A Knowledge-Based COTS-Aware Requirements Engineering Approach

Lawrence Chung
Dept. of Computer Science
The University of Texas at Dallas
P.O. Box 830688
Richardson, TX, USA
chung@utdallas.edu

Kendra Cooper
Dept. of Computer Science
The University of Texas at Dallas
P.O. Box 830688
Richardson, TX, USA
kcooper@utdallas.edu

ABSTRACT
The goals of developing systems better, faster, and cheaper continue to drive software engineering practitioners and researchers to investigate software engineering methodologies. In requirements engineering, the focus has been on modeling the software engineering process and products for systems that are being built from scratch. As the size and complexity of systems continues to grow the use of commercial off-the-shelf (COTS) components is being viewed more and more as a promising, and also perhaps inevitable, solution. The effective use of COTS components, however, requires a systematic approach that provides a set of concepts for modeling the subject matter, a set of guidelines for using such concepts, and tool support to assist the developer.

In this paper, we present a COTS-Aware Requirements Engineering (CARE) approach that explicitly supports the use of COTS components. Our CARE approach is knowledge based, has a defined process, and is agent- and goal-oriented. In particular, we present part of our meta-model for the knowledge base and a prototype of the CARE Assistant Tool. Our CARE approach is validated using a part of a Digital Library System example.

Keywords
requirements engineering, COTS components, knowledge-base

1. INTRODUCTION
The goal of developing systems better, faster, and cheaper continues to drive software engineering practitioners and researchers to investigate software engineering methodologies. In requirements engineering, the focus has been on describing the functional requirements for systems that are being built from scratch. As the size and complexity of systems continues to grow the use of commercial off-the-shelf (COTS) components is being viewed more and more as a promising, and also perhaps inevitable, solution to this problem. Effective use of COTS components, however, requires a systematic methodology that would provide both a set of concepts for modeling the subject matter and a set of guidelines for using such concepts. Our work is focused on creating, modeling and analyzing a requirements engineering approach that explicitly supports the use of commercial off-the-shelf (COTS) components. Our approach is called the COTS-Aware Requirements Engineering (CARE) approach. It is goal-oriented (as in [2],[12],[26]), agent-oriented (as in [25],[28],[29]), has a defined process (as in [20]), and is knowledge-based (as in [19],[24]). The CARE approach is intended to assist the requirements engineer in developing high quality specifications, not to replace their intelligence and experience.

Models are used to embody abstract ideas in a concrete representation. In a model, the ideas are more easily communicated, reviewed, and revised. The properties we seek in a model include completeness, consistency, correctness, etc. We also recognize that a model needs both an ontology and a methodology. An ontology is a formal description of entities and their properties; it forms a shared terminology for the objects of interest in the domain, along with definitions for the terms. A methodology describes how the entities are used or created.

The CARE approach is being defined with a set of models and is supported with a prototype of the CARE Assistant Tool. The models include a process, product, and a meta-model [9]. The process model describes the activities performed to define the system agents, goals, system requirements, software requirements, and software architecture (all with respect to using COTS components). Each activity defines who performs it, the steps, constraints, inputs, and outputs. The product model describes the format of the output, or products, created using the process. For example, a well-known product model for software requirements is UML’s Use Case. The CARE meta-model defines the ontology for the approach. Entities include agents, goals, system requirements, software requirements, software architecture, COTS components, assertions, and activities.
The meta-model is extremely important because it describes the content and structure needed in the knowledge base for the CARE approach. First, the ontology is modeled in an object-oriented framework using UML: each construct is modeled as a class with data attributes; the classes are related to one another with associations and generalizations. Subsequently, the constructs are represented in a formal notation, O-Telos, in our knowledge based tool support. This representation allows us to reason about the entities. For example, we can reason about contradictory, non-functional goals using the Non-Functional Framework (NFR) [12] [21].

We recognize that developing the CARE approach is an ambitious project. To accomplish the work, we are using an iterative approach to define, refine, and validate the CARE models and tool support [10]. This iteration is focused on the knowledge base. We define a meta-model and prototype the knowledge base of the CARE Assistant Tool. The approach is validated by applying it to (part of) a Digital Library System (DLS) example. A DLS is selected as the example because it is a complex, large-scale system with a rich set of functional and non-functional requirements. A detailed description of the CARE approach (including the process model, product model, meta-model, and ontology) is available in [9].

This paper is organized as follows. Following the introduction, we describe related work in Section 2. An overview of the CARE approach is presented in Section 3. In Section 4, the knowledge base for the CARE approach is presented. Some of our lessons learned are described in Section 5. Conclusions and future work are in Section 6.

### Figure 1. Mapping System Goals and Requirements to Component Goals and Specifications

![Figure 1](image-url)
typically represents the physical packaging of classes, interfaces, and collaborations [18].

The Model Based Architecting and Software Engineering (MBASE) approach considers four types of models: success, process, product and property [4] and is consistent for use with COTS components [5]. MBASE uses four guiding principles to develop value-driven, shared-vision-driven, change-driven, and risk-driven requirements. The Procurement Oriented Requirements Engineering (PORE) technique supports the evaluation and selection of COTS components [20],[23]. The PORE process model identifies four goals in a thorough COTS selection process: a) acquiring information from the stakeholders, b) analyzing the information to determine if it is complete and correct, c) making the decision about product requirement compliance if the acquired information is sufficient, d) selecting one or more candidate COTS components. The PORE approach also defines a meta-model that is composed of the following constructs: agents, actions (these are performed by agents), events (these trigger actions), goals (these are accomplished by actions), objects in the domain, state of an object, state transitions of objects, components, and functions provided by the product.

3. THE CARE APPROACH

An overview of the CARE approach is provided in this section (refer to Figure 1) and it is illustrated with examples from the Digital Library System. The reader is referred to [9] for a detailed description. The CARE approach is characterized as goal-oriented, agent-oriented, knowledge based, and has a defined process.

Using the CARE process, two of the first activities are to define the agents and the goals for the system under development (refer to Figure 2). The agents (stakeholders) are characterized by a set of properties [25],[28],[29]. The agent is intentional and possesses intentional properties including beliefs, goals, abilities, and commitments. An agent is autonomous and makes decisions and choices. An agent depends on other agents to accomplish goals, complete tasks, or furnish resources. This characteristic makes an agent able to accomplish goals they could not achieve on their own. At the same time, however, it makes an agent vulnerable because it depends on other agent. Lastly, an agent is strategic. This allows the agent to analyze its opportunities and risks in the various configurations of a system.

An example of an agent in the DLS is a junior librarian. A junior librarian is a user who is associated with the customer. The junior librarian interacts with developers to create proposed system and interacts with proposed system to provide reference assistance to borrowers. The junior librarian is committed to a) validating the definitions of the system agents, goals, system requirements, and software requirements for the RE. b) providing reference assistance to the borrowers. The junior librarian is trained in library science and has 0-2 years experience. The RE depends upon the junior librarian to provide information about the goals and requirements of the library system.

System goals are very high-level objectives of the system. The

![Figure 2. A High Level View of the CARE Process](image-url)
may be functional (hardgoals) or non-functional (softgoals). Once defined, these goals are used to drive the development of the system. Goals are refined into system requirements (which in turn may be refined into software requirements). Goals are related to the other goals of the system using the NFR framework [12], [21]. This framework has four kinds of relationships: makes (very positive), helps (positive), hurts (negative), and breaks (very negative). Goals are mapped to COTS components when a (possible) match is found in the repository.

An example of a goal in the DLS is "The DLS CHI should be configurable". Using the NFR framework, this goal is characterized as hurting another DLS goal that is to deliver a moderately priced system. The goal is refined into a number of system requirements including the statement: "The DLS CHI shall be configurable to enable or disable context sensitive help".

When mapping goals and requirements to possible matches of COTS components, the requirements engineer (RE) searches, matches, and selects from the components in the repository. The RE considers the functional and non-functional capabilities and, with the assistance of the software architect (SA), the impact of using a component on the software architecture. If a match is not found, the RE has a number of options. The RE may request that the component engineer (CE) attempt to find another component available in the marketplace and add it to the repository. The RE may try bridge the gap between the customer's needs and the capabilities available in the components by asking the vendor to make a change to a component or asking the customer to change a goal or requirement.

4. THE CARE KNOWLEDGE BASE

The CARE knowledge base uses a meta-model that is organized into a World model, an Interface model, and a System Model [11]. (refer to Figure 3). The World model describes the environment within which the system is going to function. The System model describes the capabilities (functional and nonfunctional) that the system under development is going to provide. The Interface model describes the interactions between the world model and the system model.

Each of the models is defined in terms of object oriented concepts: classes with data attributes, associations, and generalizations. To emphasize the agent and the goal orientation, (recall that the CARE approach is agent- and goal-oriented) the agent classes are coloured blue and the goal classes are green.

The meta-model contains constructs for the process and the product model. For example, agents associated with the process model include the requirements engineer, software architect, and component engineer. For the product model, the constructs include system goals, system and software requirements.

4.1 World Model

The most general class in the World model is the Agent class. Agents are specialized into Enterprise Agents (i.e., Agents that are associated with a company). Enterprise Agents are specialized into agents involved in developing components and agents involved in developing the customer's system. System agents are specialized into agents associated with the component vendor, development house, or customer for the proposed system. The system agents may have one or more assertions (e.g., A system agent must have at least one system goal) and have one or more goals. The system agents accomplish the goals by performing activities. To obtain knowledge about the System model, the agents use queries.

4.2 System Model

The most general class in the System model is the Goal class. Goals are specialized into Enterprise Goals (i.e., the business...
goals for a company). Enterprise Goals are specialized into component goals and customer goals. A customer's enterprise goal may be accomplished by people and/or a system. Goals allocated to a system are specialized into hardgoals (functional goals) and softgoals (non-functional goals). The system goals may have one or more assertions (e.g., A system goal must trace to at least one system requirement).

4.3 Interface Model
The Interface model contains Relationships between System Agents and System Goals, Activities and System Goals, Component Agents and Component Goals, and Queries. A rich set of queries is needed to support obtaining information from the knowledge base. For example, when a system agent is performing the activity to map a requirement to a potential COTS component, the agent uses a query to search the knowledge base for information on components. The search may be keyword- or case-based.

5. CARE ASSISTANT TOOL
The CARE Assistant Tool is specified to implement the knowledge base described in the previous section. A prototype of the tool has been developed which includes a subset of the constructs in the meta-model. The need for a prototype has been driven by the evolving requirements for the CARE Assistant Tool and the selection of a new development tool, ConceptBase [13], for the implementation. After considering these issues, a Spiral model is being used to develop the CARE Assistant Tool.

5.1 Tool Specification and Architecture
The goals, requirements, and software architecture for the CARE Assistant Tool are specified. The specification has 12 goals and 146 functional and non-functional requirements. The software architecture for the tool is based on two styles: client server (to support remote access) and layered (to provide additional structure within the client and the server). The client has the following layers: Browser Interface, Client Manager, Communication, O/S Utilities (e.g., sockets). The server has the following layers: Knowledge Base, Knowledge Base Manager, Server Manager, Communication, and O/S Utilities.

The Knowledge Base layer contains a package for the World Model, a package for the System Model, and an interface for the Interface Model. The World model package is refined into classes for the Agents and the Activities. The System model package is refined into classes for the Goals, System Requirements, Software Requirements, Components, etc.

5.2 Tool Implementation
A prototype of the CARE Assistant Tool has been implemented using capabilities available in the ConceptBase tool and its Graphic Browser. ConceptBase supports the O-Telos language [22], a descendent of RML [15] - an object oriented requirements modeling language. O-Telos is a formally defined notation.

Using ConceptBase, the classes for the knowledge base and instances for the DLS example are defined. Two examples are presented below. Another example is available in Appendix A (it defines a component class and an instance, the Java Media Framework). The first definition is for a junior librarian, one of the user agents (refer to Figure 4 for a graphical representation):

```
Individual junior_librarian in UserAgentClass with
  User_ID: "A_001"
  Association
    User_Association:"Library Company Inc."
```

![Figure 4. A Partial View of Agents, Goals, and Components in the DLS Knowledge Base](image-url)
6. LESSONS LEARNED

We have already learned a number of lessons in developing this phase of the work. The formalization into O-Telos class definitions has encouraged the development of a complete, correct model. As the definitions were being developed, it became an iterative process to define, validate, and correct the definitions. Attributes that were initially defined, but later appeared to have little value in the validation with the DLS example, were deleted. An example of this case was in the System Agent class. Initially, the name of an agent was included as an attribute. In the validation, it became clear that the role the agent was playing (e.g., junior librarian) was more relevant.

The prototyping effort, driven by evolving requirements and the adoption of a new tool, proved to be a valuable decision, at least from a pragmatic point of view. The time needed to learn the use of ConceptBase and the Graphical Browser were greater than expected. Also, it became clear as the prototyping work continued that the Graphical Browser is an excellent aid to get started. However, it does not have the flexibility the CARE Assistant Tool needs. ConceptBase provides an API and we will investigate this capability in the tool in the next step.

7. CONCLUSIONS AND FUTURE WORK

The CARE research project is defining CARE process, product, meta-model, and tool support concurrently. The intent is to have a set of models and tool support that work together. Additionally, as each part of the work matures, it can be used to detect problems in the other parts.

In this paper, our contribution is to present (part of) the knowledge base for the CARE approach. A more detailed description of the CARE approach (process model, product model, meta-model ontology) is available in [9]. We present our meta-model and, as a proof of concept, at least partially demonstrate a prototype of the CARE Assistant Tool. The meta-model is extremely important.

![Figure 5. Relationships Among Goals in the DLS Knowledge Base](image-url)
because it describes the content and structure needed in the knowledge base for the CARE approach. First, we modeled the ontology in an object-oriented framework using UML, but extended with agent- and goal-oriented frameworks. Subsequently, we represented it in our CARE Assistant Tool’s knowledge base using a more formal notation, O-Telos, so as to reason about the ontological concepts more automatically. For example, with an O-Telos representation, we can reason about contradictory, non-functional goals.

Our next step in the work is to refine the CARE approach’s process, product, and meta-model and tool support. More specifically, we propose to a) extend the prototype to support reasoning about contradictory, non-functional goals as in the NFR framework, b) develop a better user interface using the API in ConceptBase for the CARE Assistant Tool, and c) extend the knowledge base with additional component definitions.

ACKNOWLEDGEMENTS
We would like to thank the members of the Embedded Software Center, including Drs. Bastani and Deng for their moral support while this project has been going on for almost two years as well as the Center’s industrial partners.

REFERENCES
Appendix A. Component Class and Instance

Example definitions of the COTSComponentClass and an instance of it, the JMF, are provided here.

COTSComponentClass in Class is A SystemModelClass with attribute

- UniqueIdentifier: String;
- Type: String;
- Name: String;
- Keyword1: String;
- Weight1: Integer;
- Keyword2: String;
- Weight2: Integer;
- ... 
- Overview: String;
- Domain: DomainClass;
- Vendor: String;
- VendorEvaluation: String;
- VersionNumber: String;
- FunctionalSpecification: String;
- OS: String;
- StandardsCompliance: String;
- Performance: String;
- Security: String;
- Availability: String;
- Memory: String;
- DiskSpace: String;
- LessonsLearnedExperience: String;
- Subcomponent: COTSComponentClass;
- Assumptions: String;
- Notes: String

End

JMF_Component in COTSComponentClass with
UnqiueIdentifier
COTSComponentIdentifier: “C_006”
Type
COTSType: “Software”
Name
COTSComponentName: “Java Media Framework”
Keyword1
COTSComponentKeyword: “multimedia”
Weight1
COTSComponentWeight: 4

Overview
COTSComponentOverview: “The product supports adding audio and video to Java applications and applets. The product extends the multimedia capabilities on the J2SE™ platform”

Domain
COTSComponentDomain1: Distributed
COTSComponentDomain2: nonDistributed

Vendor
COTSComponentVendor: “Sun Microsystems Inc.”

VendorEvaluation
COTSComponentVendorEval: “Leader in the Java language specification and tool support.”

VersionNumber
COTSComponentVersion: “2.1.1”

FunctionalSpecification

OS
COTSComponentOS: “Multiplatform (Solaris, Windows)”

StandardsCompliance
COTSComponentStds:
“Protocols: RTP, ftp, http
File formats:
D indicates the format can be decoded and presented.
E indicates the media stream can be encoded in the format.
read indicates the media type can be used as input (read from a file)
write indicates the media type can be generated as output (written to a file)
AIFF (.aiff)
read/write
8-bit mono/stereo linear D,E
16-bit mono/stereo linear D,E
G.711 (U-law) D,E
A-law D
IMA4 ADPCM D,E
AVI (.avi)
read/write
Audio: 8-bit mono/stereo linear D,E
Audio: 16-bit mono/stereo linear D,E
Audio: DVI ADPCM compressed D,E
Audio: G.711 (U-law) D,E
Audio: A-law D
Audio: GSM mono D,E
Video: Cinepak D
Video: JPEG (411, 422, 111) D
Video: RGB D,E
Video: YUV D,E

MPEG Layer II Audio (.mp2), read only
MPEG layer 1, 2 audio D, read only

MPEG Layer III Audio (.mp3), read only
MPEG layer 1, 2 or 3 audio D read only”

Performance
COTSComponentsPerformance: “Supports the following audio sample rates: 8, 11.025, 11.127, 16, 22.05, 22.254, 32, 44.1, and 48 KHz”