

System Composition: Constructive Support for the Analysis and Design of Large Systems

R.Geoff. Dromey

Software Quality Institute, Griffith University,
Nathan, Brisbane, QLD., 4111, AUSTRALIA.

+61 7 3875 5040

g.dromey@griffith.edu.au

ABSTRACT

When confronted with a statement of requirements how should we proceed? Our task always is to use this information as a *starting point* for gaining a deep and accurate understanding of what is required. When we examine carefully any non-trivial statement of requirements what we discover is that they almost invariably contain three kinds of information: *compositional requirements*, *data requirements* and *functional requirements*. More advanced/complex systems may also contain *structural requirements*. Creating integrated views of the compositional requirements, the data requirements and the functional requirements provides an important foundation for undertaking subsequent design steps. Here will show how the concept of *systems composition* provides an important vehicle for creating an integrated view of compositional and data requirements of large systems. There are a number of benefits from creating these integrated views including quickly gaining an understanding of the nature, shape and size of a system as well as finding defects. The integrated view also provides detailed knowledge about individual components and what they contribute to a system in terms of the data they hide, the data they exchange and the behavior they exhibit. System composition provides a vehicle for distilling from diverse requirements documentation key information that can then be represented in an integrated way that provides important constructive support for subsequent design steps.

KEYWORDS

System composition, systems analysis, requirements analysis, behavior trees, requirements engineering,

“Mere information by itself is worth little, unless it is arranged in ways that make sense to its possessors, and enables them to act effectively ... To make sense of information – to understand it – one has to put it into fruitful relationship with other information, and grasp the meaning of that relationship; which implies ... seeing the whole”.
A.C.Grayling

1. INTRODUCTION

One of the greatest challenges systems and software engineers face is how should they first proceed when they are initially given documentation supplied by a customer describing their needs for a system. For a start, this documentation might vary significantly in its level of detail and size. On top of this, the system being considered may vary widely in its scale and area of application. The documentation might take the form of tender documentation, or a statement of user requirements or an operational concept document and a statement of requirements, or a detailed set of system requirements and a software requirements specification or something else.

Whatever documentation is provided, our goal is always to use this information as a *starting point* for gaining a deep and accurate understanding of what is required. Desirable as it might seem this goal is not enough because it tells us nothing about *how* we should proceed, or *what* we should produce as a result of using this information.

What we suggest is needed is a process, that when applied by different people, to the same initial information, will produce the same outcome. Counting the number of words in the original documentation is a process that would satisfy this requirement. However, the outcome that it yields would do little to advance our understanding of what is required.

In contrast to this, what we suggest is needed, is to identify is a fundamental property of every constructed system that is also a fundamental property of the documented needs or requirements for a system. If such a property, or even more fortunately, if a set of such properties exist then they should be able to be exploited constructively as we progress towards designing and implementing a system that will satisfy the requirements expressed in the original documentation.

Elsewhere (Dromey, 2003) we have shown how it possible to use the set of functional requirements for a system constructively. This involves building a system *out of* its functional requirements by first translating individual functional requirements, one at a time into behavior trees, then composing those behavior fragments one at a time like jigsaw puzzle pieces to create an integrated design behavior tree that satisfies all the original requirements. This formal integrated view of the functional requirements can then be systematically transformed to create the *systems architecture*, and the component behaviors (expressed as component behavior trees) of all the components that need to be integrated to satisfy the behaviors expressed in the original functional requirements.

What we suggest is the strategy of building a system out of its functional requirements needs to be complemented by a parallel supporting constructive design strategy of building a system out of its compositional and data requirements (that is, its static requirements) expressed in the original statement of requirements. It is these compositional and data requirements that yield a property – which we call the *system composition* which is common to both the original documented requirements and the systems design (and also the system implementation). By this we mean that if the original documented requirements were complete and defect-free then they would yield exactly the same “system composition” as a corresponding analysis of the system design. The following figure expresses the relationships between system composition, the original statement of requirements and the system design.

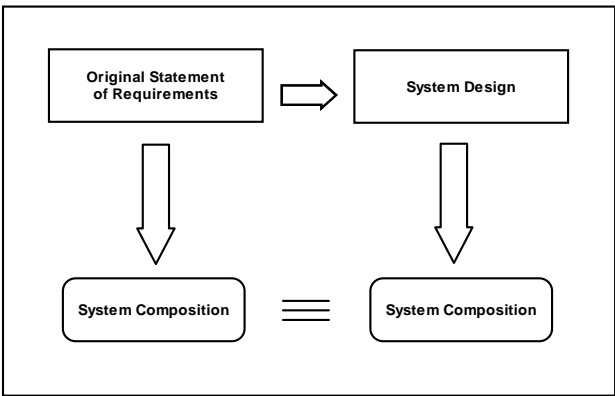


Figure 1. Relationship between system composition, requirements and design.

2. SYSTEM COMPOSITION

Composition is a concept that is used in a number of disciplines to provide useful summary information about an entity. Probably the most well known example of the use of composition is in housing advertisements which talk about things like a house having “four bedrooms, two bathrooms”, etc – that is, they refer to the “room composition” of a house. Anyone who has studied chemistry at school will also be familiar with the concept of atomic composition of chemical compounds. For example, ethane has the composition, C₂H₆.

HOUSE: 4 bedrooms, 2 bathrooms, ...
Ethane: C₂H₆

Composition is also commonly used in constructing definitions of concepts. For example, consider the following definition of a “Table”. Much of the definition focuses on the table’s components: the *legs* and the *top*.

DICTIONARY:

Table: " A piece of furniture consisting of a flat top set horizontally on legs"

What we are suggesting here, is that the idea of composition is also highly relevant when it comes to analyzing, designing and formally representing systems. It can quickly tell us a number of important things about the system being contemplated – including its relative size and what type of system it is. For systems, the concept of composition is more involved, but the principle is essentially the same. What is important about composition, even for systems, is that it should be unique for a given statement of requirements. The challenge is to define composition in such a way that this constraint is honoured.

The definition of system composition we will propose is built upon the components in a system. However to be able to accommodate large as well as small systems and very diverse types of applications we need to augment the primary focus on components with a secondary focus on contextual information. Our position is that building a substantive understanding of the components in the system creates an important platform from which to incrementally build the deeper understanding of a system needed to construct an effective design.

The problem we face in attempting to build an understanding of the components in a system is that ***in statements of requirements for a system, information about individual (and the set of) components in the system is usually widely spread/dispersed throughout the set of requirements***. In some cases important information about a particular component might be spread over tens or even hundreds of pages of text. A key role of system composition and its representation using a *composition tree* is to consolidate, integrate and document the scattered information about each of the components in the system mentioned in the requirements. This reorganization and integration of the original information does much to reduce overload on the short-term of software engineers. That is, by representing the original requirements information in an integrated composition tree we eliminate the accidental complexity introduced by the original sequential dispersed textual representation and thereby reduce cognitive demands needed to comprehend the requirements.

The primary role of a composition tree is therefore to document (extract) the following information (if it has been supplied) about a component from a statement of requirements:

- **Inputs** a component receives from other components
- **Outputs** a component sends to other components
- **Information** a component hides/encapsulates

- **States** a component realizes
- **Multiplicity** of a component relative to others in the hierarchy
- **Relational Behavior**/interactions the component participates in
- **Definitions** for the component and its associated information: inputs, etc

The second key role of a composition tree is identify and accurately define (in compositional terms) the **composite entities**/components in the system. Imposing this structural constraint on a collection of components leads to the formation of a component hierarchy – hence the construction of the composition tree. Definition in compositional terms means identifying the components that make up a composite component (e.g. we saw earlier that a table was defined in compositional terms as consisting of *a top and a set of legs*) or identifying the different types of components that a composite or abstract component may be classified into (e.g. living things can be classified into *plants and animals*). Being able to do this accurately will depend on what has been supplied in the statement of requirements. In other cases, responses from stakeholders may be necessary to resolve such matters (e.g. can a staff member work for more than one department at the same time). Often there is more than one way of dividing up the classifications for a given component (e.g. male, female versus student or teacher in a school system). Composite components may be indicative of a system within a larger system (e.g. a fuel system within an engine system). The root of the composition tree is the system component of the system being described.

Together, the information about components and their position in a composition hierarchy, that we extract from a statement of requirements add up to a specification of the compositional and data requirements for a system. Depending on the type and size of problem/system being considered different types of information will be prominent in a composition tree. The composition tree for an information system or database application will look very different to an embedded or command and control system. What is important is that the composition tree for a given set of requirements can, at a relatively small cost in construction effort tell us a great deal about the design problem we are confronting. The composition tree also serves as a representation for recording the primary **vocabulary** for a system. Once the original component information is documented, it can be refined and augmented as progress is made through the design to the implementation. It is highly likely that during the course of constructing a composition tree we will find many defects and issues/questions that need to be resolved before proceeding further with a design. Finding and resolving such problems early can significantly reduce subsequent design and implementation costs.

To summarize, our greatest need once we have a statement of requirements for a system is to undertake an analysis that will significantly enhance our understanding of the problem that needs to be solved and provide a formal representation that can support subsequent design steps. Derivation of the *system composition* from the requirements and its formal representation in a *composition tree* is an effective strategy for this purpose. It gives us a systematic method for separating out and representing the compositional and data requirements of a system. ***The goal is to make system composition a unique property of a set of requirements and composition tree construction a repeatable process.***

3. PRINCIPLES OF COMPOSITION

Our goal in construction of a composition tree from a given set of requirements is to achieve repeatability. That is, given the same statement of requirements, different people will come up with composition trees that are equivalent. A set of rules for forming a hierarchy are needed to make this possible. They follow from the most fundamental ideas on knowledge representation and concept definition. In practice, when we seek to represent the composition of larger systems, it usually turns out that more than one method for hierarchy formation needs to be used to completely construct a

composition tree. We now will use examples to introduce the ideas on composition hierarchy formation for systems.

3.1 Structural Dependency

Many man-made and natural composite entities or structures are composed of or built out of simpler identifiable components. For example, a table has one or more (+) legs and a top.

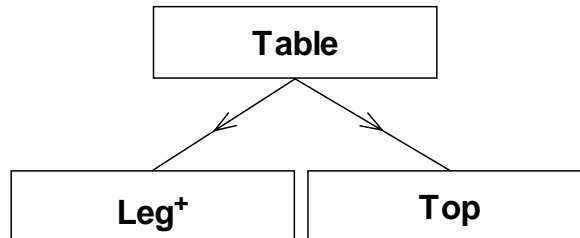


Figure 2. Composition tree for a table.

In some cases the components themselves are composite entities that are made out of or constructed from other entities. Structural dependency is used primarily in a composition tree to describe physical structures or composite data structures that need to be modeled. In the latter case, there may be multiplicity involved. For example, a Table (data structure), may contain a set of Entries. Ultimately the components in a structural hierarchy play a role in helping the composite entity at the top of the hierarchy fulfill its intended purpose or function. For example, a table without legs, ceases to be a table.

The Object-Oriented modeling technique (Booch, et al, uses the concept of *aggregation* to describe such hierarchies. It talks about the “part-of” relationship. That is, the top *is part of* the table. Aggregation is used in conjunction with the concept of *association* to create class diagrams which in general are network structures rather than trees. Because our goal is to avoid the construction of networks we have steered clear of the object-oriented treatment.

3.2 Functional Dependency

Just as a set of components may be needed to create a physical structure in other situations a set of components is needed to construct a composite entity that is able to fulfill a particular function. Systems that fulfill a function or set of functions have both a composition and an architecture. That is, integration of components that each have a function is an effective way to construct more powerful functions. Functional dependency can be viewed from both a composition and a decomposition perspective. In expressing the requirements of a system, it is a common practice to talk about functional dependencies. As an example, take the case of the composition of a simple light system. It consists of the following components: a battery, a switch, a light and three wires.

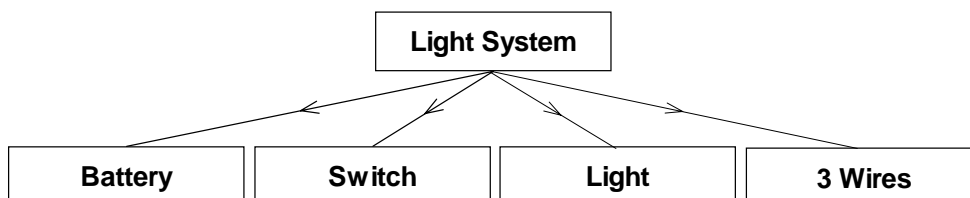


Figure 3. Composition tree for a light system

As with structural dependencies, functional dependencies can extend to a number of levels. For example, we could consider a battery to be made out of other components such as plates etc, that contribute to implementing its function.

3.2 Classification Dependency

Both in the natural world and in the man-made realm a powerful form of dependency for creating hierarchies is the use of class-sub-class relationships. For example, at one level we can classify living entities into plants and animals.

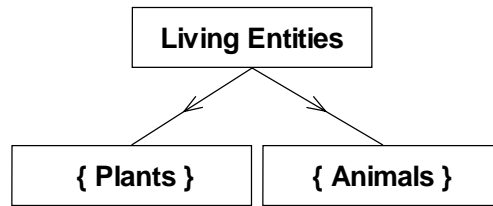


Figure 4. Composition tree for living entities

Braces { ... } are used to distinguish classification decomposition from component decomposition. (Technically, in set-theoretic terms, we are talking about bag-formation here).

3.3 Conceptual Dependency

In the management of organizations and many other human contexts people formulate what might best be described as conceptual dependencies. In order to manage, understand, implement and reason about the resulting systems people choose to construct conceptual models that are either human and/or computer-based. These conceptual models are designed to support the purpose or functions that the system needs to deliver. It is possible to arrange the components that make up these conceptual models into hierarchies using structural, functional and classification dependencies. What is challenging about undertaking this task is that the information in a statement of requirements is often not very indicative in guiding the construction of the composition hierarchy. What this opens up is the risk that different people will produce different hierarchies which is at odds with our goal of obtaining repeatability in constructing composition trees. Additional ordering criteria are needed to resolve these variations. The issues we are talking about are best understood by way of example. Consider the following data requirements for a Schools Information System (see Booch, et al, 1999).

- DR1. Students attend courses
- DR2. Every student may attend any number of courses
- DR3. Every course may have any number of students
- DR4. Instructors teach courses
- DR5. For every course there is at least one instructor
- DR6. Every instructor may teach zero or more courses
- DR7. A school has zero or more students
- DR8. Each student may be a registered member of one or more schools
- DR9. A school has one or more departments
- DR10. Each department belongs to exactly one school
- DR11. Every instructor is assigned to one or more departments
- DR12. Each department has one or more instructors
- DR13. For every department, there is at most one instructor who is chairperson
- DR14. An instructor can be the chair of no more than one department
- DR15. Some instructors are not chairs of any department

Figure 5. Schools information system data requirements (after Booch, et al)

Data/compositional requirements like these are typically represented using either a class diagram or an entity-relationship diagram (see Booch, et al [2], pp.72-73,111). The approach we take is quite different. Just as for a set of functional requirements (which we translate individually to behavior trees and then integrate) we translate each individual data/compositional requirement to a composition tree fragment and then integrate the fragments to produce a composition tree for the system – this ensures direct traceability to original requirements and hence helps with preservation of intention and validation. Below we show the composition tree fragments for requirements, DR8 and DR9. Requirement DR9 clearly indicates that a department is structurally (conceptually) *a-part-of* a school. Other data requirements (like DR8) express relational behavior, e.g student “*registered*” for one or more schools. Relational behavior is integrated with its corresponding component (in this case STUDENT#) *after* that component has been systematically positioned in the composition tree. In this particular case, because the relational behavior indicates that a student does not “belong to” a single school, it follows that the

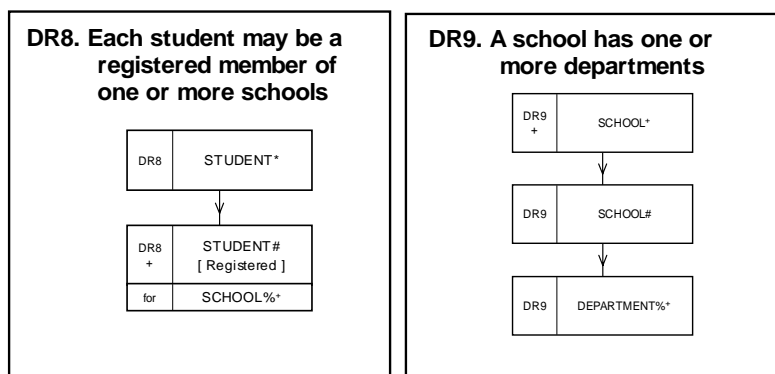


Figure 6. Translated data requirements as composition tree fragments

STUDENT* (zero or more students) component should be at the same level in the composition tree as the SCHOOL* component. That is, we sometimes need to use a test for “is-part-of” (or more accurately, “is-not-part-of”) in combination with relational behavior to systematically position a component in a composition tree.

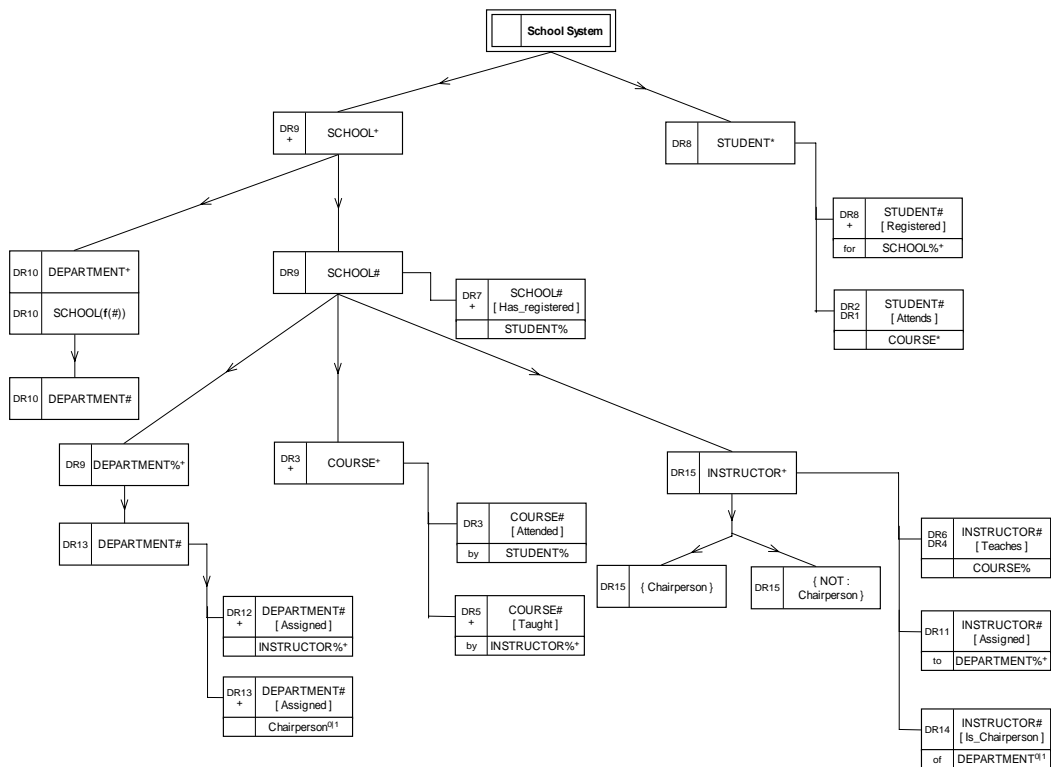


Figure 7. Composition tree for schools information tree.

As we mentioned earlier DEPARTMENT% (some departments) are clearly part of each SCHOOL# (per DR9). This brings us to the issue of positioning the component INSTRUCTOR*. Using a similar line of reasoning to that used for STUDENT*, because DR11 indicates each instructor is assigned to one or more departments INSTRUCTOR* needs to be placed at the same level as DEPARTMENT*. (This positioning rests on the probably reasonable assumption that the assignments are made to departments within a single school in each case – this would need confirmation with those proposing the requirements). So what we see here again, in the absence of other information, is that relational behavior can help to systematically and repeatably position a composite component in the composition tree hierarchy. When we come to position the component COURSE* we see there is no indication in the requirements that a given course is offered by a single department. However requirements DR1, DR2, ... , DR6 all mention course in relational behavior. Below we cite a rule for resolving the position of a component just on the basis of relational behavior. In our inclusion of COURSE* in the composition tree we have chosen to make the assumption that a course may be offered by a number of departments within a school – this would need to be confirmed with the author(s) of the requirements (Booch, et al, also make this assumption even though the requirements neither state nor imply it). As the position of each component is settled we can integrate in the relational behavior associated with that component. This representation strategy avoids the creation of a network as happens with class diagrams that use associations to record relational behavior.

The composition tree obtained is significantly different from the corresponding class diagram for the data requirements listed above (see Booch, et al, [2] pp 72-73). In the class diagram STUDENT is seen as forming an “is-part-of” relationship with SCHOOL. This does not seem consistent with the requirement that a student can be a member of more than one school (see Booch, et al, p.73). These

sort of anomalies are avoided using the ordering rules for composition trees summarized below. These rules represent a form of normalization for composition trees.

3.5 Determining a Component's Position in the Composition Hierarchy

1. The primary determiner of position of a component in the hierarchy is the "is-part-of" relationship. This has precedence over the secondary determiner of position. Three things determine the "is-part-of" relationship (a) structural dependency (b) functional dependency, and (c) classification dependency.
2. The secondary determiner of position of a component in the hierarchy is the relationship a component forms with other components (via relational behavior) that already have their position in the hierarchy. The rule is that if an "is-part-of" relationship is not provided then place a component at the same level in the hierarchy as the highest placed component it forms a relationship with.
3. In practice, a client (or author of requirements) can usually provide answers to questions that will uniquely place a component in the composition hierarchy of a composition tree. (e.g. in the School System above, they will be able to tell us whether or not a course could only be offered by departments in a single school). When all else fails the primary and secondary determiners of position are sufficient to achieve repeatability in systematically placing components at a level in the composition tree.

4. COMPOSITION TREES – CASE STUDIES

To gain a deeper understanding of composition trees it is advisable to study a diverse range of applications both small and large. Here we only have space to give brief consideration to two examples. While with smaller examples the differences may appear minor or cosmetic, it is with the analysis and representation of larger systems where the advantage of using composition trees really becomes apparent. What is important with larger systems is how composition trees are able to do four things:

- to provide, at relatively small cost, an overall integrated picture of a large system;
- to consolidate for each component in the system, strategic information that is widely dispersed throughout a statement of requirements;
- to incorporate structural information that positions each component relative to others and informs us of important indicators that play a key role in characterizing the architecture of the system;
- to easily adapt to capturing the different kinds of information that is emphasized in diverse types of applications and by different standards used to construct initial statements of requirements.

From this great diversity the composition tree enables us to always transition to an informative integrated view that provides a platform for understanding even the most complex of systems. This simple representation can inform both developers and clients/users alike.

4.1 Case Study [1] – Video Store System

The Video Store case study has been used by Basili, et al, to show a method for conducting software inspections. The statement of requirements for the case study runs to 13 pages.

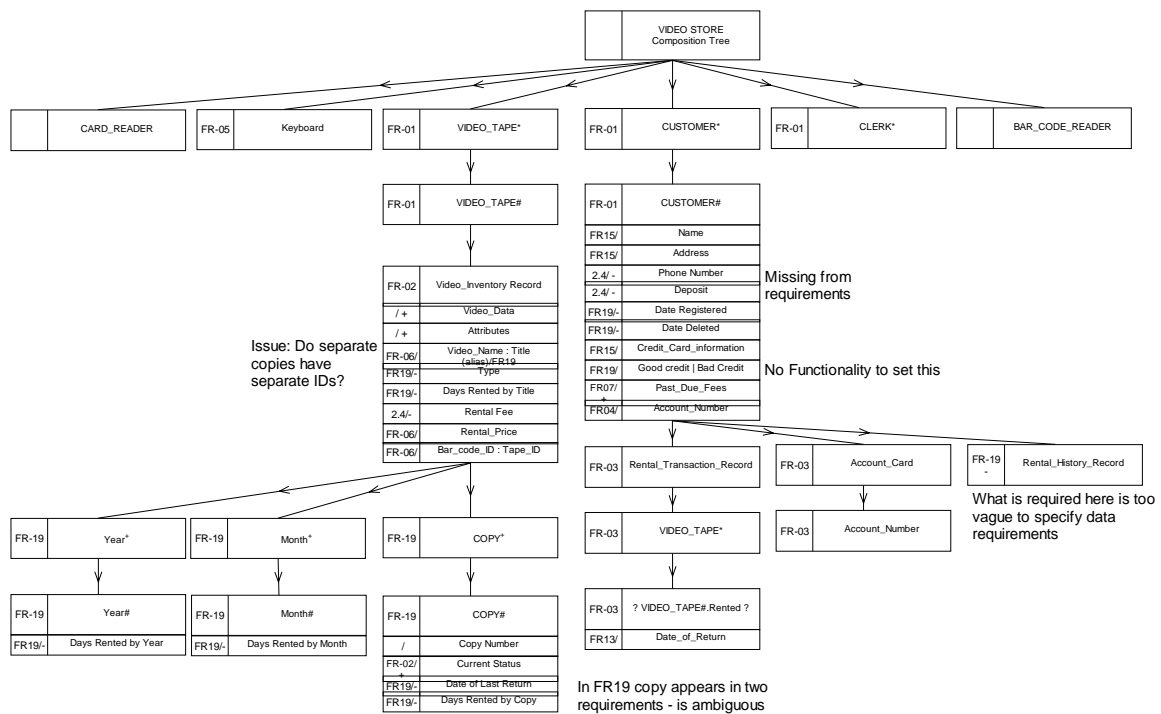


Figure 8. Composition tree for video store system.

We have used this document to extract the data requirements from the statement of the functional requirements and construct the corresponding composition tree for the Video Store case study (see above). As each data requirement is integrated into the composition tree we have included the corresponding original functional requirement reference (e.g. FR-19) to maintain direct traceability.

This example clearly illustrates:

- That information for individual components is widely dispersed throughout a set of requirements. For example, the figure shows the customer component gets its data requirements from *nine* different functional requirements in the original document.
- That a focus on data requirements often reveals missing functionality needed to capture and process that data. If we only discover these types of defects later at the implementation stage they can be costly to fix. Reporting functional requirements (see FR-19) are notorious for this – which happens with this case study. They often imply a lot of complex processing and data requirements that are missed in stating the functional requirements. In general data requirements often imply functional requirements which may sometimes be missing. The complementary situation also applies. Functional requirements often imply data requirements (which need to be recorded or which may be missing)
- Some data requirements are too vague to indicate what information needs to be stored to satisfy the functional requirements (see FR-19, Rental History Records).
- A number of aliases are revealed, e.g. video-tape and tape, etc
- Sometimes important requirements are not listed in the formal statement of requirements but are found in other parts of the requirements document (e.g. see reference to section 2.4 above)

The Composition Tree is able to provide the direct traceability to original statements of requirements. What this reveals for this case study is that a single component may have its data requirements spread across many functional requirements. Another clear message from this example is that an integrated view of data requirements often reveals considerable missing functionality needed to

manage those data/compositional requirements. A consolidated view of the data associated with each component also tells us a lot about what internal processing capability (functionality) is needed by a component to manage that data.

There is one other perspective that the composition tree of the Video Store brings out. When we examine the composition tree for the case study what we see is that it is dominated in its form by the stored data encapsulated within the main components of the system. What this indicates is that data is being stored to record behavior and support queries that may be made of the system. This style of stating requirements is typical for what are usually classified as database applications. This contrasts sharply with some of the other examples. Even though this example, might be regarded as a database application it is quite different in form from the School Information System considered earlier. In that example, while behavior was recorded, it was predominantly relational behavior that involved more than one component. So what we are seeing is that there are two types of recorded behavior, behavior that pertains to a single component and behavior that involves more than one component, and hence an interchange of information between components.

4.2 Case Study [2] – Satellite Control System

The Satellite Control System case study (Ett, 1998) consists of an original statement of requirements that runs to more than 20 pages (note (Prowell, et al, 1999) also contains a discussion of a different version of this case study). The form and the amount of detail and the type of information provided is more typical of the statements of requirements that we see for larger systems. In sharp contrast to the Video Store case study the dominant information recorded in the composition tree is the information exchanged among the components and the definition of the structure of that information. There are three things that we want to use this case study to illustrate. First, that input and output information for a particular component can be widely dispersed among a set of functional requirements. Second, how consolidation of input/output information for each component can show up inconsistencies and other problems with a set of input/outputs. At the same time, when we have completed the consolidation of the input/output information, we should have a major part of the component interface information settled and in a form that can support subsequent design steps. Third, how the compositional and data requirements for a system, as represented in the composition tree make an important partial contribution to defining the system architecture, Fourth, how construction of the composition tree significantly advances our understanding of the system being proposed, its size and the type of system we are dealing with. Below we show the details of a single component which forms part of the composition tree. It provides a significant amount of detail about a component that can subsequently be used in the design and implementation of that component.

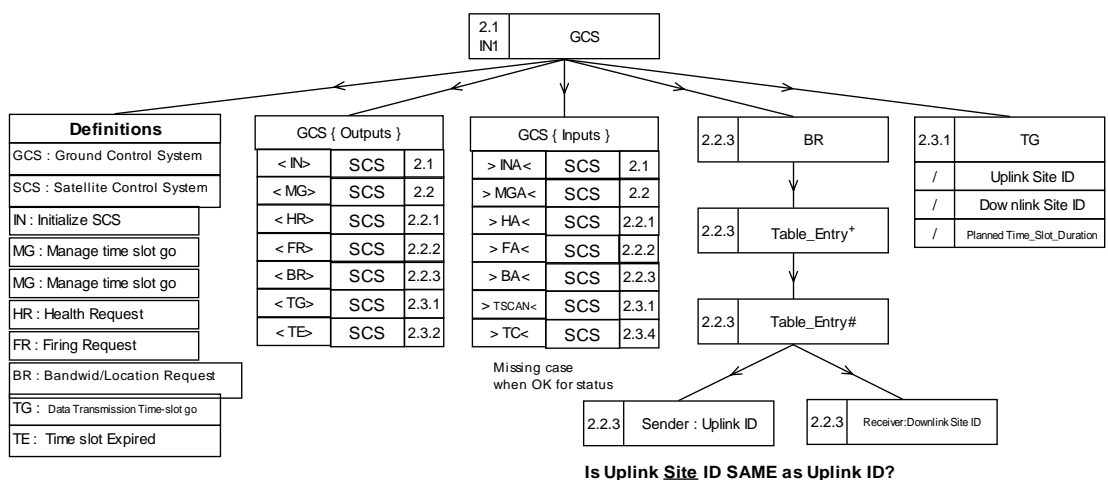


Figure 9. Composition tree fragment for GCS component of satellite system

This fragment also shows up several defects. For example, we might suspect that “Uplink ID” used in 2.2.3 is an alias for “Uplink Site ID” used in 2.3.1. All messages from the GCS receive an acknowledgement except MG. That is, an MGA is missing as an input. Also, in 2.3.1 there is a missing case for TSCAN when its status is okay.

The complete composition tree for the four components of the Satellite Control System is shown below. It consists of the GCS (Ground Control Station), the SCS (Satellite Control System), the Uplink Sites and the Downlink sites.

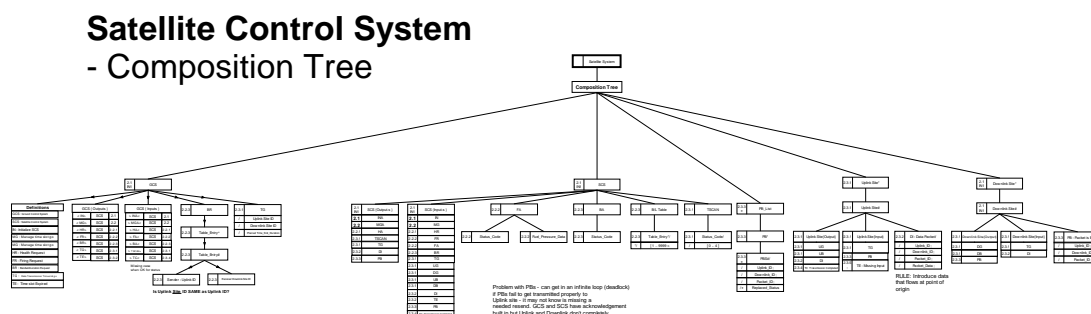


Figure 10. Composition tree for satellite control system.

What this composition tree indicates is that all of the components are on the same level (there are no hierarchical dependencies among the components) and none of them are made up of other components which means that architecturally we are dealing with a simple system. The components interact in accordance with the behavior spelled out in the requirements. In this particular case the composition tree is not telling us anything significant about the architecture other than we are dealing with a simple system. Examination of the inputs and outputs is enough to tell us that that the GCS interacts with the SCS and the SCS interacts with the Up-link and Down-link components. Each component contains a number of messages or stored data that is defined. The messages are transmitted to other components. The contents of the composition tree emphasizes that the focus of this system is on the exchange of messages between components. The information gathered for each component tells us a lot about the component that will prove useful when implementing the component.

5. CONCLUSION

We have provided a very brief introduction to the concept of system composition and its representation using composition trees. Composition trees have been put forward as a vehicle for creating an integrated view of the compositional and data requirements of large systems. Organisation and representation of compositional and data requirements in this way has a number of advantages. It provides a first step to getting the complexity of a large system under control. It also yields an estimate of a project’s size while at the same time providing constructive support for subsequent component and system architecture design. Experience has shown that the integrated view that the composition tree creates allows us to identify a range of incompleteness and inconsistency defects. Because composition trees represent defined properties that are *common* to both original requirements documentation and the formal design of a system their construction approaches repeatability. This further implies that the task of constructing a composition tree can be divided up among a number of software engineers. Our claim is that construction of a composition tree represents an important investment in a project that provides an ongoing return that far outweighs its cost of construction. It is not an analysis work-product that is constructed and then discarded. Instead

it represents important constructive documentation that can be refined and remain useful throughout the evolution and life-time of a system.

Acknowledgements

The author would like to acknowledge the support of research funding from Australian Research Council as part of the ARC Centre for Complex Systems, head-quartered at University of Queensland. I would also like to acknowledge the support of my students Lian Wen, Saad Zafar, Xuelin Zheng and my ARC Centre colleagues Peter Lindsay, Ian Hayes, David Carrington, Lars Grunske, Dan Powell, Cameron Smith, Kirsten Winter, and Nisansala Yatapanage.

6. REFERENCES

- [1] Basili, V., Green, S., Laitenberger, O., Lanubile, F., 1998, *Experimenting with Error Abstraction in Requirements Documents – Video Store Case Study*, www.di.uniba.it/~lanubile/papers/metrics98.pdf.
- [2] Booch, G., Rumbaugh, J., Jacobson, I., 1999 *The Unified Modelling Language User Guide*, Addison-Wesley, Reading, Mass.
- [3] Dromey, R.G., 2003, “*From Requirements to Design: Formalising the Key Steps*”, (Invited Keynote Address), IEEE International Conference on Software Engineering and Formal Methods, SEFM’2003, pp. 2-11, Brisbane, September.
- [4] Ett, W., Trammell, C., *Satellite Control System Case Study - Object-Orientation/Cleanroom Integration Study*, 1996, <http://source.asset.com/stars/loral/cleanroom/oo/study.htm>.
- [5] Prowell, S.J., Trammell, C., Linger, R., Poore, J.H., 1998, *Cleanroom Software Engineering*, Addison-Wesley, Reading, Mass.