العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 1: *Introduction*

Developing a software system that meets the purpose for which it was proposed is the main concern for any software developer. Requirements analysis is the first and the most critical phase of the software system development. The raw requirements can be considered as a commitment between the software system developers and the customer who requested the system under development. The developers always work hard to achieve the customer's aspirations by implementing a software system that contains all the business processes which are explicitly stated and implied in the raw requirements.

Although, the raw requirements are the most influential association between the developers and the customer, the software system’s development process does not originate from natural language raw requirements specified by the customer. The requirements specifications which are engineered from natural language raw requirements can be considered as the basis and the first step of software systems construction in software engineering.

Human knowledge and ingenuity is the only resource that can be used to define the requirements specifications and intermediate models such use cases [12]. Therefore, these requirements specifications and intermediate models do not cover the raw requirements exactly; they, at best, only approximate them [4].

The main objective of this project is to discover a systematic approach that processes raw requirements which are expressed in natural language and to extract the information that helps in constructing the component-based software system architecture directly. There is relevant research which is concerned with the analysis of raw requirements and associating them to elements that relate to software units in order to achieve a better match between the final system and the raw requirements.

Current reliance on human knowledge and ingenuity in mapping out system specifications from natural language raw requirements limits software development to the skills of an individual [12]. There is evidently a need for a better approach. This approach should be one that is systematic and relies less on individual human skill.

A component based approach that maps directly from raw user requirements in natural language to executable components is a viable answer to this problem. This is the aim of this project. This paper aims to describe a systematic approach to mapping user requirements directly into executable components.

Tied to this is the notion of architecture in which we will discuss viable and function architectural systems that allow component addition to partial architecture in an incremental fashion.

Using the approach described in this paper we should be able to use raw requirement to immediately select components from the repository or develop the components and deposit them in a repository. This would then be followed by constructing a partial architecture and then compose it with the system architecture after that returning to the raw requirements and carry on in that manner. Now clear the system must allow for extensibility and that is the detail that this paper goes into. This paper will also describe how we will use existing technologies to parse user raw user requirements for valid content and how this is then employed in deciding which component to make use of.

The foundation of this paper is that an individual raw requirement can be directly mapped to executable software components. In addition to this, it is also possible to develop a systematic manner in which to join these components together whilst allowing flexibility for any other requirement to be added into the system at any stage of development [7]. In other words, the system should allow incremental development such that one does not need to read the full natural language raw requirements before starting development.

This paper will also go in quite some detail on the XMAN tool that will be used as our component based model. The workings of this approach are discussed in some detail with a special focus on the tools available for building partial architectures and the use of components in the repository and adding components to the same.

Some of this research is based on object-oriented software development while this project approach intends to use component-based software development. The component-based development differs from the object-oriented software development in many features. This report will investigate these differences and will review the literature of related works such as behaviour tree approach and object-based mapping approach.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 2: *Software System's requirements*

In developing any piece of software the driving force are the stakeholders and the goal of developers is to supply stakeholders with software that meets their specific needs. The issue of software system requirements is concerned with ascertaining the requirements that a client has and then developing a piece of software that meets those requirements. A number of issues arise during this endeavour of ascertaining requirements. Sometimes the client does not know what they actually want. Sometimes requirements are incomplete or perhaps ambiguous. Software system requirement techniques are concerned with overcoming these challenges. We are particularly interested in how we derive formal specifications for an informal requirements document that is written in natural language. How do we map the required functionality from the language of the client to a developer perspective? This chapter touches on these particular issues, addressing the different aspects of software requirements analysis.

The Software Requirements Definition document sets out the functional requirements of the software under development [13]. This document should be drawn from a reading and interpretation of the Business Functional Requirements definition document. Before commencement of actual development work the Software Requirements Definition document must be fully documented, approved and signed by all stakeholders.

To minimise risk, no actual programming beyond conceptual demonstrations and proof of concept mockups should be undertaken until the Software Requirements Definition is approved and signed off.

As already mentioned, the Software Requirements Definition is drawn from the desired business functionality as laid out by the client. The first stage in developing the definition is in setting out a clear definition of what the software is required to do [13]. An exhaustive process of project requirements gathering must be undertaken. A detailed exploration of project requirements gathering is outside the scope of this paper.

The project requirements gathering stage gives a users perspective to the software. The developers must then examine these requirements and build a ‘logical model' by using recognised methods and specialist knowledge of the

problem domain. The logical model is a high level abstraction describing system abilities and should be free from implementation technology [14]. This model gives structure to the problem set giving it greater manageability.

The logical model is then used in the production of an ordered set of software requirements. These requirements would specify the level of functionality, detail performance, set out available interfaces, give assurances over quality and reliability etc.

This document sets out the developers' view of the problem set as opposed to that of the user. The relationship between the Software Requirements Definition document and the Business Functional Requirements is not necessarily one-to-one and often is not.

It is very important that all the stakeholders agree on one consistent view of the various requirements of the system. The development team will interpret the software requirements from the user perspective and express this in the developers view in the form of the Software Requirements Definition. There may be a disparity between the two which is why it is essential that all stakeholders approve and sign off the Software Requirements Definition document before any actual development work begins.

2.2 Requirements Types

Most software requirements can be categorised into the following: *Architectural Requirements*, *Functional Requirements*, *Non-Functional Requirements* and *Constraint Requirements* [15].The following section briefly examines each of these requirements.

2.2.1 Architectural Requirements:

This is a high level description of what must be done. It identifies the system architecture of a system that is to be developed. The architectural requirements are primarily concerned with the shape of the solution space. They establish the structure of a solution to a set of problems imposed by a set of requirements.

A distinction must be set between Architectural Instance and Architectural Family. An architectural instance gives a high level abstraction of a software system. A system architecture would describe, at the very least, how the system is broken into different components and how those components work together, carefully setting out dependencies and so on [2]. An effective architecture would expose the crucial properties of a system. Here we can define crucial as those properties that must be considered for an effective reasoning of the system to be carried out.

Contrast the architectural instance with an architectural family, an architectural family sets out constraint definition on a group of associated systems. Architectural families can be merely generic idiomatic patterns and styles (for example, "pipe and filter" or "client-server organisation") and can be reference architectures (for example, "OSI layered communication standard"). An effective architecture is one that ensures a measure of integrity constraint but also permitting a measure of flexibility that subsumes the family of systems to be built over the life-time of the product family. Architectural requirements are established by the developers and system architects, not the user.

2.2.2 Functional Requirements

A functional requirement defines the function of a software system or component [15]. Functional requirements could refer to data manipulation, calculations or data processing that define what a system is supposed to accomplish. Functional requirements are often expressed as, "the system must do<this>" and "the system must do<that>". Functional requirements are a high level expression.

It is a system/software requirement that specifies a function that a system/software system or system/software component must be capable of performing. These are software requirements that define behaviour of the

system, that is, the fundamental process or transformation that software and hardware components of the system perform on inputs to produce outputs.

A functional requirement will often have a unique name and identifier, a brief description and a rationale. The key point is the description of the required behaviour; this must be clear and easy to understand. This type of requirements is concerned by this research.

2.2.3 Non-Functional Requirements

Non-Functional Requirements are often described as quality requirements. These are characteristics of the software system that the user is not able to perceive. It is a software requirement that does not describe what the software does, but how it will do it. An example would be software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Non-functional requirements are difficult to test; as such they are often evaluated subjectively.

Non-functional requirements will often, if not always, take a descriptive tone; for example, “the system shall be <requirement>”. An example following this would the following requirement: “the system shall be <easy to navigate>”.

2.2.4 Constraint Requirements

Better addressed as “constraints,” these are merely restrictions within which the software under develop must operate or be developed under. For example a requirement that states, “software must be ready for developed before year 2000” would be a constraint. This would be a project constraint. Constraints often refer to non-functional requirements. An example would be a requirement that demands that the application "require no more than 100mb of hard drive space during installation" or that "the application must gracefully degrade on older browsers."

2.3 Attributes of Good Requirements

The IEEE standard stipulates that a Software Requirement Document must satisfy the following.

a.) Functionality - This should state clearly exactly what the software should do and, if there is scope for ambiguity, what the software should not do.

b.) External Interface - This part of the specification should detail how the system will interact with people (human computer interaction), the system's hardware and other software.

c.) Performance - These requirements regard speed, system availability, speed of response and recovery time of various functions.

d.) Attributes - These are extensibility, maintainability, security, usability, correctness, etc. considerations.

e.) Design constraints - These requirements address required standards, implementation language, database integrity policies, resource limitations, operating environment, etc.

Even after these types of requirements have been laid out it is important that they conform to the rigors again imposed by the IEEE Standard. The following are qualities of good requirements.

a.) Correct - This much is largely self explanatory. The requirements should state what is actually meant by the client.

b.) Unambiguous - The requirement must have one interpretation. Any ambiguities must be highlighted and the actual desired requirement stated explicitly.

c.) Complete - All the requirements necessary for the software to be operational must be stated.

d.) Ranked for importance - Requirements that are fundamental to the operation and success of the software should be listed above aesthetic and non-crucial requirements.

e.) Verifiable - Requirements should avoid subjectivity. Instead of requiring the software to simply be "fast," requirements should state "on form submission user should receive response in no more than 600 milliseconds."

2.4 Requirements analysis and software design

Requirements analysis is the process of investigating the properties of a specification and developing an initial software model [12]. It describes the set of tasks involved in determining the exact needs or conditions that need to be met to satisfy the requirements of a client. There are a number of recognized techniques in the literature on how best to conduct Requirements analysis. A full a in-depth examination of all these techniques is outside the scope of this paper. However this section will briefly examine some of the key techniques in use. This much is necessary for the sake of comparison with our own component based direct mapping approach that will be introduced in a later chapter.

Use Cases can be used in requirements analysis. A use case is a structure for documenting the functional requirements of a piece of software. In each use case a scenario that shows how the system will interact with humans or other systems is provided. Use cases are often developed by requirements engineers in conjunction with stakeholders. Use cases simply show the steps needed to accomplish a task, they do not show the workings of the system or how the task will actually be implemented at a development level.

Use cases are just one example of requirements analysis but already an important question arises. How do we move from have the raw user requirement in natural language and begin to move toward an more software oriented expression? How do we manage to correctly draw fromt he raw requirements what exactly it is that the stakeholders require? A great measure of this relies on human ingenuity and an understanding of the problem domain.

Semantic case analysis is an effective way of moving from raw requirements in natural language to a position where the stakeholders requirements are actually established. This is especially true in object oriented analysis of software requirements.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 3: *Mapping requirements to software system architecture approaches*

After software system requirements are determined as being clear, unambiguous and giving the developers a clear picture of the system required by the stakeholders the next challenge is to map these requirements to software system architecture. How do we take a raw requirement and translate this into a functional piece of software? Perhaps the greater challenge is not merely having the knowledge to implement a code level solution for a specific requirement but how to organise these solutions into a coherent piece of software bringing together solutions to the various requirements into a coherent and complementary system. This chapter explores the various approaches to mapping requirements into software system architectures.

3.1 Object-Oriented software development

Object-oriented software development is a development technique where a system is considered in terms of "things" or "objects" as opposed to functions or operations [10]. The system is built up by a combination of objects that interact with one another and maintain their own state and allow operations to manipulate that state. Access to information about state representation is limiting in the concept of information hiding. An object-oriented system is

designed by creating object classes and defining the relationship between some or all of these classes. The objects are not actually written explicitly by they are created dynamically from the class definitions.

An object oriented system is built from classes. These classes can be instantiated numerous times in a system with different states. Each such instantiation is called an instance. This instance is what we describe as an object.

Object-oriented design has gained wide recognition as a cost-effective and fast way to develop software. It cuts development time and overhead costs by allowing the development of highly reusable and easy to maintain applications.

Inheritance is a key concept in object-oriented system design and is one of the characteristics that make for greater code reuse. Inheritance allows classes to inherit behavior from a super class. This prevents code duplication.

As mentioned earlier, one of the benefits of object-oriented development is information hiding, commonly referred to as encapsulation. Encapsulation involves grouping data with the procedures that operate on them. An interface is then provided through which data can be accessed through various procedure but without direct access to the data. The procedures should not reveal the implementations used to manipulate the data.

One of the key benefits of the class definition is looser coupling. Using object-oriented techniques it is possible to have highly cohesive applications which still retain loose coupling. The benefits of this become clear when errors are

identified. Because an object –oriented system is so loosely coupled it is very easy to correct errors in one class without affecting the rest of the code in other classes since each class has a specific task that is assigned to it.

Perhaps one understated benefit of object-oriented development is its simplicity. The consideration of a system in terms of real world object increases the comprehension and understanding of a problem and is inherently less prone to mistakes.

3.1.1 The main features of Object-Oriented software development

Object oriented development can only be carried out using a fully object oriented language or a language that at least supports object oriented programming. Object oriented languages have all the features of other languages but in addition to this they also support key features that distinguish them.

Some of the key features of object oriented programming are:

1. *Inheritance*
2. *Polymorphism*
3. *Data Hiding*
4. *Encapsulation*
5. *Reusability*

3.1.1.1 Inheritance

Inheritance involves the inheriting or deriving of qualities (properties) from an existing class. The class from which these properties are derived is referred to as the super class whilst the class which inherits is referred to as the subclass. Inheritance becomes useful when we have a set of common features, characteristics or behavior that might be needed in a number of classes. Instead of writing these out in each and every class that they might appear they can be written in one class and simply made use of in other classes that might have need for them. This has a number of advantages which include reducing the code size, making error correction and code extension easier (since only one class needs to be changed). Inheritance also allow for reusability since code written in only one place can be reused numerous times in a piece of software.

3.1.1.2 Polymorphism

Polymorphism refers to the ability to create a variable, method or object that has more than one form. The underlying objective of polymorphism in object oriented programming is the implementation of message-passing. In message-passing object of differing type make use of a common interface. Users can then programme onto that interface. This then gives objects from different types to answer method and field calls of the same name but offering the appropriate type-specific behavior.

It is possible for these objects to be entirely unrelated but in practice, since there is a common interface, they are often subclasses of one superclass. Though not a requirement it is often expected that the different methods will produce similar output.

3.1.1.3 Data Hiding & Encapsulation

Data hiding is at the core of object oriented development principles. Data is kept hidden by declaring it as private inside a class. By declaring it as private it can only be accessed by the class in which it was defined. Public data is accessible in the whole application (outside the class). Data hiding is important because it allows for better control in a system since data cannot be manipulated from any part of the system without specifically accessing it from methods declared in that class. This makes for easier to maintain applications.

Encapsulation simply refers to bundling data and methods that operate on that data together. So though data might be private there could be a public method in the same class that carries out operations on that data. The beauty of encapsulation is that it allows a class set limitations on the type of operations that can be performed on data which prevents many errors.

3.1.1.4 Reusability

Object oriented development lends itself to high reusability. Reusability is one of the key reasons why it has gained widespread acceptance. The separation of concerns in the development of a project allows the same piece of code to be

reused numerous times in different places in the software and it also allows the same classes to be used in entirely different projects where they can be useful.

3.1.2 Object-based mapping requirement approach

This section presents the, in quite a brief fashion, one of the earliest literature on object-based mapping requirements (***Software Development Process from Natural Language Specification***). The problem set is unique because the challenge is not only to interpret, translate and map user requirements but to do this in an object oriented fashion. In this we mean to consider raw requirements in the natural language and to then map then into object oriented theme that is immediately ready for implantation in an object oriented development environment.

The foundation of the work by Saeki, Horai and Enomoto is the conviction that the lexical and semantical structures of the words used in the informal natural language raw requirements are similar to the software component in the system. Their logic, it follows, is that we can therefore read informal requirements and immediately map these into formal specifications or software components. Their fairly extensive paper proposes "the process for constructing incrementally formal specifications from their informal specifications written in English."

It is already possible (at time of publication) to extract nouns and verbs from a natural language specification document. The challenge is the inability of the computer to determine which words of relevance and which ones are not. This role is left to the developer to use human knowledge and an understanding of the problem domain to determine which nouns and verbs are useful for developing a formal specification.

An object oriented model is one in which the system is considered as being a set of separate objects that communicate with one another. The objects also have attributes which represent internal state.

The design activity proposed in this paper should produce the module design document from an informal natural language raw requirements specification document. This module design will have external design class specification such as method names, class module names and message protocols; All this being derived directly from natural language raw requirements.

The process of establishing a design module is quite straightforward:

Input product → Extract Candidates (nouns and verbs) → Select product items out of candidates → Output product [8]

In the first step nouns and verbs are selected using rules. This is followed by a developer selecting words deemed relevant.

The following rules are used in the extraction and creation of candidates:

1) Verb table: this contains a list of information about verbs that have been extracted e.g verb names, category, subjects etc

2) Noun table: Details information about extracted nouns
3) Action table: Actions are extracted from items in the verb table.
4) Action relation table: Here the relationship between the various determined actions are identified.

When determining the noun and verb table the general rule is that nouns and verbs correspond to objects or class and to messages, respectively.

Because the natural language specifications that are used are not written in an object oriented fashion there is needed for a more thorough analysis of nouns and verbs. An interesting example is in the sentence “temperature of the room”. In this instance temperature is an attribute of the room rather than an object in itself. Nouns and verbs are categorised into various categories.

Noun and verbs in the natural language raw requirements are initially identified using a noun and verb dictionary without parsing the text. It is also possible to parse the text. However it is only this much that can be automated. You need a human to select the key words and to classify.

In the action table we establish the agents and targets of all action verbs in a sentence. Action verbs will cause changes of state in their target. To find these you seek the objective words which will have their state changed.

3.2 Behaviour Tree approach

A behavior tree is a graphical tree representation of the behavior of individual or networks of entities which realise state, change states, make decisions, cause and respond to events, and interact by exchanging information and passing control [7], [8], [5].

The conventional strategy in software engineering is to use an underlying design technique to construct a design that will satisfy a set of functional requirements. However, behavior trees allow the construction of a design out of its set of functional requirements.

The underlying and core conventions of component-state notations are are the graphical forms for associating with a component a [state], ??Event??, ?Decision?, [Sub-cpt[State] or relation, or [Attribute := expression | State]. To allow for traceability with the original requirements the tagging convention is following numbering the tags R1, R2 and so on.

At times the requirements are not complete in the sense of explicitly stating a requirement. These missing or implied requirements can be expressed with a "+" in the tag. A "-" mark in a tag indicates behaviour that is missing in the natural requirements.

The behaviour tree approach translates each raw requirement in the natural language into behaviour trees. The behaviour tree is deceptively simple but it is an incredibly powerful innovation that allows manageable mapping of raw requirements. It is a graphical representation of the behaviour of sets of

components. It recognises and depicts change of state, decisions, response to events and control flow.

The behaviour trees for each individual requirement are then joined together to form a design behaviour tree.

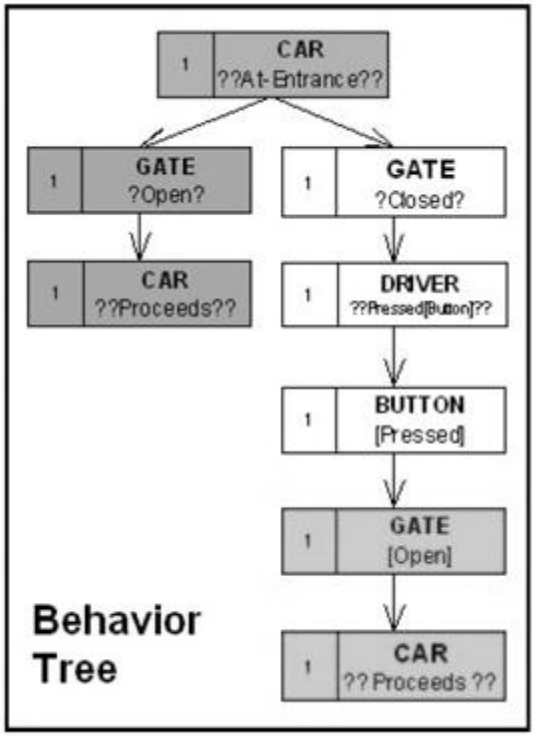


Figure 1: Daniel Powell: Requirements Evaluation Using Brhavior Trees- Finding from Industry

Behaviour trees provide a direct relationship between what is expressed in the natural language and its equivalent in the behaviour tree. This relation is also highly traceable.

Translating the requirements is the initial step in behavior tree requirements mapping. As mentioned earlier each required is mapped into a decision tree called a requirement behavior tree (RBT). During translation we identify,

among other things, components, the states they realise, events, logical dependencies. Because this is done on a requirement by requirement basis complexity is easily managed.

When all the requirements have been mapped into RBT's they are integrated by the precondition and interaction axioms. In practice this is simply identifying where the component state root node of one behavior tree occurs in the rest of the tree. Translation and integration can be done in any order.

A precondition axiom is always necessary in order to integrate requirements with at least another member of the set of requirements. Precondition axiom is necessary if we are to be able to attach each requirement to a cause (state or event). Such a linking is necessary if the behavior tree is to be used to draw up system architecture because the behavior tree is thus equipped with all the details necessary for the construction of a system. These give us clues about what additional information is needed to achieve integration.

Interaction Axiom - Each function requirement represented in a behavior tree must have the preconditions it must satisfy in order to display its behavior set by another behavior tree of at least one functional requirement that belongs to the system.

Together, the precondition axiom and interaction axiom have an crucial position is definition the interaction between a functional requirements. This allows the design of a system right from the natural language specification by building

behavior trees for each requirement and then move to integrate that set of decision trees. Following this technique we find ourselves in a four phase development process which involves

1.) Requirements Translation

2.) Requirements Integration

3.) Component Architecture Transformation

4.) Component Behavior Projection

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 4: *Component-Based Software Development*

The foundational concept in component based systems is the separation of concerns in respect to the various functionalities that are available in a piece of software [7], [3]. This separation of concerns expresses itself in components. "A software component is a software element that conforms to a component model, and can be independently deployed and composed without modification according to a composition standard." This use of component leads to faster development as already existing components can be used rather than building from scratch. It is difficult to imagine how developers can build components that will later be used in systems that they had no consideration for at the time of design. How can one be sure a component will work in their system? These very real potential problems and questions are answered by Component Models. Component Models are a sort of contract between the developers who work within that model. Component Models define what a component is, the framework within which they can be built, how they are assembled and how they can be deployed. Following a specific Component Model developers can

build components that can be used in other software as well as make use of components already written in their own software development process.

4.1 Why Component-Based Software Development?

Component based development immediately becomes attractive when you consider that most software have very similar underlying functionalities. Take a web based service for example. Most of these services maintain a login system of one form or the other. Instead of every web application developer coding their own login system from scratch would they not be better served by simply making use of a login component that has already built? It is this view of reusability that gave birth to and indeed drives component based development. Our example is limited to logins but could be extended to numerous examples of functionality that is made use of across a spectrum of very different services.

In addition to this is the matter of complexity. Developing software from prefabricated components that simply needed to be assembled in accordance with a set protocol abstracts a great measure of complexity. The developer simply needs to understand the Component Development Model, grasp that inputs of the component he wishes to make use for and he can immediately benefit from powerful computations may perhaps be outside his domain expertise.

These benefits in turn lead to additional benefits. Making use of component dramatically reduces the development time since developers need not waste time writing up code for components that are already existing in the repository. Developers become much more productive when they can focus on actually solving real problems and their time is devoted toward this endeavour. In addition to this quality is improved when component are used. This is because a component that is used by a large number of developers is under constant improvement. Bugs are quickly identified and solved. Components can be replaced in the repository and indeed in software that has already been deployed.

4.2 What are Software Components?

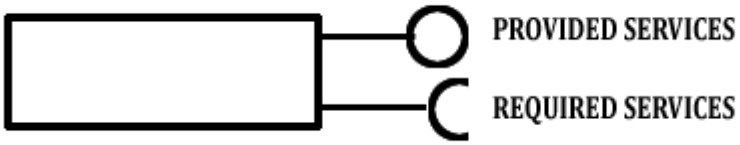


Figure 2: A component abstraction

A software component is a unit of software that offers services and makes use of other services [3]. The services that are provided are operations that are carried out by the component whilst the required services are those services that are needed by the component for it to be able to provide the services that it offers.

'"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."*Clemens Szyperski, Component Software, ACM Press/Addison-Wesley, England, (1998).*

A component will have an interface [3]. This includes the specification of the services it provides and requires. This interface ideally points out reliance (dependency) between the provided and required services. Sometime a component can have more than one interface; these interfaces being for different services.

In the object oriented case where we have components operating as objects the methods provided by these objects operate as the provided services. It is not possible for these objects to specify the services they require so they are often hosted in an environment that which handles the interaction between components.

Three of perhaps the most important characteristics of components is that they hide their implementation. Components can be thought of as epitomising the concept of black box abstraction. The developers who use the components have a full understanding the output a component can provide but they have no idea about how it actually accomplishes this. The second important characteristic is Context Independence. This means that components can be transferred from one application to the other. This is because components are by definition self contained software elements. The third characteristic is Implicit Invocation.

Since the components are exchangeable, components should not address one another directly but work via an interface.

4.3 The main differences between Object Oriented Software development and Component-Based Software Development

It is important to set a distinction between component and objects, the two are often mixed up and confused. Although component based development borrows a great amount from object oriented development techniques there are fundamental differences between them. The major difference is the measure of encapsulation [10]. Components are fully and totally encapsulated. There is no access whatsoever to the internal workings of a component. Compare this with object oriented development an understanding of the inner workings of an object is very necessary, for example when inheriting from a class. In object oriented programming the reuse is termed white box reuse since the source code is open for insight. Contrast this with the black box abstraction is component based development.

It is interesting to consider how component based systems and object oriented systems are extended. The former relies on composition rather than inheritance which is the foundation of object oriented systems.

Components have no persistent state whilst objects do. Because components have no state they can only be loaded into a system once. It makes no sense to speak of multiple components of the same in the same system because they

perform the exact same task. Contrast this with objects that have particular state and can be active in a system in multiple forms.

4.4 Component Models

"A component model specifies the standards and conventions that are needed to enable the composition of independently developed components."

Component models are a very important concept in component based development because they establish a contractually relationship between developers on how components are defined, how they can be specified and how to actually assembly them. There are a number of component models that have been developed and are in use but most of them have considerable differences in approach [3], [7]. In this section the aim is to classify some of the existing component models. Because these models are so wide in their scope a thorough examination of each model will not be possible.

The currently available component models include, but are not limited to, AUTOSAR, BIP, BLUEArX, CCM, COM, COMDES II, CompoNETS, EJB, Fractal, IEC 61499, JavaBeans, Koala, KobrA, OpenCOM, Palladio, PECOS, Pin, ProCom, ROBOCOP, Rubus, SaveCCM, SOFA 2.0 [16].

4.5 The restriction and limitation of the Existing Software Component Models

Perhaps the greatest limitation of existing component models is the fact that there are so many of them and there isn’t a single accepted standard that would allow for rapid and concentrated development within that one single model.

This is unlike object oriented development which, perhaps by chance, gained near unanimous support and is highly standardised. As it is an engineer who decides to approach a problem using component model is first faced with the difficult challenge of deciding which component model to make use of.

Looking closer at individual models one immediately notices a number of distinct limitations that are manifest in existing component models. Software components that are built within the Component Model frameworks often mix control and computation. The computation would be any operations performed by the component whilst control refers to the method calls made by a component. We make special mention of control because when one component calls the method of another control immediately moves away from that component. This is an obvious dependency which is undesirable. Although components seem to hold a great measure of specified operations in themselves they remain tightly coupled. This dependency makes it difficult to reuse components in other software systems since these dependencies would still need to be satisfied.

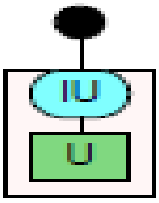
When grouping components we make use of connectors. In current component models these connectors encapsulate the communication between components. This encapsulation hides away one of the problematic areas with existing component models. If components are making calls to the methods of other components it becomes increasingly difficult to make use of that component in another software system without making significant changes to the structure.

4.6 Exogenous Connectors Component Model (XMAN)

The Exogenous Connectors Component Model is different because it makes use of exogenous connectors in the connections of all the software components [7]. The main difference between these connectors and those used in existing component models is because it is in these connectors that all the control is performed and handled. Components do not call any methods on other components. Instead it is the connectors that call methods on different components thereby freeing individual components of any dependencies. Exogenous connectors allows for very loose coupling which is a desirable benefit. The fact that components not call methods on any other component directly means that these components can be easily reused in other projects that are entirely different.

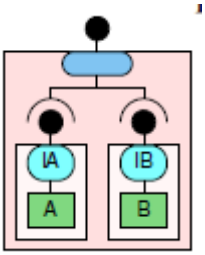
The XMAN component model has two component type; atomic component and composite component. An atomic component is a component that is singular in nature. It is not built up from other components. This is the smallest construction that can be found in a system that is built using the XMAN component model and it consists of a computation unit and an invocation connector. The computation unit abstracts all the computation and information associated with the component. It also offers methods held by the component and thus consist of a number of method objects. Computation Units do not invoke any operations outside themselves. The role of invoking methods outside a component is given to the interface which is the invocation connector. The

only way the method encapsulated in the computation unit can be access is via the invocation connectors.



After the atomic component we have the composite component which is built up of two or more components. The composite component can be built up of more either atomic or composite components. These are the components that are utilised in building the higher level structures XMAN component model.

In a composite component you can find many components each of which have their own methods defined inside them. A composite method definition allows specification the order in which subcomponent methods should be executed. This allows connectors to know the order in which methods which should be executed so that the necessary input parameters are given.



Composition connectors are the exogenous connectors that abstract method call control. There a three types of exogenous connectors that can be used on composite components: SEQUENCER, PIPE AND SELECTOR. The sequencer invokes subcomponent methods in sequence. Pipe connectors work in the exact

same way as the sequencer in the sense that it invokes the methods of a subcomponent but it also makes use of the subcomponent output as the input to the next subcomponent. As the name suggests, the selector connectors works by selecting which subcomponent method to execute. The determination of which subcomponent method is made evaluating boolean expressions.

4.7 Exogenous Connectors Component Model Tool

The Exogenous Connectors Component Model is built on the Generic Modelling Environment (GME). The GME allows the creation of application models. Because it can be configured clients can define metamodels and make use of these metamodels in creating implementations for particular domains.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 5: *Component-Based Mapping Requirements Approach*

The objective of this project, as stated before, is to define a systematic approach to map raw requirements expressed in natural language to component-based software architecture. As shown in figure 3 this approach has been defined through four phases. The first phase involves the analysis the natural language requirements into key words and then assigns these key words to parts of speech such as noun, verb or phrase. The key words resulting from this analysis are assigned to elements that relate to the XMAN component model. The second phase is the extraction of the semantic conceptual elements of the XMAN component model which are the component, computation and control, in order to prepare to build the partial architecture. The second phase is named the component analysis phase because it analyses the raw requirement to components and displays the computations that are produced by these components. The third phase involves the design of these components in the design phase of the XMAN component model; if the component is already in the repository this phase is skipped. The final phase involves retrieving the component from the repository and then composing it to the partial architecture

of the system. The final phase is processed in the deployment phase of the XMAN component model because it supports the incremental composition.

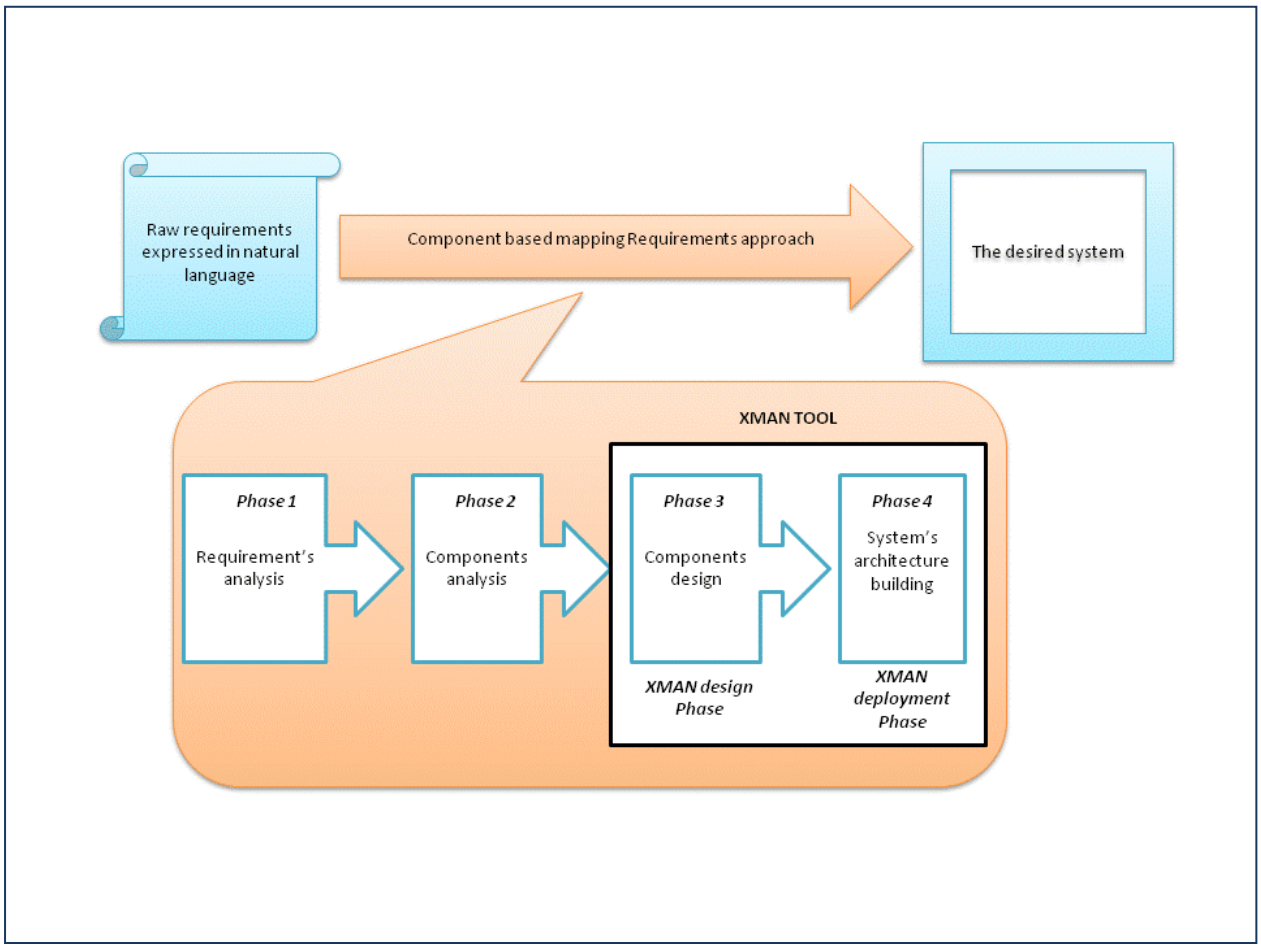


Figure 3: Component Based mapping requirement approach structure

5.1 PHASE 1: Requirement Analysis

The requirement Analysis can be considered as the initial phase in this approach. In this phase, the requirement is taken from natural language expression and analysed into its parts of speech, which is then assigned to the possible conceptual elements of X-MAN component model. Each part of speech

is mapped to X-MAN conceptual element according to specific rules [17]; these rules will be explained clearly in this chapter. The outcome of this phase is the requirement analysis table shown in Table. 1.

The requirements of the Home Heating System Controller [HHSC] will be used as an example to demonstrate this approach. This example has been taken from requirement analysis project done by a group of students [19]. (See appendix A)

The encapsulation feature of XMAN component model enables working on the requirements one by one. The main advantage of processing one raw requirement at a time is the flexibility which is missed in the usual practice of analysing all requirements together. Moreover, this procedure of processing the raw requirements enables dealing with any number of requirements [17].

In this phase, each requirement passes through three steps presented below:

1) The POS Tagger is applied on the raw requirement to parse it to key words which assigned to parts of speech such as verbs, nouns and phrases [18].

2) The output of the POS Tagger is examined based on three tables, verbs table, noun table and phrases table. These tables are used as a reference of the mapping process (see table1, 2 and 3). If the part of speech that assigned to the key word is verb then we return this key word to the verb table to find out its

category and the aspect of XMAN component model that could denotes, the same is done with the noun and the phrases.

If the part of speech that assigned to the key word, that extracted from the requirement, is verb then it could be one of three types; Computation, State and Event verb [17]. Computation verbs are the verbs that represent a data transformation, which processes some actions on the inputs data in order to produce the desired output data and achieve specific behaviour [17].This type of verbs is extracted from Action [5],[4]. Count, Calculate and analyse are some examples of computation verbs. The second type is state verb, which is extracted from State [5], [4].State verbs denotes computations that realise states [17], this type of computation results changes in the components attributes. Keep, remain and maintain are some examples of state verbs[17]. Finally, event verbs which denote event that triggers the computation[17]. Event verbs are extracted from Emergence [4]. Signal, reach and press are some examples of event verbs. Table 1 summarises the information that could be extracted from verbs [17].

Table 1: Verb Table

Category of verbs	*Denotes*	*Example*
Computation	Computation (data transformation)	withdraw, deposit, cooking
State	Internal state of component (attribute value of component)	Keep, remain
Event	Events (that can trigger computation)	press,cancel,push

If the key word is noun, it is transferred to the noun table. There are four types of nouns; conceptual component, data, state and computation noun [17]. The abstraction of the candidate component can be specified from the conceptual component noun such as word counter, auto-teller machine, etc [17]. Conceptual component can be extracted from Class [17], [5]. The second type of noun is data noun, which denotes values that may have to be stored or retrieved; data noun is extracted from Value [17], [5]. State noun is another type of noun which denotes attributes, identification and states [17]. Closed and opened are examples of state nouns. State noun is extracted from attribute [5]. Finally, computation noun such as authentication refers to data transformation processes. In other word, the computation of the component can be extracted from the computation noun. Table 2 shows the information that can be extracted from noun [17].

Table 2: Noun Table

Category of noun	*Denotes*	*Example*
Conceptual component	Conceptual hooks for components	Power tube Word counter
Data	Value or set of values	1,c,integer
State	Attribute name and state	closed, open
Computation	Computation (data transformation)	registration, transmission, movement

Furthermore, there are four aspects can be driven from phrases. First, the component and/or computation which are extracted from descriptive expression phrases such as a graduate student and the pin is incorrect [17]. The descriptive expression is implemented from descriptive expression [4], [17]. Secondly, predicate methods of the computation unit which extracted from predicate phrases such as is valid and is normal [4], [17]. Third, the control structure phrase which is adapted from English control structure [4] and denotes flow of control [17]. Once, until, if…then…else and while are some examples of control structure phrases. Table 3 summarize the elements that could be extracted from phrases [17].

Table 3: Phrase Table

Category of phrase	*Denotes*	*Example*
Descriptive expression	Denotes components or computation	The earlier date May denote date or compare date function
Predicate	Computation – true or false methods	Is enabled, is valid
Control structure	Control flow	If, then, else While

3) The analysis process, which is performed on the requirement, is summarized on a table called the Requirements Analysis Table. This step recapitulates the output of the requirement analysis in one table that make it easy to filter this

information and transfer it to the next phase to determine the candidate components which extracted from this requirement and its computation.

The requirement Analysis Table contains three columns. The first column contains the key word or the part of the requirement which could be verb, noun or phrase. The second column contains this part type. The last column displays what it could denote. Figure 4 presents the steps of the requirement analysis phase.

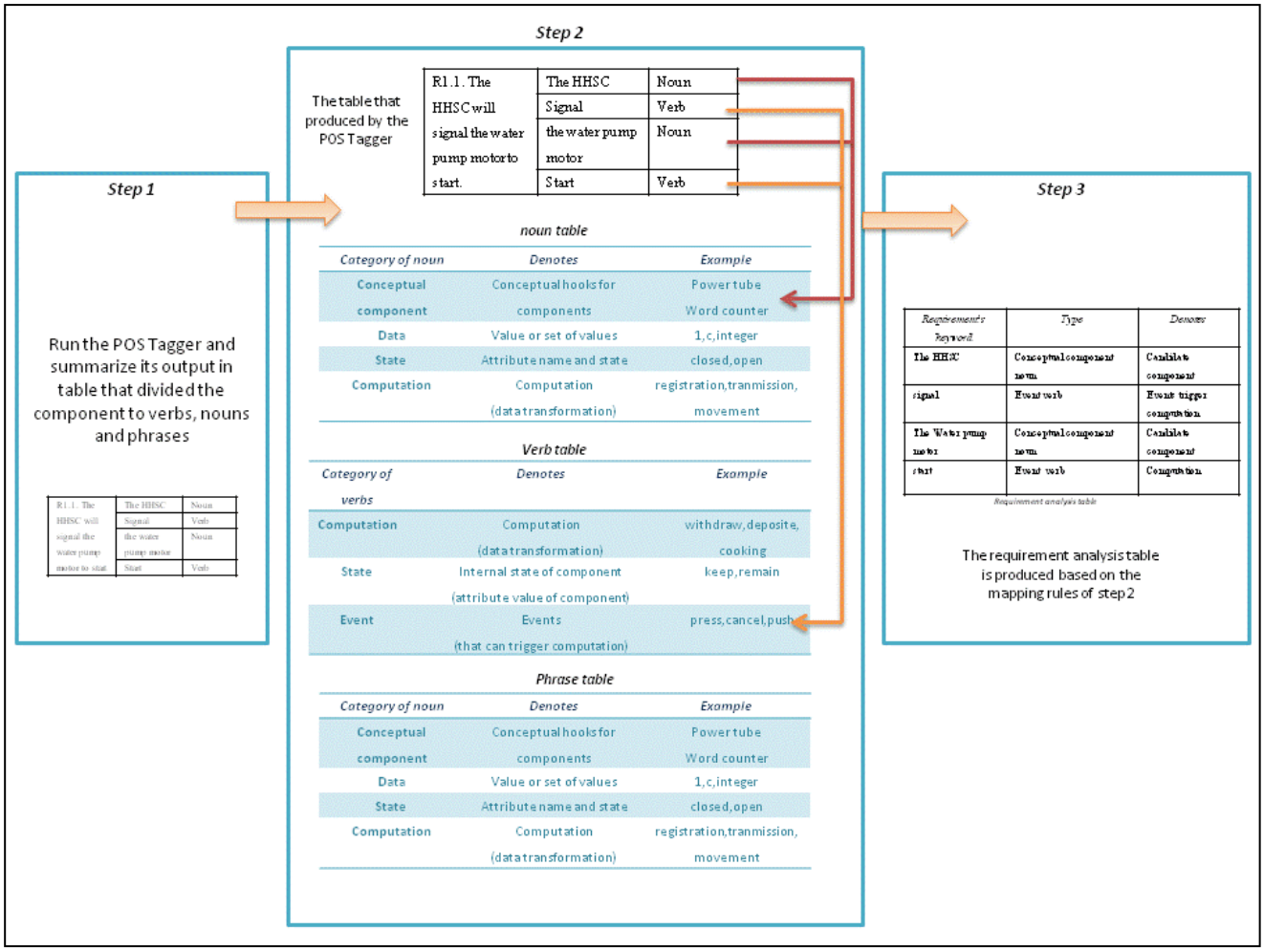


Figure 4: The steps of Requirement Analysis Phase

5.2 PHASE 2: Components Analysis

In this phase four main factors are determined; candidate components and their type, computation and data. This information is summarised based on the previous phase.

5.2.1 Candidate Components

The candidate components are extracted directly from the conceptual component noun or the descriptive expression phrases. For example, (Water Pump Motor) in the requirement analysis table of R1.1 Table 9, which will be presented later in this chapter is stated as a conceptual component therefore it provides a candidate component possibility [17]. In some cases the conceptual component noun refers to the control of the system. For instance, in R1.1 the HHSC is abbreviation of home heating system controller which is settled as a conceptual component noun in the requirement analysis table but it is actually refers to the control of the system.

In addition, the descriptive expression can be an inspiration for the candidate components [17]. For instance, the primary water circulation valve is a descriptive expression that denotes candidate component.

5.2.2 Candidates Components' type

The type of candidate component is determined in this phase. As we presented in chapter four the components in the XMAN component model can be divided in to tow types; atomic component and composite component. The type of component can be extracted from the context of requirements. For example, if we looked at the requirements of the home heating system controller figure 5, we will find that requirement 3.1 gives an abstraction of the furnace's activation process. However, the explanation of this process is expanded in requirements (3.1.1 – 3.1.4). That means the requirement 3.1 represents a composite component while the requirements (3.1.1 – 3.1.4) represent the atomic components that construct the composite component.

3.1 The HHSC will activate the furnace.
3.1.1 The HHSC will signal the water pump motor to start.
3.1.2 Once motor speed reaches 1000 rpm, the HHSC will signal the ignition device to be activated and oil valve to be opened.
3.1.3 Once the system water temperature reaches a value predefined by the user, the HHSC will signal the primary water circulation valve to be opened.
3.1.4 The HHSC will retain the length of time from the last deactivation and not reactivate the furnace until a period of 5 minutes has elapsed.

Figure 5: The Requirements of Furnace Activation Process

5.2.3 Candidate's Computation of a component

As we mentioned in chapter four, the computation and control can be considered as the key semantic concepts of XMAN component model. Each component has its own computation unit. The Computation represents the transformation that is made on data or by other word function evaluation which could result in some updates in variables. On the other hand, control represents the flow of execution of computations' pieces. "The result of a piece of control invoking a computation is a piece of behaviour". [17]

In this phase the computation of the component is determined and the code is prepared. The computation can be extracted directly from action noun such as *authentication* or data transformation verbs such as *withdraw* [17], [8]. State verbs such as *keep* and *remain*, and events verbs such as *select* can also denote the computation directly [17], [8]. However, the computation can be denoted implicitly in the descriptive expression such as *to be activated* [17], [8]. The computation of functions that return true or false is extracted from predicate phrases such as *has elapsed*, *is normal* and *is green*[17], [8]. Table 4 summarize the key words that denote computation [8].

Table 4: Computation Extraction

Part of speech	Category	Denotes	Example
Verbs	Data Transformation	Computation (data transformation)	Change Price
	State	Internal state of components (attribute values of component)	Keep, remain
	Event	Events that can trigger computation	Press, enter, select
Noun	Action	Computation (Data transformation)	Identifier, validation
Phrases	Descriptive expression	Denotes computation implicitly	Is displayed
	Predicate	Computation (functions that return true or false)	Is normal , is green

The information that is resulted from this phase is summarized in the component analysis table. This table contains the candidate components that are extracted from the requirement and its type wither atomic or composite component and the last field is the computation that performed by this component. After this phase the component is ready to be designed and composed to the system architecture in the coming phases of this approach. Table 10, show an example of component analysis table which performed on requirement R.1.1 of the home heating system.

5.3 PHASE 3: Components design.

In phase 1 and 2 the information that is needed to design the components that are extracted from the requirement is prepared. In this phase the components are designed if they are not already in the repository.

5.3.1 Designing component steps

There are two main types of component as we mentioned previously; an atomic component and Composite Component. The atomic component consists of computation unit, invocation connector and interface. The component encapsulates the computation by having a set of methods in the computation unit that do not need to invoke methods in the computation units of other components. On the other hand, the role of the invocation connector is passing the control and the input parameters, which come from other components to the computation unit of the component in order to invoke a specific method, and return the control with the results to the place that it came from, that is the meaning of encapsulating the control.

Composite components are made from joining together any number of atomic components through the use of a composition connector. The composition fully encapsulates the components control structure such that branching and looping and other control structures that allow the connection of the sub-components are

completely hidden. Composite components encapsulate computation in the same way as atomic components.

In this project the XMAN tool will be used to show how to use the information that is extracted from the raw requirement and prepared in phase 1 and 2, to build the component based architecture. Each requirement is mapped into one or more components which are displayed clearly in the components analysis table from phase 2. The XMAN tool allows designing the components in the design phase and linking these components with current architecture in an incremental fashion in the deployment phase. This section will explain the steps of designing the component and deposit it into the repository.

1) Search for the components in the repository

The first step of phase 3 is searching for the components in the repository. If this component is already in the repository the component design phase will be skipped, otherwise the component will be design as in step 2.

For the first requirement the initial system architecture is created in the deployment phase. This architecture is empty by default. Figure 6 shows the initial architecture of the Home Heating System Controller.

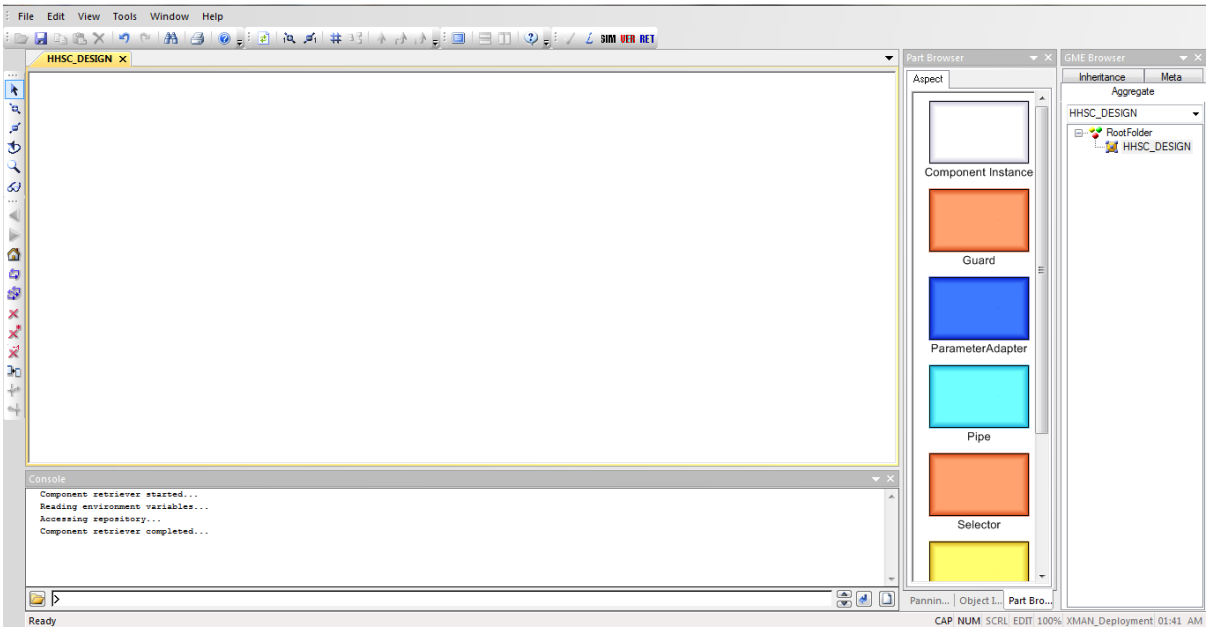


Figure 6: The initial architecture of HHSC system

To find a component we should press the retrieve button which opens the component retrieval Dialog. After that, we type the component name which stated in the component analysis table, in the search string text box in the component retrieval Dialog. If the component already exists in the repository it will be shown in the component retrieval Dialog as shown in figure 7.

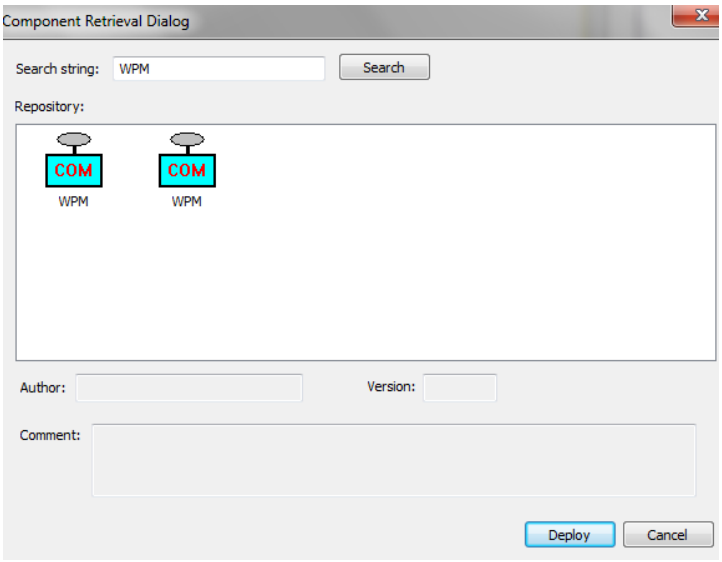


Figure 7: exist components in the repository

On the Other hand, if we search for non-existent component, the component retrieval dialog will show the repository box empty as in figure 8.

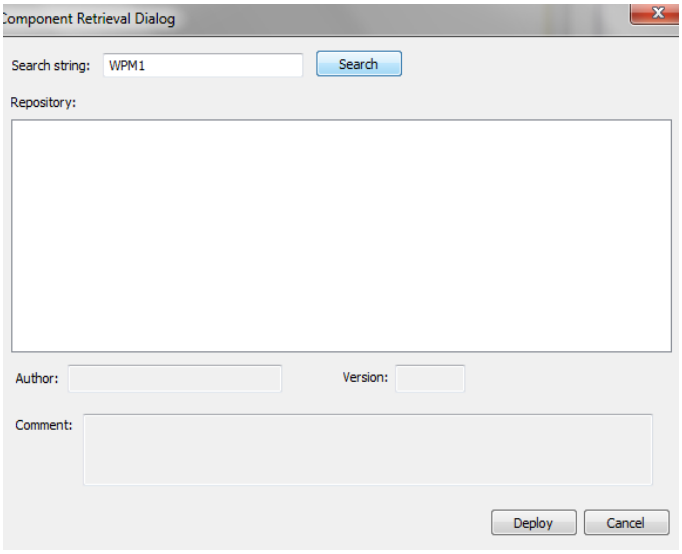


Figure 8: search for non-existent component

2) Design the components

If the component does not exist in the repository we should design it and store it in the repository. In this phase the component is designed in the design phase of the XMAN component model.

2.1) Select the XMAN_Design paradigm from the listed paradigm and create a project file to the component design.

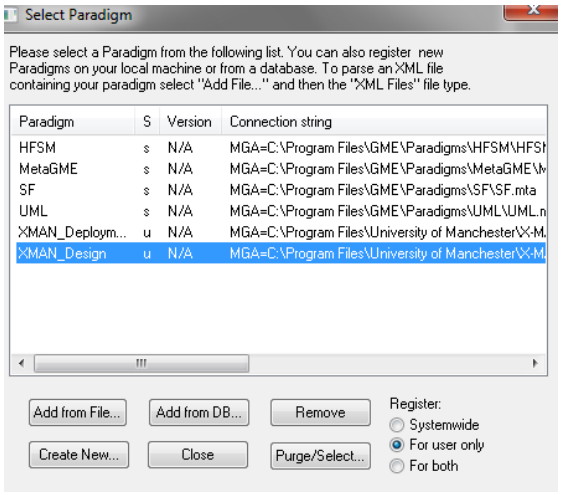
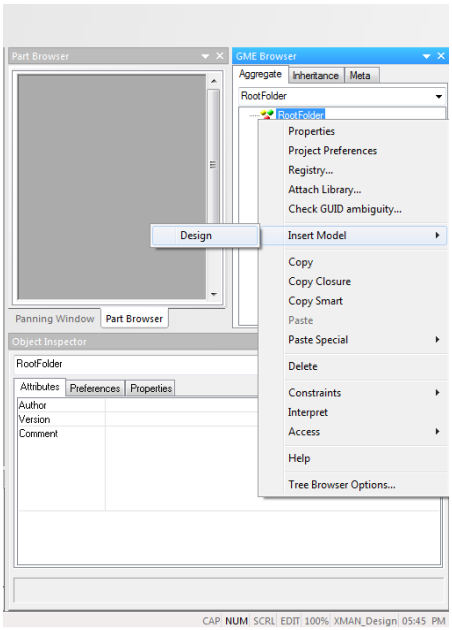


Figure 9: Paradigm selected in component design phase.

2.2) create the file that contains the design.

An empty project associated with “XMAN_Design” paradigm has been created by the GME. The project is not actually empty it contains a root folder which named "RootFolder".

2.3) Insert the design model by clicking the right click on the root folder then insert a “NewDesign”. To distinguish the design model of a component it is better to rename it by the component's name.



2.4) after creating the design model the part browser will displays the two types of component to select from; The Atomic Component, and Composite Component. The component’s type has been determined from phase 2 in the component’s analysis table.

The part browser shows the aspects of the selected component type. If the atomic component is selected, the aspects that are presented in the part browser are the invocation connector which should be connected to the computation unit and the interface.

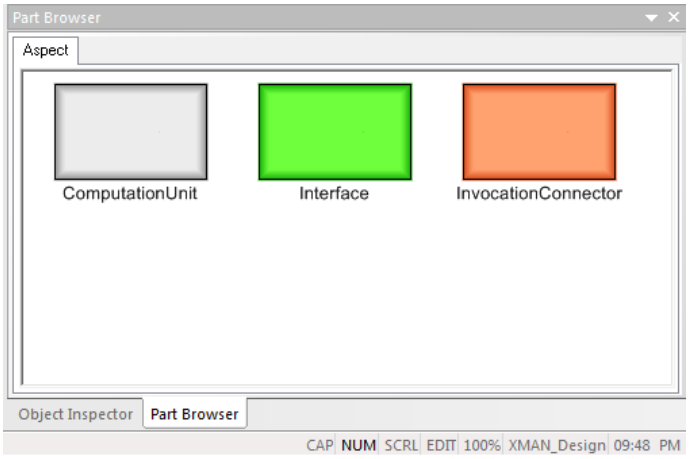


Figure 10: The aspects that construct the atomic component

2.4.1) the computation unit is inserted and named. After that the behaviour of the component is implemented in the property executable code part. This code has been prepared in the component analysis table from phase 2.

2.4.2) the provided behaviour needs to be modelled as shown in appendix A.

2.4.3) the invocation connector is inserted and linked to the computation unit via a non sticky connect mode.

2.4.4) the component's interface is generated by the IGN- component interface generator - button in the tool bar.

2.5) the component is deposited in the repository to be saved until the system architecture is built.

5.4 PHASE 4: System architecture building:

The system architecture in the component based software development is built by composing components. The component analysis table is used to organize the components that build the partial architecture of the requirement. That means each requirement has its own partial architecture. The partial architecture is composed with the system architecture incrementally. In this section the incremental composition is discussed briefly.

5.4.1 Incremental Composition

XMAN Component model is unique and effective because composition corresponds to actual system architecture unlike the other mapping approach which maps the requirement to intermediate model such as use cases. Incremental composition is defined as (i) allowing the addition components and compositions to an existing architecture; and (ii) preserving the properties of the existing architecture whilst incrementing on that architecture. Because it preserves the existing architectures and simply adds and compliments it, the XMAN component model is an effective technique for mapping requirements. Requirements can be confronted one by one without the concern that a

requirement in the later parts could disrupt the entire architecture. The requirements are mapped into the partial architecture by adding components and compositions. [17]

The mapping process beings without an architecture, as components are added a partial architecture begins to develop and as we add additional components and compositions to that architecture this is incremental. The partial architecture satisfies at least one of the requirements. [17]

The partial architecture is incremented as we add new components that satisfy a new requirement. The components we add to the partial architecture will satisfy new requirements but will not affect the partial architecture because of the behaviour preservation that is characteristic of the XMAN component model. The system architecture is completed when all the requirements have been satisfied by the incremental adding of components to the partial architecture. [17]

The steps of building the partial architecture of the components that extracted from raw requirement is listed below.

4.1) Select the XMAN_Deployment paradigm from the listed paradigm and create a project file to the component design.

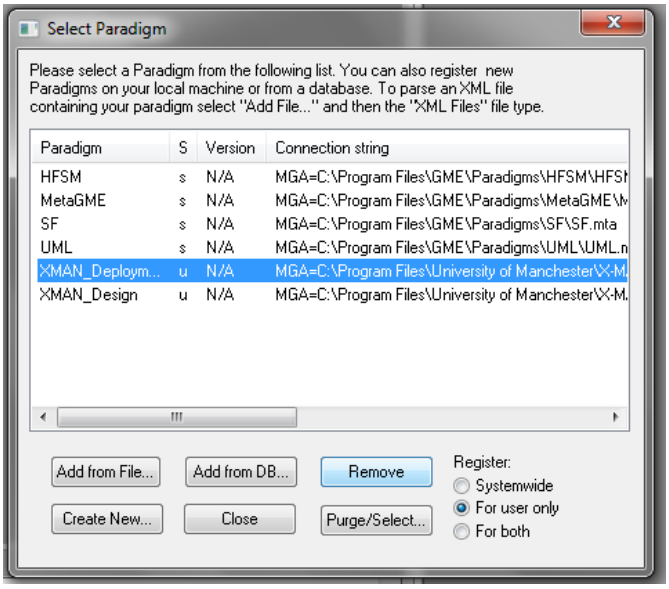
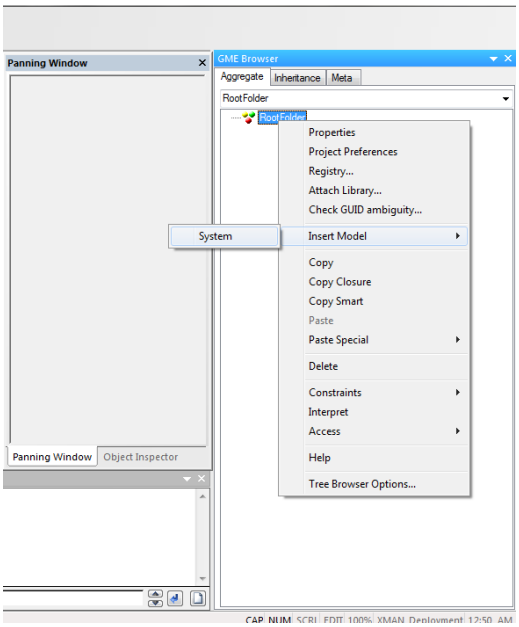


Figure 11: The Paradigm selected in the architecture building phase

4.2) create the file that contains the system.

An empty project associated with “XMAN_Deployment” paradigm has been created by the GME. The project is not actually empty it contains a root folder which named "RootFolder". Insert the system model by clicking the right click on the root folder then insert a “New system”.



4.3) the components that correspond to build the partial architecture,which determined in the component analysis table, are retrieved and composed by one of the composite connectors which are listed in the part browser(Pipe-Selector-Sequencer). These connectors also used to compose the new partial architecture with the existing architecture.

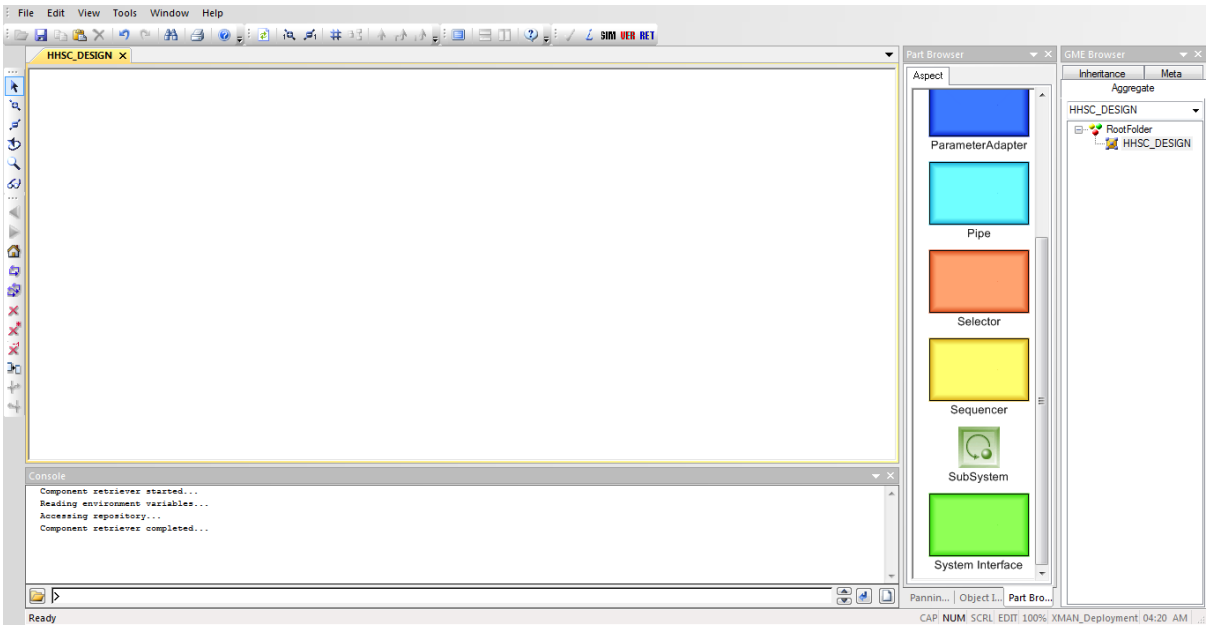


Figure 12: Composition Connectors

These steps are repeated for each partial architecture that corresponds to one requirement until all requirements are satisfied.

5.5 complete example

We will use a part of the heating home system controller [hhsc] functional requirements as an example to show procedure of this approach. The functional requirements of this part are listed below:

R.1 The HHSC will activate the furnace.

R.1.1 TheHHSC will signal the water pump motor to start.

R.1.2 Once motor speed reaches 1000 rpm, the HHSC will signal the ignition device to be activated and oil valve to be opened.

R.1.3 Once the system water temperature reaches a value predefined by the user, the HHSC will signal the primary water circulation valve to be opened.

R.1.4 TheHHSC will retain the length of time from the last deactivation and not reactivate the furnace until a period of 5 minutes has elapsed.

5.5.1 REQUIREMENT R.1

PHASE 1: Requirement Analysis

The raw requirement is sent firstly to the requirement analysis and the steps of this phase are applied as followed in order to produce the requirement analysis table.

1- The requirement is send to the POS Tagger to divide it into; verbs, nouns and phrases.

Table 5: POS Tagger result of R.1

R.1 The HHSC will activate the furnace.	The HHSC	Noun
	Activate	Verb
	the furnace	Noun

2- The output of step 1 is analysed based on the 3 tables; the verb table, noun table and phrases table. Firstly, The HHSC is conceptual component noun. Secondly, activate is data transformation verb and it denotes computation. Finally, the Furnace is conceptual component noun, it denotes candidate component.

3- The information in step 2 is summarised in the requirement analyser table of R.1.

Table 6: the requirement analysis table of R.1

Keyword	Type	Denotes
The HHSC	conceptual component noun	Candidate component
Activate	Computation verb	Computation
The Furnace	conceptual component noun	Candidate component

PHAE 2: the Component Analysis

Filtering the information produced in phase one shows that:

The Candidate component that extracted from R.1 is Furnace Activator. This component is composite component therefore its computation cannot be extracted directly from one raw requirement.

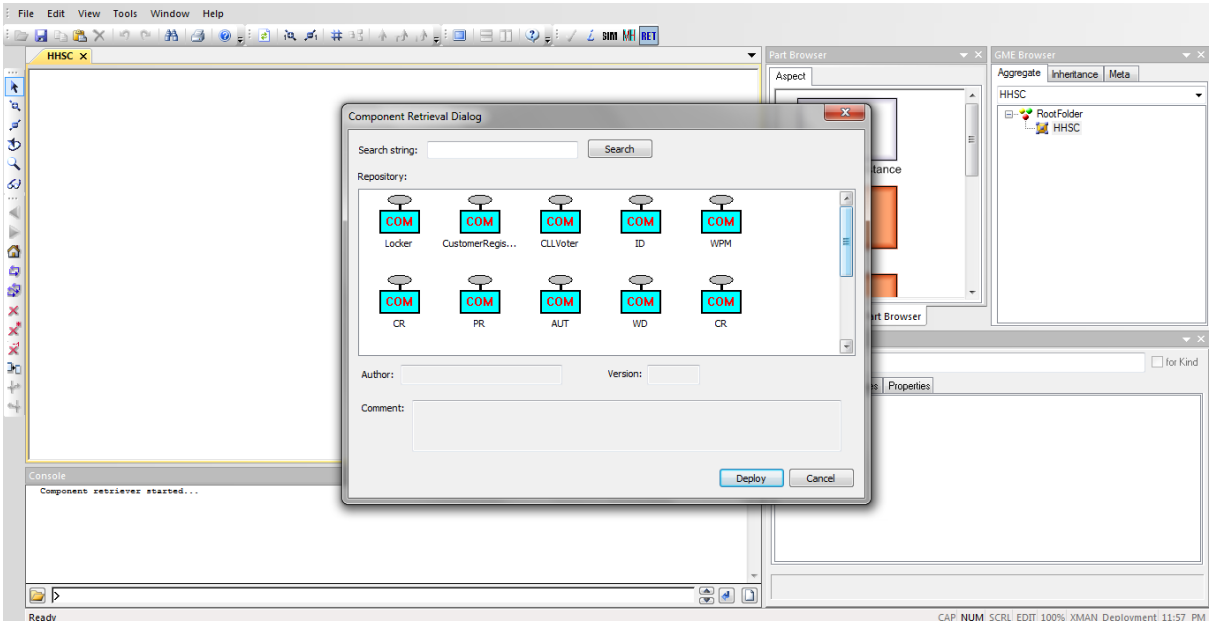
Table 7: Component Analysis Table of R.1

Candidate Components	Type	Computation
Furnace Activator	Composite Component	Activate The Furnace through composing the atomic components expressed in the coming requirements

PHASE 3: the Components design.

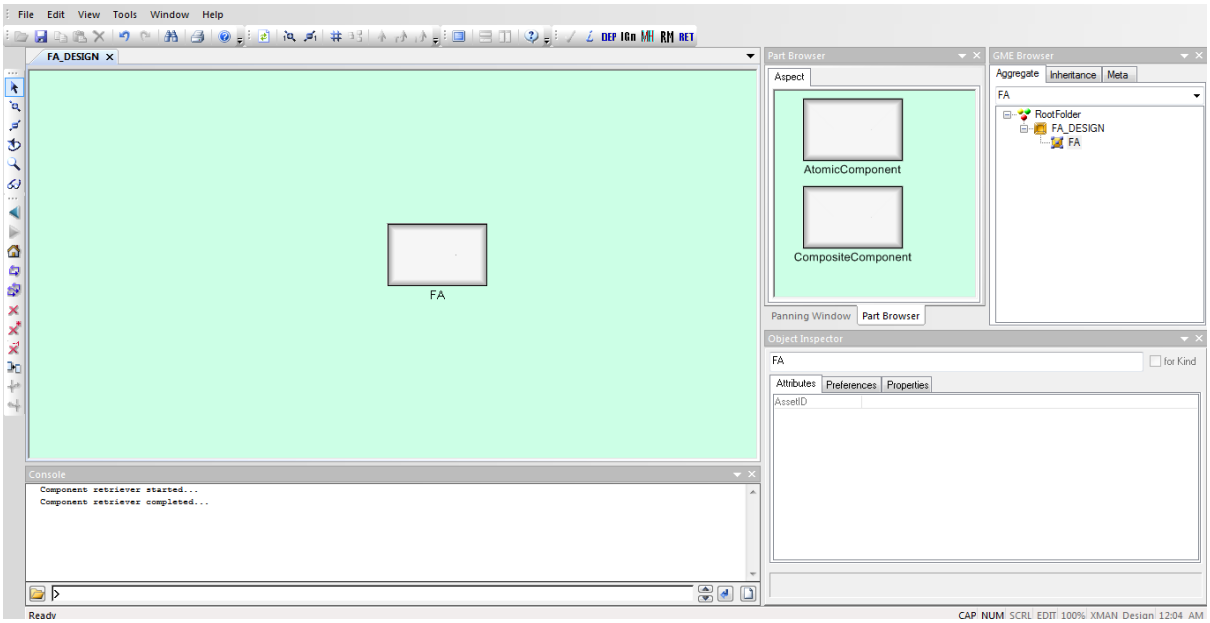
To design the components listed in the components analysis table the XMAN tool is run.

1- Search for the Furnace Activator (FA) component in the repository.



The (FA) component does not exist. So it needs to be designed.

2- The (FA) component type is composite component (see component analysis table of R.1 (Table 7)). Therefore, a composite component selected from the part browser and renamed by the component name which is (FA). As shown in fig X.



The third and fourth phases of this requirement will be skipped because the XMAN tool is under developing tool and it still does not support designing the composite component. The requirements (R1.1-R1.4) will be processed as individual requirements, not related to R.1, to demonstrate the incremental composition in the deployment phase.

5.5.2 REQUIREMENT R1.1
PHASE 1: the Requirement Analysis

1- POS Tagger Result.

Table 8: POS Tagger result of R1.1

R1.1. The HHSC will signal the water pump motor to start.	The HHSC	Noun
	Signal	Verb
	the water pump motor	Noun
	Start	Verb

2- **Sending the POS Tagger result to the verbs, nouns and phrases tables.**

We can see that the HHSC noun refer to the system control. Moreover, by returning to the verbs table (Table 1) we can find that signal and start are event verbs that trigger computation. On the other hand, the water pump motor is a conceptual component noun which could refer to the candidate component extracted from this requirement. These information summarized bellow in the Requirement Analysis Table.

Table 9: Requirement Analysis Table of R1.1

Key Words	Type	Denotes
The HHSC	Conceptual component noun	Candidate component
Signal	Event verb	Events trigger computation
The Water pump motor	Conceptual component noun	Candidate component
Start	Event verb	Computation

PHASE 2: the Components analysis.

1- **The candidate Components**

From Table 9: the requirement analysis table of R.1.1 we can see that we have two conceptual component nouns. The first is the HHSC which is the abbreviation of Home heating system controller. As it obvious this noun refers to the system control so we can not use it as a candidate component. However, the water pump motor is the second conceptual

component noun in R1.1 which can be used as the candidate component extracted from this requirement.

2- Components' types

The WPM component is an atomic component because we can build it by itself. By other word, WPM component does not need other components to be constructed.

3- Computation

The requirement analysis table of R.1.1 shows that we have two verbs *signal* and *start*. Both of these verbs are event verbs which denote events that trigger computation. The computation that is triggered by these verbs is pumping. Pumping computation means increasing the motor speed in order to pump water.

The code of the computation is prepared and written with C programming language which is the language that supported by the XMAN Tool.

Table 10: Component Analysis Table of R1.1

Candidate Components	Type	Computation

Water Pump Motor (WPM)	Atomic Component	void waterPumping(intCMD,int& MS)
		{
		if(CMD == 1)
		MS++;
		}

PHASE 3: the Components design phase.

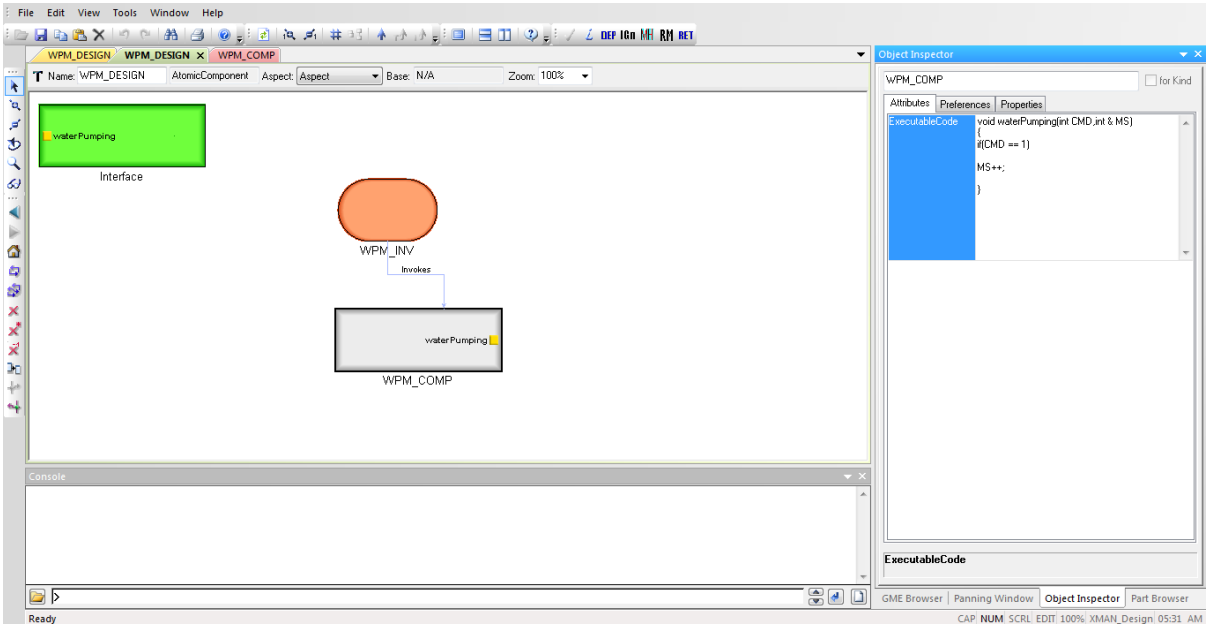


Figure 13: WPM component design

Before depositing the component into the repository we have to check in all requirements to see whether all information related to this component has been designed or not because depositing the component into the repository prevent the ability of modification on the component design. We can find some computation divided through more than one requirement on the other hand, some requirements contain more than one computation.

For WPM component the first part of the next requirement (R.1.2) is related to this component because it defines the condition that stop pumping computation.

5.5.2 Modifications made on the component extracted from R1.1

PHASE 1: the Requirement Analysis.

1- POS Tagger Result.

Firstly, the first part of R1.2 will send to the POS tagger.

Table 11: POS Tagger result of R.1.2 (P1)

R.1.2 (first part)once motor speed reaches 1000 rpm	Once	Phrase
	motor speed	Noun
	Reaches	Verb
	1000 rpm	Noun

1- Sending the POS Tagger result to the verbs, nouns and phrases tables.

2- (Once) is a phrase denotesthe flow of control. In addition, (motor speed) is a conceptual component that denotes attribute name. On the other hand, (reaches) is an event verb that triggers computation. Moreover, (1000 rpm) is data noun which denotes the value that needs to be used in the condition.

Table 12: Requirement Analysis table of R1.2 (P1)

Key words	Type	Denotes
Once	Control structure Phrase	Flow of control
motor speed	Conceptual component Noun	Attribute name
Reaches	Event Verb	Events trigger computation
1000 rpm	Data Noun	Value

PHASE 2: the Components' analysis.

1- **The candidate Components**

As we mentioned before, this part of requirement R1.2 is related to the water pump motor component which extracted from R1.1.

2- **Components' types**

The WPM component is an atomic component because we can build it by itself. By other word, WPM component does not need other components to be constructed.

3- **Computation**

The modification is been done on the computation part of the component by adding for loop that stops this computation when the motor speed (ms) reach 1000.

Table 13: Component Analysis Table of R.1.1 + R.1.2 (P1)

Candidate Components	Type	Computation

Water Pump Motor (WPM)	Atomic Component	void waterPumping(intCMD,int& MS)
		{
		if(CMD == 1)
		for(inti=0; MS<=1000 ;i++)
		{
		MS++;
		}
		}

PHASE 3: the Components design.

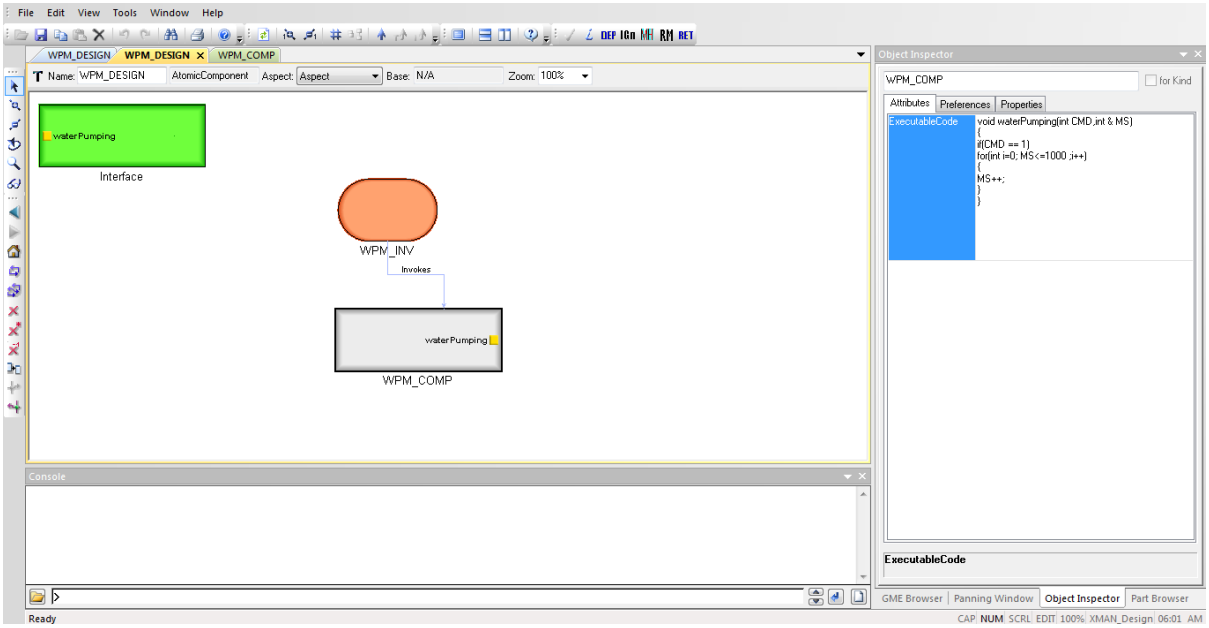


Figure 14: WPM component modified design

The WPM component is ready now to be deposited to the repository.

PHASE 4: The architecture building

Firstly, a project file that will contain the system architecture is created and the system model is inserted. The WPM component is retrieved and instantiated as the first partial architecture because it is the only component corresponds to R.1.1 and R.1.2.

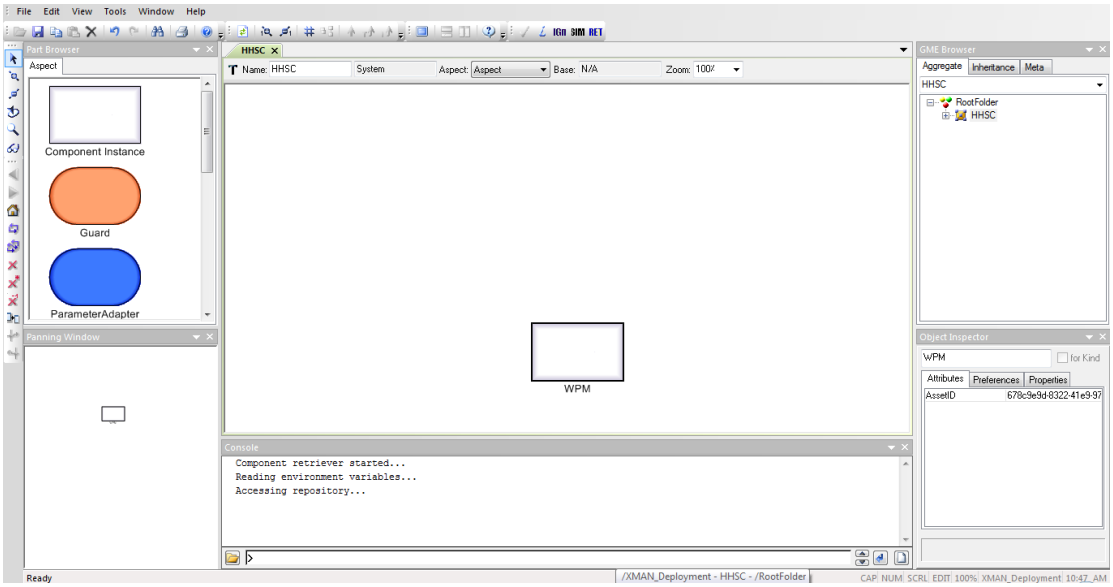


Figure 15: The first partial architecture

5.5.3 REQUIREMENT R.1.2 (P2):

PHASE1: the Requirement Analysis

2- POS Tagger Result.

The second part of R1.2 will send to the POS tagger.

Table 14: POS Tagger result of R1.2 (P2)

R.1.2 (P2) the HHSC will signal the ignition device to be activated and oil valve to be opened.	the HHSC	Noun
	Signal	verb
	the ignition device	Noun
	Activated	Noun
	oil valve	Noun
	Opened	Noun

3- Sending the POS Tagger result to the verbs, nouns and phrases tables.

Table 15: Requirement Analysis Table of R.1.2 (P2)

Key words	Type	Denotes
the HHSC	Conceptual component noun	Candidate component
Signal	Event verb	Events trigger computation
the ignition device	Conceptual component noun	Candidate component
Activated	State noun	Internal state of component (Attributes values of component)
oil valve	Conceptual component noun	Candidate component
Opened	State noun	Internal state of component (Attributes values of component)

PHASE 2: the Components analysis

4- The candidate Components

The candidate components extracted from R1.2 (P2) are the ignition device (ID) and oil valve.

5- Components' types

The ignition device (ID) and oil valve (OV) are atomic components.

6- Computation

As we can see from the requirement analysis table of R.1.2(P2)the key word that corresponding to the computation is signal which is a event verb that trigger a computation.

The computation that triggered by this verb is changing the state of the components.This computation is inspired from the state noun key words which are activated and opened.

Table 16: Component Analysis Table of R.1.2 (P2)

Candidate Components	Type	Computation

the ignition device(ID)	Atomic component	intsetIDState(intcmd, int&IDstate)
		{
		if(cmd==1)
		IDstate=1;
		else
		if(cmd==0)
		IDstate=0;
		}

Oil Valve (OV)	Atomic Component	void setOVstate(int CMD , int& OVSTATE)
		{
		if(CMD == 1)
		OVstate=1; //opened
		else
		if(CMD == 0)
		OVSTATE = 0; //closed
		}

PHASE 3: the Components design

(The ignition device (ID))

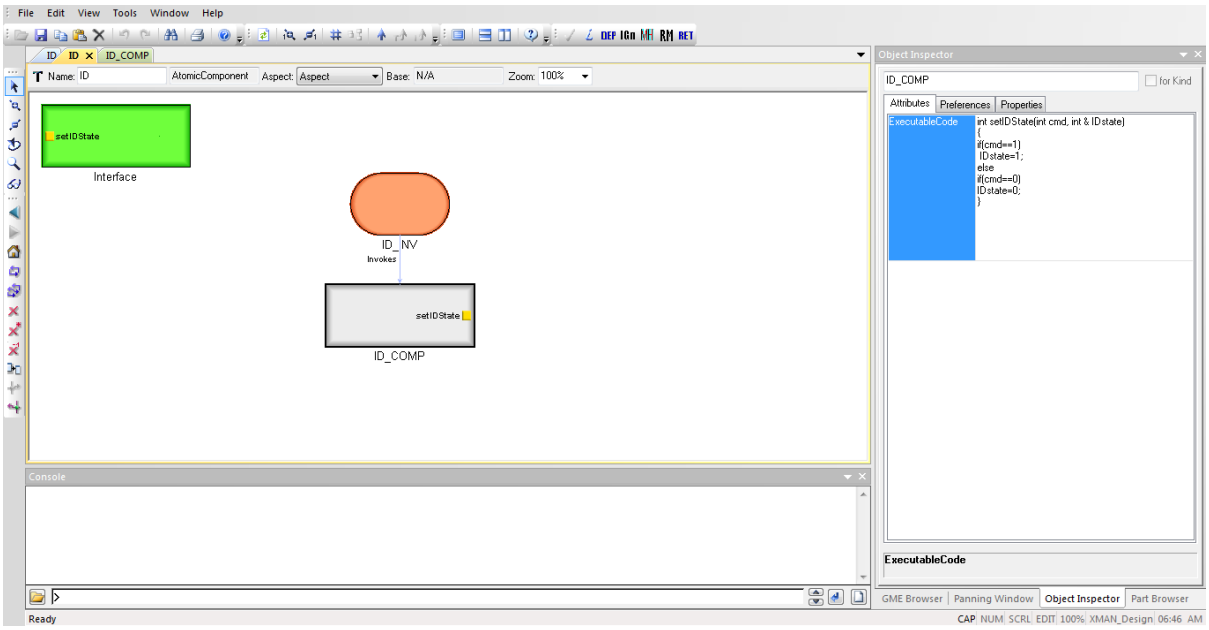


Figure 16: ID component design

(The Oil Valve (OV))

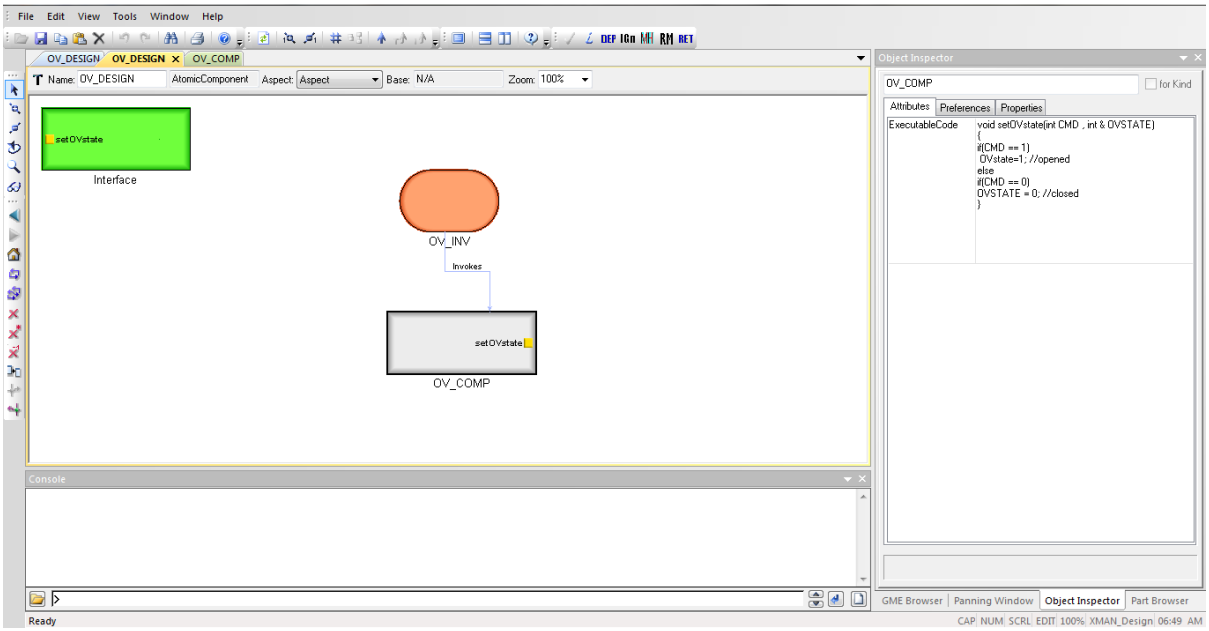


Figure 17: OV component Design

The ID and OV components are deposited into the repository to be ready to build the second partial architecture which derived from R1.2 (P2).

PHASE 4: The architecture building

The Components that are listed in the component analysis table of R.1.2 (P2) are retrived and composied with composite connectors in order to build the partial architecture of this requirenment.

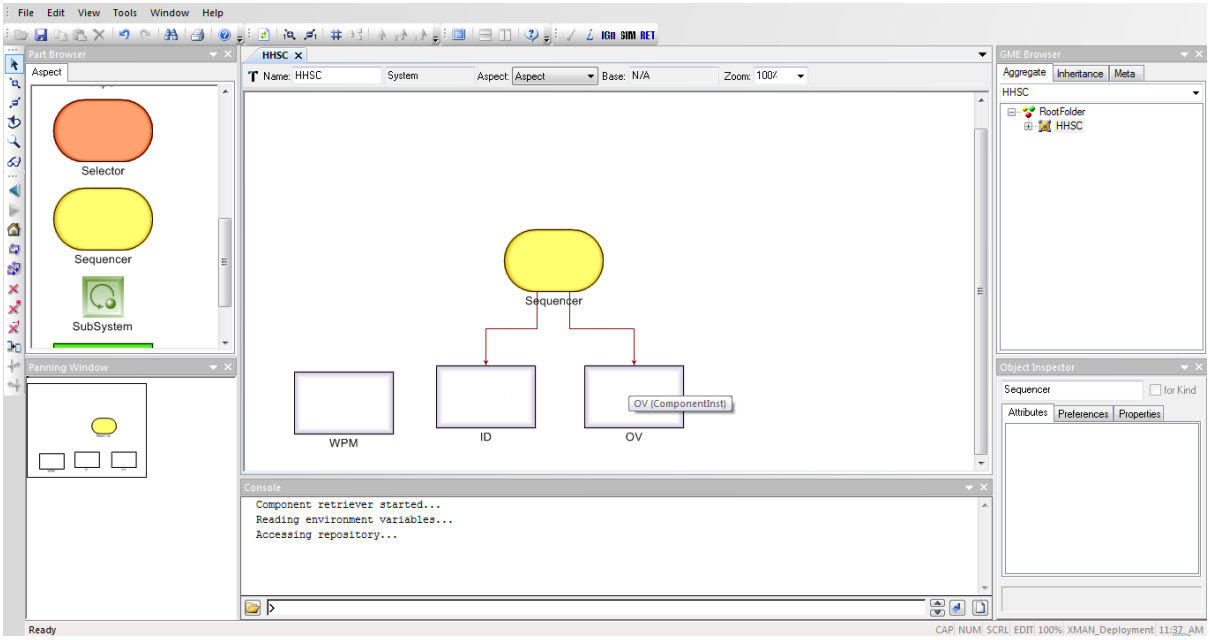


Figure 18: The second partial architecture

The new partial architecture is composed with the previous one via the composite connectors.

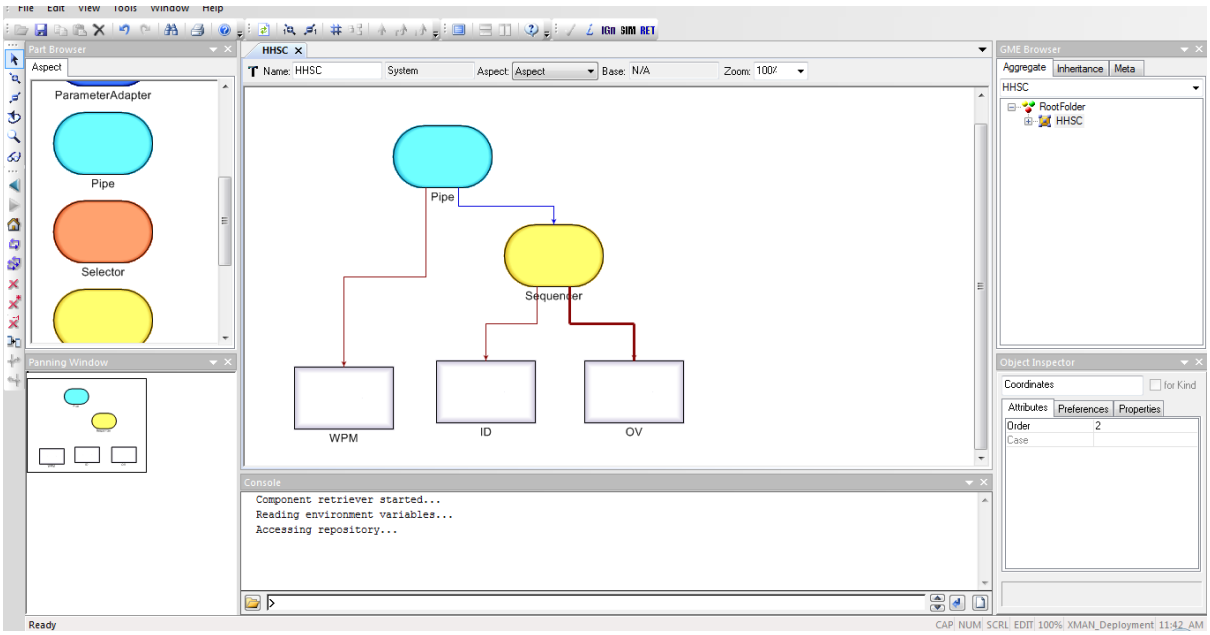


Figure 19: linking the second partial architecture with the current architecture of the system

5.5.4 REQUIREMENT R.1.3

PHASE 1: the Requirement Analysis

3- POS Tagger Result.

Table 17: The POS Tagger result of R1.3

R1.3: Once the system water temperature reaches a value predefined by the user, the HHSC will signal the primary water circulation valve to be opened.	Once	Phrase
	The system water temperature	Noun
	Reaches	Verb
	a value predefined by the user	Phrases
	the HHSC	Noun
	signal	Verb
	the primary water circulation valve	Noun
	to be opened.	Verb

4- **Sending the POS Tagger result to the verbs, nouns and phrases tables.**

Table 18: The requirement Analysis Table of R.1.3

Key Words	Type	Denotes
Once	Control Structure phrase	Control
The system water temperature	Conceptual component	Candidate component
Reaches	Event verb	Event that trigger computation
a value predefined by the user	Data noun	Value
the HHSC	Descriptive expression	Computation
Signal	Event verb	Event
the primary water circulation valve	Conceptual component	Candidate component
to be opened.	State verb	Attribute

PHASE 2: the Components analysis

The candidate Components

The candidate components extracted from R1.3 are the system water temperature (SWT) and the primary water circulation valve (PWCV).

Components' types

The system water temperature (SWT) and the primary water circulation valve (PWCV).are atomic components.

Table 19: Component Analysis Table of R.1.3

Candidate Components	Type	Computation

The system water temperature (SWT)	Atomic component	void measure(int TEMP, int WTEMP, int SIGNAL)
		{
		if (WTEMP == TEMP)
		SIGNAL = 1;
		else
		SIGNAL = 0;
		}

the primary water circulation valve (PWCV).	Atomic Component	voidsetOVstate(int CMD , int& OVSTATE)
		{
		if(CMD == 1)
		OVstate=1; //opened
		else
		if(CMD == 0)
		OVSTATE = 0; //closed
		}

PHASE 3: the Components design

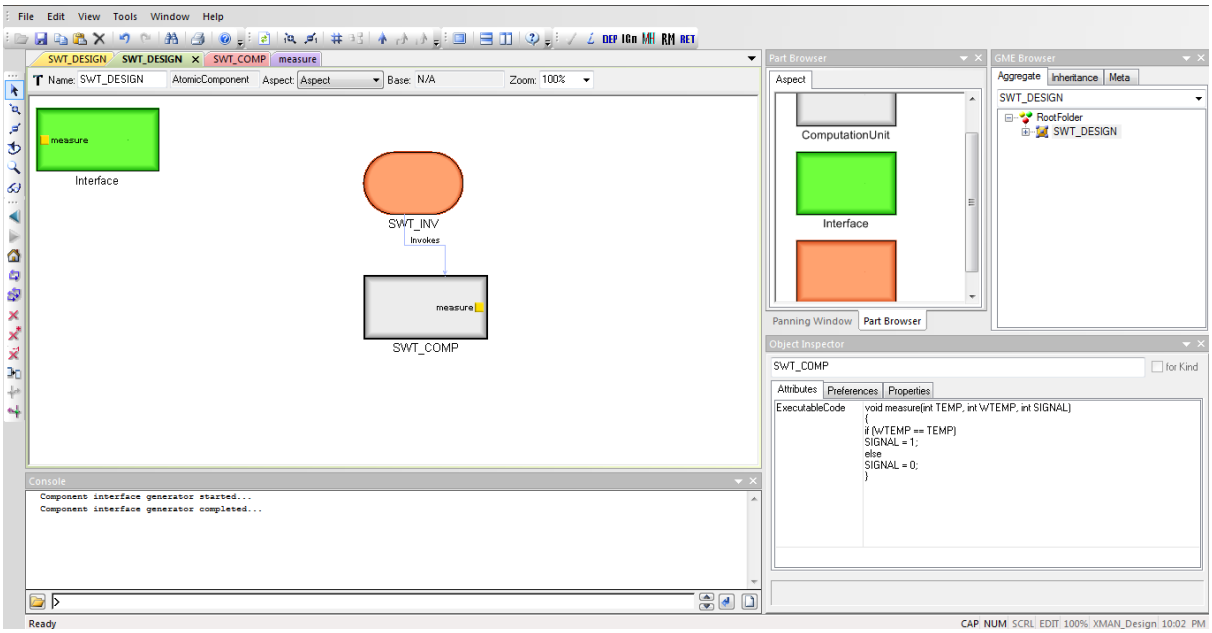


Figure 20: SWT component Design

PHASE 4: The architecture building

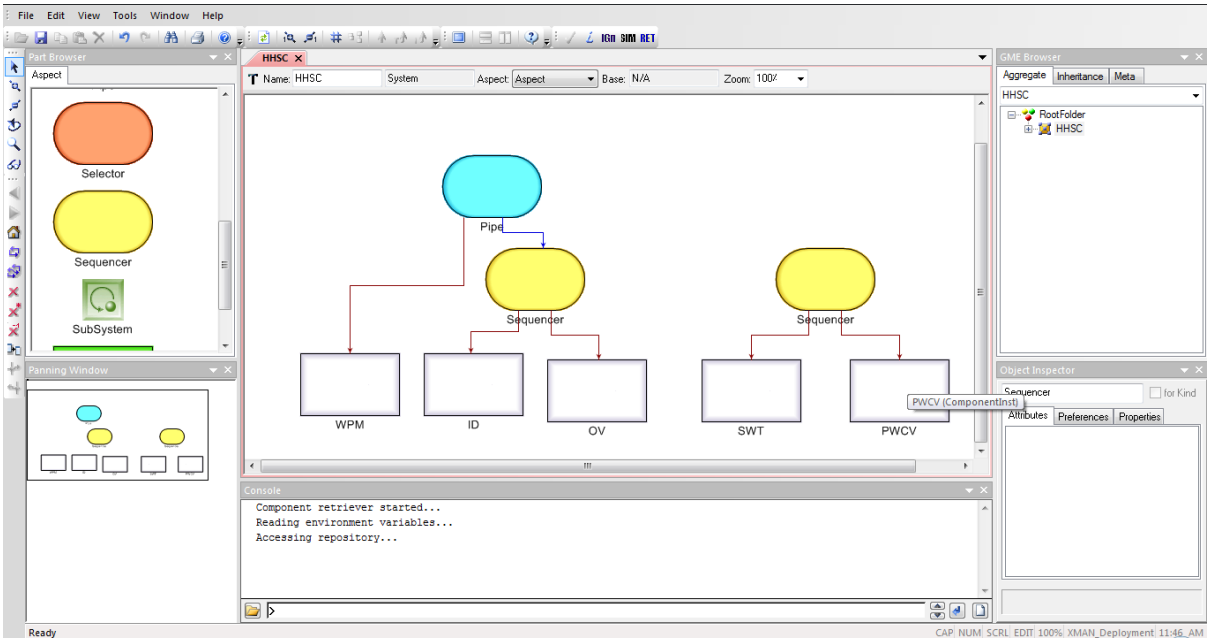


Figure 21: The Partial architecture from R.1.3

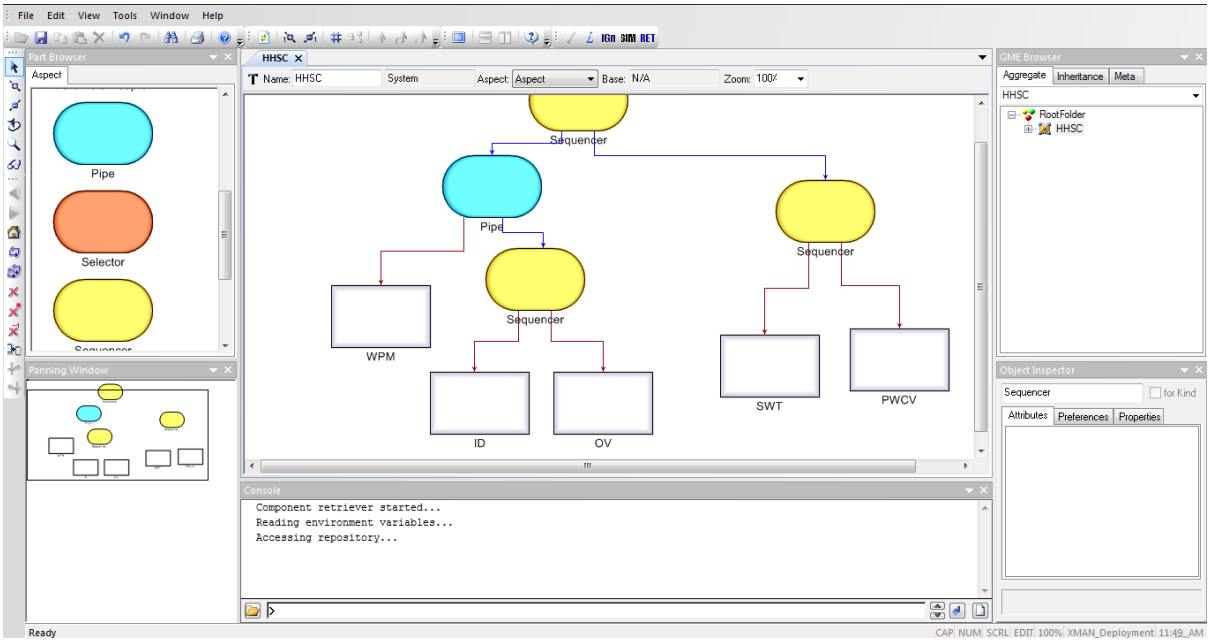


Figure 22: The final system Architecture

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Chapter 6: *Conclusion*

This chapter gives an overview of the component based mapping requirements approach and investigates whether the outlined objectives of this approach have been successfully met. The limitations, challenges and future possibilities will be also be explored in brief detail.

6.1 Accomplishments

This project has defined a systematic approach for mapping raw requirements expressed in natural language to a component-based architecture through four phases. The systems are directly constructed from the raw requirements based on the XMAN component model. There are a number of factors that make the XMAN component model ideal for the mapping process:

1- The reusability.

We have seen that a component can be mentioned numerous times in the raw requirements of a system. Reusability is one of the most important features of component-based software development and the XMAN component model is one of these models. In this approach we search for the component in the repository, if it does not exist then we design it. If a raw requirement contains a

component that was mentioned in a previous requirement we simply retrieve it and add it appropriately to the partial architecture.

2- **The XMAN Component model enables incremental system construction.**

As stated previously, unlike the behaviour tree approach which enables constructing the behaviour trees incrementally from the raw requirements, the XMAN component model allows incremental construction of the actual system from the raw requirements.

3- Encapsulation of computation and control.

The encapsulation offered by the XMAN component model guarantees that each component which is extracted from the new requirement and added to the partial architecture will not alter the behaviour of the architecture.

The objectives of this project have been met and proved by applying this approach on the heating home system controller example. Some restrictions and limitations have been identified during the application of this process. These limitations and challenges will be discussed in the next section.

6.2 Limitation and Challenges

Although a systematic approach for mapping raw requirements to a component-based architecture is defined and all requirements have been satisfied, a number of limitations and challenges have been encountered. Firstly, an automated tool that edits and analyses the requirements and then provides the extracted information to the XMAN tool needs to be implemented. The limited time available for the project did not allow for such an implementation. However,it should also be noted that this approach still needs human knowledge and intervention in the selection of appropriate components and in coding the component's behaviour (computation). The defined rules in this approach make the mapping process more systematic and clear. Another challenge that was encountered in this approach was the lack of recourses about the XMAN tool because it is a relatively new component model.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Abstract

The success of any software system is evaluated by measuring the degree of achieving the requirements detailed in the natural language raw requirements which is provided by the customer. In software development, the software system developmentprocess always starts from requirements specifications which are defined manually from the raw requirements which are expressed in natural language. This process is performed based on the human knowledge of the considered problem set and ingenuity in understanding how available tools can solve the expressed problem. Therefore, the requirements specification and other intermediate models only give an approximation of the raw requirements depending on the skills of the developer mapping the system specification from the raw user requirements in natural language. This project attempts to define an approach that maps raw requirements directly into component based architecture by using a component based model which supports incremental composition. Each stated requirement will be immediately mapped into executable components that build a partial architecture which compose to the system architecture. The aim is to bring about greater congruence between the raw requirements given by the customer in natural language and the software system which is eventually developed.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Table of Contents

Table of Figures

Table of Tables

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

MAPPING REQUIREMENTS DIRECTLY TO COMPONENT BASED SOFTWARE ARCHITECTURE

A THESIS
SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2011

SOZAN A. ALJOAHNY

SCHOOL OF COMPUTER SCIENCE

Table of Contents

Table of Figures

Table of Tables

Abstract

The success of any software system is evaluated by measuring the degree of achieving the requirements detailed in the natural language raw requirements which is provided by the customer. In software development, the software system developmentprocess always starts from requirements specifications which are defined manually from the raw requirements which are expressed in natural language. This process is performed based on the human knowledge of the considered problem set and ingenuity in understanding how available tools can solve the expressed problem. Therefore, the requirements specification and other intermediate models only give an approximation of the raw requirements depending on the skills of the developer mapping the system specification from the raw user requirements in natural language. This project attempts to define an approach that maps raw requirements directly into component based architecture by using a component based model which supports incremental composition. Each stated requirement will be immediately mapped into executable components that build a partial architecture which compose to the system architecture. The aim is to bring about greater congruence between the raw requirements given by the customer in natural language and the software system which is eventually developed.

Declaration

No portion of the work referred to in this dissertation has been submittedin support of an application for another degree or qualification of this orany other university or other institute of learning.

Copyright

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written

permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/display.aspx?DocID=487), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University's Guidance for the Presentation of Dissertations.

Acknowledgments

It is a pleasure to thank my supervisor, Dr. Kung-Kiu Lau, for his support and guidance and for his help when I needed him. I would also like to thank my family for providing support during my year of study. I would also like to thank Cuong M. Tran and AzlinNordinfor their guidance andsupport throughout the project.

Chapter 1: *Introduction*

Developing a software system that meets the purpose for which it was proposed is the main concern for any software developer. Requirements analysis is the first and the most critical phase of the software system development. The raw requirements can be considered as a commitment between the software system developers and the customer who requested the system under development. The developers always work hard to achieve the customer's aspirations by implementing a software system that contains all the business processes which are explicitly stated and implied in the raw requirements.

Although, the raw requirements are the most influential association between the developers and the customer, the software system’s development process does not originate from natural language raw requirements specified by the customer. The requirements specifications which are engineered from natural language raw requirements can be considered as the basis and the first step of software systems construction in software engineering.

Human knowledge and ingenuity is the only resource that can be used to define the requirements specifications and intermediate models such use cases [12]. Therefore, these requirements specifications and intermediate models do not cover the raw requirements exactly; they, at best, only approximate them [4].

The main objective of this project is to discover a systematic approach that processes raw requirements which are expressed in natural language and to extract the information that helps in constructing the component-based software system architecture directly. There is relevant research which is concerned with the analysis of raw requirements and associating them to elements that relate to software units in order to achieve a better match between the final system and the raw requirements.

Current reliance on human knowledge and ingenuity in mapping out system specifications from natural language raw requirements limits software development to the skills of an individual [12]. There is evidently a need for a better approach. This approach should be one that is systematic and relies less on individual human skill.

A component based approach that maps directly from raw user requirements in natural language to executable components is a viable answer to this problem. This is the aim of this project. This paper aims to describe a systematic approach to mapping user requirements directly into executable components.

Tied to this is the notion of architecture in which we will discuss viable and function architectural systems that allow component addition to partial architecture in an incremental fashion.

Using the approach described in this paper we should be able to use raw requirement to immediately select components from the repository or develop the components and deposit them in a repository. This would then be followed by constructing a partial architecture and then compose it with the system architecture after that returning to the raw requirements and carry on in that manner. Now clear the system must allow for extensibility and that is the detail that this paper goes into. This paper will also describe how we will use existing technologies to parse user raw user requirements for valid content and how this is then employed in deciding which component to make use of.

The foundation of this paper is that an individual raw requirement can be directly mapped to executable software components. In addition to this, it is also possible to develop a systematic manner in which to join these components together whilst allowing flexibility for any other requirement to be added into the system at any stage of development [7]. In other words, the system should allow incremental development such that one does not need to read the full natural language raw requirements before starting development.

This paper will also go in quite some detail on the XMAN tool that will be used as our component based model. The workings of this approach are discussed in some detail with a special focus on the tools available for building partial architectures and the use of components in the repository and adding components to the same.

Some of this research is based on object-oriented software development while this project approach intends to use component-based software development. The component-based development differs from the object-oriented software development in many features. This report will investigate these differences and will review the literature of related works such as behaviour tree approach and object-based mapping approach.

Chapter 2: *Software System's requirements*

In developing any piece of software the driving force are the stakeholders and the goal of developers is to supply stakeholders with software that meets their specific needs. The issue of software system requirements is concerned with ascertaining the requirements that a client has and then developing a piece of software that meets those requirements. A number of issues arise during this endeavour of ascertaining requirements. Sometimes the client does not know what they actually want. Sometimes requirements are incomplete or perhaps ambiguous. Software system requirement techniques are concerned with overcoming these challenges. We are particularly interested in how we derive formal specifications for an informal requirements document that is written in natural language. How do we map the required functionality from the language of the client to a developer perspective? This chapter touches on these particular issues, addressing the different aspects of software requirements analysis.

The Software Requirements Definition document sets out the functional requirements of the software under development [13]. This document should be drawn from a reading and interpretation of the Business Functional Requirements definition document. Before commencement of actual development work the Software Requirements Definition document must be fully documented, approved and signed by all stakeholders.

To minimise risk, no actual programming beyond conceptual demonstrations and proof of concept mockups should be undertaken until the Software Requirements Definition is approved and signed off.

As already mentioned, the Software Requirements Definition is drawn from the desired business functionality as laid out by the client. The first stage in developing the definition is in setting out a clear definition of what the software is required to do [13]. An exhaustive process of project requirements gathering must be undertaken. A detailed exploration of project requirements gathering is outside the scope of this paper.

The project requirements gathering stage gives a users perspective to the software. The developers must then examine these requirements and build a ‘logical model' by using recognised methods and specialist knowledge of the

problem domain. The logical model is a high level abstraction describing system abilities and should be free from implementation technology [14]. This model gives structure to the problem set giving it greater manageability.

The logical model is then used in the production of an ordered set of software requirements. These requirements would specify the level of functionality, detail performance, set out available interfaces, give assurances over quality and reliability etc.

This document sets out the developers' view of the problem set as opposed to that of the user. The relationship between the Software Requirements Definition document and the Business Functional Requirements is not necessarily one-to-one and often is not.

It is very important that all the stakeholders agree on one consistent view of the various requirements of the system. The development team will interpret the software requirements from the user perspective and express this in the developers view in the form of the Software Requirements Definition. There may be a disparity between the two which is why it is essential that all stakeholders approve and sign off the Software Requirements Definition document before any actual development work begins.

2.2 Requirements Types

Most software requirements can be categorised into the following: *Architectural Requirements*, *Functional Requirements*, *Non-Functional Requirements* and *Constraint Requirements* [15].The following section briefly examines each of these requirements.

2.2.1 Architectural Requirements:

This is a high level description of what must be done. It identifies the system architecture of a system that is to be developed. The architectural requirements are primarily concerned with the shape of the solution space. They establish the structure of a solution to a set of problems imposed by a set of requirements.

A distinction must be set between Architectural Instance and Architectural Family. An architectural instance gives a high level abstraction of a software system. A system architecture would describe, at the very least, how the system is broken into different components and how those components work together, carefully setting out dependencies and so on [2]. An effective architecture would expose the crucial properties of a system. Here we can define crucial as those properties that must be considered for an effective reasoning of the system to be carried out.

Contrast the architectural instance with an architectural family, an architectural family sets out constraint definition on a group of associated systems. Architectural families can be merely generic idiomatic patterns and styles (for example, "pipe and filter" or "client-server organisation") and can be reference architectures (for example, "OSI layered communication standard"). An effective architecture is one that ensures a measure of integrity constraint but also permitting a measure of flexibility that subsumes the family of systems to be built over the life-time of the product family. Architectural requirements are established by the developers and system architects, not the user.

2.2.2 Functional Requirements

A functional requirement defines the function of a software system or component [15]. Functional requirements could refer to data manipulation, calculations or data processing that define what a system is supposed to accomplish. Functional requirements are often expressed as, "the system must do<this>" and "the system must do<that>". Functional requirements are a high level expression.

It is a system/software requirement that specifies a function that a system/software system or system/software component must be capable of performing. These are software requirements that define behaviour of the

system, that is, the fundamental process or transformation that software and hardware components of the system perform on inputs to produce outputs.

A functional requirement will often have a unique name and identifier, a brief description and a rationale. The key point is the description of the required behaviour; this must be clear and easy to understand. This type of requirements is concerned by this research.

2.2.3 Non-Functional Requirements

Non-Functional Requirements are often described as quality requirements. These are characteristics of the software system that the user is not able to perceive. It is a software requirement that does not describe what the software does, but how it will do it. An example would be software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Non-functional requirements are difficult to test; as such they are often evaluated subjectively.

Non-functional requirements will often, if not always, take a descriptive tone; for example, “the system shall be <requirement>”. An example following this would the following requirement: “the system shall be <easy to navigate>”.

2.2.4 Constraint Requirements

Better addressed as “constraints,” these are merely restrictions within which the software under develop must operate or be developed under. For example a requirement that states, “software must be ready for developed before year 2000” would be a constraint. This would be a project constraint. Constraints often refer to non-functional requirements. An example would be a requirement that demands that the application "require no more than 100mb of hard drive space during installation" or that "the application must gracefully degrade on older browsers."

2.3 Attributes of Good Requirements

The IEEE standard stipulates that a Software Requirement Document must satisfy the following.

a.) Functionality - This should state clearly exactly what the software should do and, if there is scope for ambiguity, what the software should not do.

b.) External Interface - This part of the specification should detail how the system will interact with people (human computer interaction), the system's hardware and other software.

c.) Performance - These requirements regard speed, system availability, speed of response and recovery time of various functions.

d.) Attributes - These are extensibility, maintainability, security, usability, correctness, etc. considerations.

e.) Design constraints - These requirements address required standards, implementation language, database integrity policies, resource limitations, operating environment, etc.

Even after these types of requirements have been laid out it is important that they conform to the rigors again imposed by the IEEE Standard. The following are qualities of good requirements.

a.) Correct - This much is largely self explanatory. The requirements should state what is actually meant by the client.

b.) Unambiguous - The requirement must have one interpretation. Any ambiguities must be highlighted and the actual desired requirement stated explicitly.

c.) Complete - All the requirements necessary for the software to be operational must be stated.

d.) Ranked for importance - Requirements that are fundamental to the operation and success of the software should be listed above aesthetic and non-crucial requirements.

e.) Verifiable - Requirements should avoid subjectivity. Instead of requiring the software to simply be "fast," requirements should state "on form submission user should receive response in no more than 600 milliseconds."

2.4 Requirements analysis and software design

Requirements analysis is the process of investigating the properties of a specification and developing an initial software model [12]. It describes the set of tasks involved in determining the exact needs or conditions that need to be met to satisfy the requirements of a client. There are a number of recognized techniques in the literature on how best to conduct Requirements analysis. A full a in-depth examination of all these techniques is outside the scope of this paper. However this section will briefly examine some of the key techniques in use. This much is necessary for the sake of comparison with our own component based direct mapping approach that will be introduced in a later chapter.

Use Cases can be used in requirements analysis. A use case is a structure for documenting the functional requirements of a piece of software. In each use case a scenario that shows how the system will interact with humans or other systems is provided. Use cases are often developed by requirements engineers in conjunction with stakeholders. Use cases simply show the steps needed to accomplish a task, they do not show the workings of the system or how the task will actually be implemented at a development level.

Use cases are just one example of requirements analysis but already an important question arises. How do we move from have the raw user requirement in natural language and begin to move toward an more software oriented expression? How do we manage to correctly draw fromt he raw requirements what exactly it is that the stakeholders require? A great measure of this relies on human ingenuity and an understanding of the problem domain.

Semantic case analysis is an effective way of moving from raw requirements in natural language to a position where the stakeholders requirements are actually established. This is especially true in object oriented analysis of software requirements.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

REFERENCES

[1] Russell J. Abbott, Program Design by Informal English Descriptors, Communications of the ACM, November 1983, Volume 26, Number 11

[2] R G Dromey, Software Quality Institute, Architecture as an Emergent Property of Requirements Integration

[3] Kung-Kiu Lau and Zheng Wang, Software Component Models, IEEE TRANSACTION ON SOFTWARE ENGINEERING, VOL.33, NO.10, OCTOBER 2007

[4] C Rolland, C Proix, A Natural Language Approach for Requirements Engineering

[5] Motoshi Saeki, Hisayuki, Hajime Enomoto, Software Developmemt Process from Natural Language Specification, Tokyo Institute of Technology

[6] Brian W. Kerningham and Dennis M. Ritchie, The C Programming Language, Prentice-Hall 1988

[7] Kung-Kiu Lau, Ling Ling and Zheng Wang, Composing Components in Design Phase using Exogenous Connectors, School of Computer Science, The University of Manchester

[8] Kung-Kiu Lau, Azlin Nordin and Keng-Yap Ng, Extracting Elements of Component-based Systems from Natural Language Requirements, School of Computer Science, The University of Manchester

[9] Dongyun Liu and Hong Mei, Mapping Requirements to Software Architecture by Feature-Orientation, Institute of Software, School of Electronics Engineering and Computer Science, Peking University

[10] Grady Booch, Object-Oriented Development, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986

[11] Systems Engineering Fundamentals, Department of Defense, Systems Management collection, January 2001

[12] Terry Anthony Byrd, Kathy L. Cossick and Robert W. Zmud, A Synthesis of Research on Requirements Analysis and Knowledge Acquisition and Techniques

[13] Guide to the Software Definition Phase, ESA Board for Software Standardisation and Control, ESA PSS-05-03 Issue 1 Revision 1, March 1995

[14] Martin Maguire, User Requirements Analysis, Research School in Ergonomics and Human Factors

[15] Requirements: An Introduction, IBM DeveloperWorks, 16 Apr 2004

[16] Ivica Crnkovic, Séverine Sentilles,Aneta Vulgarakis and Michel Chaudron, A Classification Framework for Component Models, Mälardalen University, Sweden

[17] K.-K. Lau, A. Nordin, T. Rana, and F. Taweel. Constructing component-based systems directly from requirements using incremental composition. In Proceedings of the 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA '10, pages 85–93, Washington, DC, USA, 2010. IEEE Computer Society

[18] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: the penn treebank. Comput. Linguist., 19:313–330, June 1993

[19] A. D. Conlin Potts, Mark Carlson, et al. Requirements analysis project. Technical report, 2000.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

Appendix A- The HHSC Requirements

3. Functional Requirements

3.1 The HHSC will ACTIVATE THE FURNACE.

3.1.1 The HHSC will signal the water pump motor to start.

3.1.2 Once motor speed reaches 1000 rpm, the HHSC will signal the ignition device to be activated and oil valve to be opened.

3.1.3 Once the system water temperature reaches a value pre-defined by the user, the HHSC will signal the primary water circulation valve to be opened.

3.1.4 The HHSC will retain the length of time from the last DEACTIVATION and not REACTIVATE THE FURNACE until a period of 5 minutes has elasped.

3.2 The HHSC will DEACTIVATE THE FURNACE.

3.2.1 The HHSC will signal the oil valve to close.

3.2.2 Once the oil valve is closed, the HHSC will shut down the furnace.

3.2.3 Five seconds after signaling the oil valve to close, the HHSC will signal the water pump motor to stop.

3.3 The HHSC will signal the home heating system to heat the home.

3.3.1 When the temperature sensor in any room indicates a temperature less than t R – 2 °F (where tR is the desired temperature for that room, set by the user) and the Master Switch is set to ON, the HHSC will ACTIVATE THE FURNACE (unless it is already on) and open the water circulation valve for that room.

3.4 The HHSC will signal the home heating system to cease heating the home.

3.4.1 When the furnace is on and the temperature sensor in every room indicates a temperature greater than tR + 2 °F (where tR is the desired temperature for that room, set by the user), the HHSC will close all water circulation valves that are open and

then DEACTIVATE THE FURNACE.

3.4.2 When the furnace is on and the temperature sensor in a room indicates a temperature greater than tR + 2 °F while the temperature sensor in at least one other room indicates a temperature less than tR + 2 °F (where tR is the desired temperature for that room, set by the user), the HHSC will close the water circulation valve for the room in question.

3.5 The HHSC will avoid unsafe furnace operating conditions

3.5.1 When the Master Switch is set to OFF, the HHSC will DEACTIVATE THE FURNACE within 5 seconds.

3.5.2 The HHSC will ensure that the furnace is shut down in the event of abnormal fuel oil flow.

3.5.2.1 When the oil valve sensor indicates CLOSED, the HHSC will DEACTIVATE THE FURNACE within 5 seconds.

3.5.2.2 When the oil flow sensor indicates a flow less than 0.01 m /sec, the HHSC will DEACTIVATE THE FURNACE within 5 seconds

3.5.2.3 If abnormal fuel oil flow is indicated while the furnace is on, the HHSC will DEACTIVATE THE FURNACE.

3.5.2.4 If abnormal fuel oil flow is indicated while the furnace is shut down, the HHSC will prevent ACTIVATION OF THE FURNACE.

3.5.3 The HHSC will ensure that the furnace is shut down in the event of abnormal fuel combustion.

3.5.3.1 If abnormal fuel combustion is indicated while the furnace is on, the HHSC will DEACTIVATE THE FURNACE.

3.5.3.2 If abnormal fuel combustion is indicated while the furnace is shut down, the HHSC will prevent ACTIVATION OF THE FURNACE.

العنوان:	Mapping Requirements Directly to Component Based Software Architecture
المؤلف الرئيسي:	Al Joahny, Sozan A.
مؤلفين آخرين:	Lau, Kung Kiu(Super.)
التاريخ الميلادي:	2011
موقع:	مانشستر
الصفحات:	1 - 93
رقم MD:	601673
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	University of Manchester
الكلية:	Faculty of Engineering and Physical Sciences
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسبات، البرمجيات، تصميم النظم
رابط:	https://search.mandumah.com/Record/601673

MAPPING REQUIREMENTS DIRECTLY TO COMPONENT BASED SOFTWARE ARCHITECTURE

A THESIS
SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2011

SOZAN A. ALJOAHNY

SCHOOL OF COMPUTER SCIENCE