMA (reliability, maintainability and availability) and safety models and simulations. For example, a performance model of the railway allows detailed simulations of train movements. This shows whether performance and capacity requirements are met, given the characteristics defined by the subsystems' requirements and/or specifications.

An important part of rich traceability structuring is its re-use. Rich traceability for different requirements may repeat some justification. Any repetition can be factored out by introducing a derived requirement, which is referenced wherever the repetition occurs. The previously repeated justification is given by the rich traceability for the derived requirement. Figure 5 is a simple example, where introducing *Rd1* avoids repeating Rich3 for *R1* & *R2*.



**Figure 5: Example of rich traceability re-use**

On the WCRM, justifications of different requirements often have common parts. For example, the WCRM is implemented in phases, with different journey time requirements etc for each phase. Significant parts of the justifications are the same for each phase and are re-used.

For WCRM all the requirements material and rich traceability are included in a single database using the DOORS [13] information management tool. Graphical

viewers allow specialists, in the various technical disciplines relevant to the railway, to navigate and review the material. This provides crucial assurance in the logical robustness of the justifications.

It is our belief that without the support rich traceability provides it would not be practical to assess whether the requirements on individual subsystems are appropriate to achieve the requirements of the whole WCRM.

## 6. Summary and conclusions

Our stated aim is to use RE to reduce the risk of large heterogeneous distributed systems not working in their intended environments.

First we need to understand the operating environment, which means:

- understanding and documenting the relevant facts about the operating environment, and
- using the Satisfaction Argument to ensure that the behaviour of the Machine is sufficient, with domain knowledge, to achieve the requirements.

The Satisfaction Argument is the central relationship to demonstrating that a system will work.

The relationship for integrated systems is complicated by the introduction of a number of subsystems. Achieving overall requirements (and/or specifications) by combining subsystem requirements (and/or specifications) requires design. Someone (a person or organisation) must take responsibility for this design and use appropriate techniques to capture it (eg sequence diagrams for dynamic behaviour between subsystems).

Given the often complex relationship between overall requirements and subsystem behaviour, simple tracing between these two levels is inadequate. Rich traceability provides an explicit and structured way to justify the satisfaction of overall requirements. Typically, this includes references to detailed evidence. For example, performance models, capacity simulations or reliability calculations.

We address the heterogeneous nature of subsystems by using notations and techniques that are not dependent on whether subsystems are software, mechanical, process etc. For example, the rich traceability fragment in Figure 4 relates specifications of very different types of subsystem.

In summary, we have briefly described just three of the aspects of REVEAL, which are relevant to our stated aim. These aspects are to:

1. capture domain knowledge, to make the connection between what a system does and the requirements;
2. recognise that the step between overall requirements and subsystem requirements is design;
3. use rich traceability, supported by modelling etc, to justify that integrated subsystems met requirements.

We have successfully applied REVEAL with our clients to diverse projects such as a security critical e-commerce application, a railway modernisation and even business strategies (where the business is the Machine, M).

## Acknowledgements

All the techniques described in this paper have been developed as part of Praxis Critical Systems' RE method REVEAL. We would like to thank the many members of Praxis Critical Systems staff who have contributed to its development. We also gratefully acknowledge the contribution that our clients have made to these ideas.

Thanks are also due to Jeremy Dick, of Telelogic DOORS UK, for naming our rich traceability concept.

We are grateful to Railtrack for permission to publish details of the WCRM work and for the support given by Sean Dennien and Brian Halliday.

## References

- [1] V. Basili and D. Weiss, "Evaluation of a software requirements document by analysis of change data", *Fifth International Conference on Software Engineering*, IEEE Computer Society Press, 1981, pp. 314-323.
- [2] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modelling Language User Guide*, Addison-Wesley, 1999.
- [3] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, Volume 20, 1993, pp. 3-50.
- [4] O.C.Z. Gotel and A.C.W. Finklestein, "Contribution Structures", *Proceedings of the Second IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 1995, pp. 100-107.
- [5] O.C.Z. Gotel and A.C.W. Finklestein, "Extended Requirements Traceability: Results of an Industrial Case Study", *Proceedings of the Third IEEE Symposium on Requirements Engineering*, IEEE Computer Society Press, 1997, pp. 169-178.
- [6] I.A. Hooks and K.A. Farry, *Customer-Centred Requirements*, AMACOM, 2000.
- [7] M.A. Jackson, "The World and the Machine", *Proceedings of the 17th International Conference on Software Engineering*, IEEE, 1995, pp. 283-292.
- [8] M.A. Jackson, *Software Requirements and Specifications*, Addison-Wesley, 1995.
- [9] M.A. Jackson, *Problem Frames*, Addison-Wesley, 2001.
- [10] D. Leffingwell and D. Widrig, *Managing Software Requirements - A Unified Approach*, Addison-Wesley, 2000.
- [11] J.-L. Lions, Chairman of the Board, *ARIANE 5 Flight 501 Failure - Report by the Inquiry Board*, 19 July 1996, <http://www.esa.int/htdocs/tidc/Press/Press96/ariane5rep.html>
- [12] Praxis Critical Systems, *REVEAL - A Keystone of Modern Systems Engineering*, 2001. (Freely available by sending a request by email to: [reveal@praxis-cs.co.uk](mailto:reveal@praxis-cs.co.uk))
- [13] Telelogic DOORS tools, <http://www.telelogic.com/doors>
- [14] E. Yourdon, *Modern Structured Analysis*, Prentice Hall, 1989.