ARTICLE IN PRESS

ELSEVIER

Computer Standards & Interfaces 2177 (2002) 1–9

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

ACASA—a framework for Adaptable COTS-Aware Software Architecting

Lawrence Chung*, Kendra Cooper, Stephen Lee, Faisal Shafique, Anna Yi

Department of Computer Science, The University of Texas at Dallas, Dallas, TX, USA

Abstract

The use of Commercial-Off-The-Shelf (COTS) components presents a great promise, as well as challenges and risks. In this paper, we describe our ongoing research on developing adaptable software architectures using COTS components. In particular, we describe an Adaptable COTS-Aware Software Architecting (ACASA) framework that addresses the concerns of the various stakeholders of the proposed system in the presence of COTS components, with a special emphasis on adaptability as a nonfunctional requirement. The ACASA framework is illustrated by way of a telepresence system example.

Keywords: COTS; Nonfunctional requirements; Adaptability; Software architecture

1. Introduction

The use of Commercial-Off-The-Shelf (COTS) components offers a great promise, since it has the potential to drastically reduce software production time and cost through the reuse of ready-made components. The use of COTS components, however, also seems to present greater challenges and risks. It confines the potential design space by the availability of the COTS components and, more fundamentally, requires the determination that such components can indeed fulfill the requirements of the system under consideration, and to what extent.

In this paper, we describe our preliminary research on developing adaptable software architectures using COTS components. In particular, in this paper, we present ACASA [7], a framework for adaptable COTS-Aware Software Architecting. ACASA's goal is to develop a software architecture which can more easily accommodate changes in the stakeholder requirements while retaining earlier, or even expanding, the use of COTS components.

ACASA specializes our COTS-Aware Software Architecting (CASA) framework by putting emphasis in the adaptability aspect of software systems. CASA is an extension of our earlier and ongoing work on COTS-Aware Requirements Engineering (CARE) [6,8]. CASA aims to address the concerns of the various stakeholders of the projected system using CARE, and uses the resulting requirements specification in developing candidate software architectures. This process is incremental and the two phases (requirements and architecture) interleave.

ACASA provides both design- and run-time adaptability. COTS components are evaluated with respect to their *relevance* (excellent, good, etc.) for the system under development (SUD). Run-time alterna-

* Corresponding author.
E-mail addresses: chung@utdallas.edu (L. Chung), kcooper@utdallas.edu (K. Cooper), klee@utdallas.edu (S. Lee), faisals@utdallas.edu (F. Shafique), annayi@utdallas.edu (A. Yi).

0920-5489/02/$ - see front matter
PII: S0920-5489(02)00093-4

tives are highly relevant components that may be chosen during run-time to allow the system to adapt to changing conditions. In the absence of run-time alternatives, another cycle of CASA would have to be repeated to address the adaptability of the software architecture.

Throughout the ACASA process, the decisions and the rationale for making them are documented. For example, when the architect matches and selects components, the evaluation and the reasons for selecting the components as candidates are maintained. When a design-time iterations is needed, the architect uses the development history as an aid.

In related work, the Rational Unified Process (RUP) is an object-oriented software engineering technique [12], which is based on four phases (transition, construction, elaboration and inception) and five core workflows (requirements, analysis, design, implementation, test), and uses the unified modeling language (UML). In UML, a COTS component is represented as a component, a physical and replaceable part of the system that provides a set of interfaces and typically represents the physical packaging of classes, interfaces and collaborations [13]. The Model-Based Architecting and Software Engineering (MBASE) approach considers four types of models: success, process, product and property [2], and is consistent for use with COTS components [1]. 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 [15]. The PORE process model identifies four goals to be performed in a COTS selection process: acquiring information from the stakeholders, analyzing the information to determine if it is complete and correct, making the decision about product requirement compliance if the acquired information is sufficient and selecting one or more candidate COTS components. Research that considers the long-term maintenance impact of using COTS components [5,10] is also being investigated.

Section 2 provides a brief review of CARE whose focus is on defining requirements in the presence of COTS components. Section 3 presents ACASA, which extends CARE to define a more adaptable architecture (design- and run-time). Section 4 illustrates ACASA using a telepresence system as an example application. Contributions and future directions are summarized in Section 5.

2. A brief review of CARE

The COTS-Aware Requirements Engineering (CARE) approach is focused on creating and modeling a requirements engineering approach that explicitly supports the use of Commercial-Off-The-Shelf (COTS) components. The effective use of COTS components requires the identification and resolution of conflicting goals as well as bridging the gaps between stated requirements and 'approximately fitting' components while still satisfying the customer [9]. The CARE approach is characterized as agent-oriented, goal-oriented, knowledge-based and has a defined methodology or process.

Using the i* framework [20], CARE is agent-oriented. The agents are the stakeholders for the SUD. An agent is intentional and possesses intentional properties including goals, beliefs, abilities and commitments. An agent is also autonomous, makes decisions and choices and depends on other agents to accomplish goals, complete tasks or furnish resources. An agent can analyze its opportunities and risks in the various proposals and configurations for a system.

The agents' goals for the SUD are defined. Using the NFR Framework [9,14,15], CARE is softgoal-oriented, allowing for systematic tradeoff analysis, where functional alternatives can be explored and represented through OR dependencies. Once defined, the goals are used to drive the development of the system. First, the goals are refined into system requirements. In turn, system requirements may be refined into software, hardware or interface requirements.

The CARE knowledge base uses a meta-model that is organized into a World model, an Interface model and a System model [8]. 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 SUD 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.

Using the IDEF0 notation [11], CARE has a defined process. At a high level, the process has activities to define the agents, goals, system requirements and software requirements (with COTS). Each of the activities corresponds to developing a product artifact for the system.

CARE uses a knowledge base (repository) that is populated with descriptions of components (refer to Fig. 1). These descriptions are at two levels of abstraction: goals and product specifications. The goals provide high level descriptions of the functional and nonfunctional capabilities of the component. The product specifications provide detailed descriptions of the functional and nonfunctional capabilities in addition to the system requirements (e.g., memory requirements) for the component. The components in the repository are called "foreign" in the sense that they are (initially) unrelated to the SUD. In CARE, the goals and requirements in the repository are called foreign goals and foreign requirements. The goals and requirements of the SUD are called native goals, native system requirements, native software requirements, etc. When components are selected for use in the SUD, native goals are mapped to foreign goals and native requirements are mapped to foreign requirements.

3. An overview of ACASA

ACASA extends our CARE approach to define an adaptable architecture, i.e., a software architecture which can more easily accommodate changes in the stakeholder requirements while retaining earlier, or even expanding, use and reuse of COTS components [7].

In ACASA, architectures are represented in terms of their essential constituent concepts, including their components, connections, constraints, patterns and styles—consistent with other proposals (e.g., Ref. [16]) (refer to Fig. 1). COTS components are selected according to the *relevance* of their corresponding requirements (i.e., foreign requirements) to the native requirements.

In ACASA, the rationale behind matching and selection is documented and maintained. The current ACASA offers run-time adaptability by creating wrappers that choose among foreign architectures, or their constituents, whose corresponding requirements best match the native requirements. Considering the difficulties with searching, ACASA also maintains those foreign architectures whose corresponding requirements are believed to have the potential to match variations in the native requirements, for the purpose of the next round(s) of design-time adaptation.

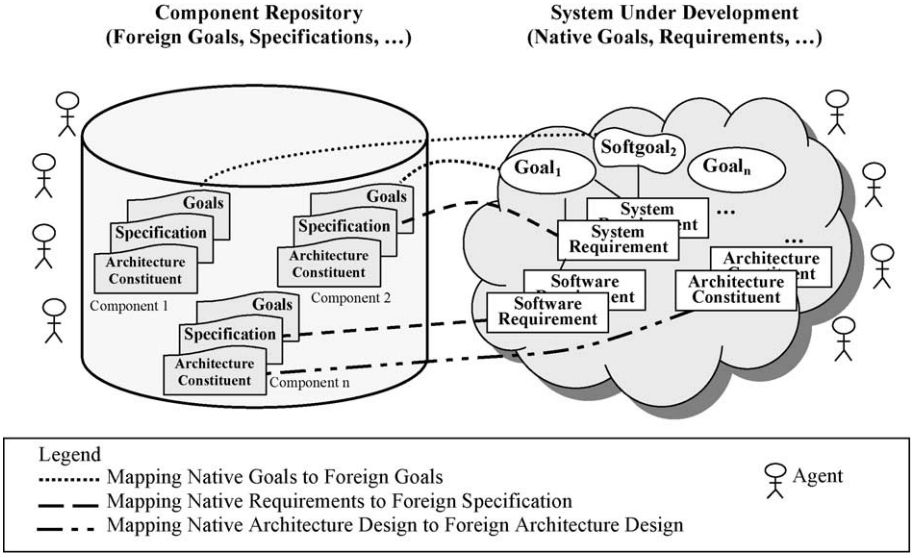


Fig. 1. CASA: mapping native goals, requirements and architecture components to foreign goals, specifications and architecture components.

Using ACASA, an architectural design process would proceed as follows.

(1) Postnative requirements (nR), consisting of adaptability and any other important nonfunctional requirements (NFRs) as well as functional requirements (FRs):

nR = nNFR + nFR, where
nNFR = {nNFR.1, nNFR.2, . . ., nNFR.l}
nFR = {nFR.1, nFR.2, . . ., nFR.k}.

(2) Consult (sets of) foreign requirements in the repository to refine the native requirements, nR, considering and comparing any particular characteristics of the native and foreign domains:

fR = {fR1, fR2, . . ., fRm}, where
fRi = fNFRi + fFRi
fNFRi = {fNFRi.1, fNFRi.2, . . ., fNFRi.p}
fFRi = {fFRi.1, fFRi.2, . . ., fFRi.q}

(3) Search for the foreign architectural designs (or their parts) whose requirements (or their parts) match the native requirements (or their parts), and consider them as candidates for the native architectural design in order to meet the native requirements stated:

{fAi | fRi matches nRj} or {fAi.s | fRi.s matches nRj} where matches is:

- excellent: $fRi \sim nRj$ *or* $fRi.s \sim nRj$
- good, but foreign requirements are bigger in scope: $fRi \gg nRj$ *or* $fRi.s \gg nRj$
- good, but foreign requirements are smaller in scope: $fRi \ll nRj$ *or* $fRi.s \ll nRj$
- overall similar, but different context or scope: $fRi \sim \sim \sim nRj$ *or* $fRi.s \sim \sim \sim nRj$
- poor: $fRi <> nRj$ *or* $fRi.s <> nRj$

(4) Analyze tradeoffs among the alternative architectures, in relation to nNFR, the native NFRs, while considering the impact of the various design decisions upon the requirements:

{fAi contributes-to nNFRj or
{fAi.s contributes-to nNFRj, where
contributes-to = {strong-positive, weak-positive, neutral, weak-negative, strong-negative, . . .}

(5) Select, as alternatives of the native architectural design, among the candidate architectures (or their parts) that best satisfies the native requirements in the context of the native application domain, and prioritize them into several different groups, such as:

excellent run-time adaptation alternatives:
{fAi | fRi $\sim$ *nRj} or {fAi.s | fRi.s* $\sim$ *nRj}*
good run-time adaptation alternatives:
{fAi | fRi $\ll$ *nRj} or {fAi.s | fRi.s* $\ll$ *nRj} or*
{fAi | fRi $\gg$ *nRj} or {fAi.s | fRi.s* $\gg$ *nRj}*
good development-time adaptation alternatives:
{fAi | fRi $\sim \sim \sim$ *nRj} or {fAi.s | fRi.s* $\sim \sim \sim$ *nRj}*

(6) Compose the selected architectural alternatives (or their parts) into the native architectural design (or its parts):

{case nRj@now of
fRi: fAi
fRi′ : fAi′
. . .
otherwise: consider {fAr} if the user is willing to try; time for design-time evolution, otherwise}

The steps in the entire process are interleaving and iterative. They are carried out in terms of a visual representation, called softgoal interdependency graph (SIG), where each node represents a softgoal and each link between a parent goal and its descendants represents the degree to which the descendants (positively or negatively) contribute to the satisfying of the parent softgoal. Accommodating informal, semiformal and formal representations, an SIG acts not only as a means to arrive at the end (an architectural design) but also as a development history, which can later be used to understand the rationale behind various design decisions and make improvements.

4. Illustration: architecting a telepresence system

Telepresence is defined as *the experience of presence in an environment by means of a communication medium* [18]. The telepresence system used as our

example is a business application that supports the remote interaction of colleagues. For example, Erin in New York can meet remotely with Eric, John and Sue in Dallas (refer to Fig. 2). The system allows Erin to see, move, talk and interact with every person at the remote site. Erin needs to have a head-tracking device, head mounted display, microphone, speakers, camera and force feedback devices (e.g., a joystick or gloves). The location where Eric, John and Sue attend need to have a microphone, camera, display and speakers.

The ACASA process is used to select an architectural design and the COTS components for a telepresence system. Part of an example is illustrated here using three native requirements and six foreign requirements that are related to audio and video signals.

4.1. Step 1. Native requirements

The native requirements posted in this example are identified as nR1, nR2 and nR3. These requirements, described below, are composed of functional and nonfunctional parts.

4.1.1. nR1

The system shall be able to encode/decode video in compliance with ITU-T H.263 1998-02 standard. The system shall achieve this goal while providing the highest quality of service possible (best resolution, least jitter).

4.1.2. nR2

The system shall be able to display to the user the image of the remote meeting site. The video image should be in color and be three-dimensional. The video shall be displayed in such a way that the viewpoint displayed changes in accordance with user's head movements (orientation). The system shall be able to accept SVGA input signal and at least one of either NTSC or PAL signal. The minimum resolution shall be of SVGA.

4.1.3. nR3

The remote system shall use pan/tilt/zoom cameras that supply an NTSC signal and a microphone to supply audio. The system shall be able to send synchronized audio and video to the head mounted display (with headphones) at the local site in real time.

4.2. Step 2. Foreign requirements (of components in repository)

The foreign requirements considered in this example are identified as fR1, fR2, ... fR6. A brief description of the component is provided followed by the requirements. The requirements are composed of functional parts and nonfunctional parts.

4.2.1. fR1 (component 1)

This software component for the Windows NT OS supports encoding/decoding all five picture formats in ITU-T H.263 1998-02 (Sub-QCIF, CIF, SQCIF, 4CIF, and 16CIF) and the optional performance enhancements Unrestricted Motion Vectors, Syntax-based arithmetic coding, Advance prediction, and P–B frames. The system shall be able to do so in real time

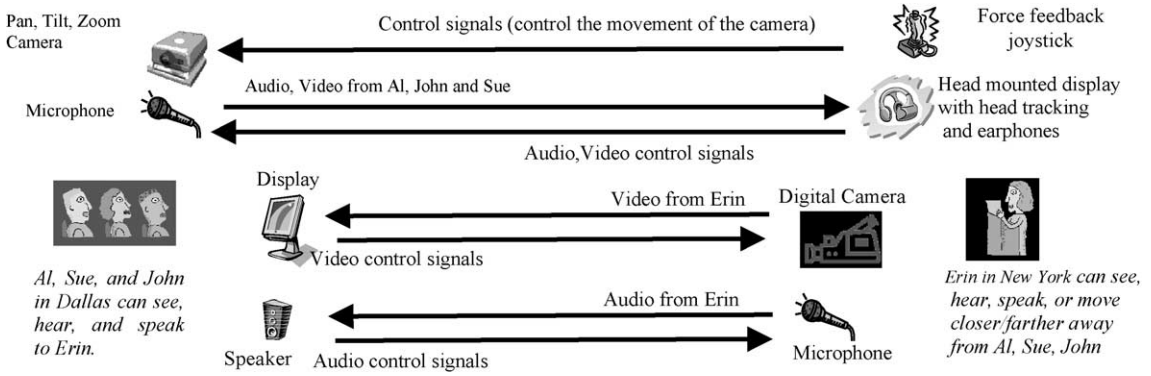


Fig. 2. Telepresence system.

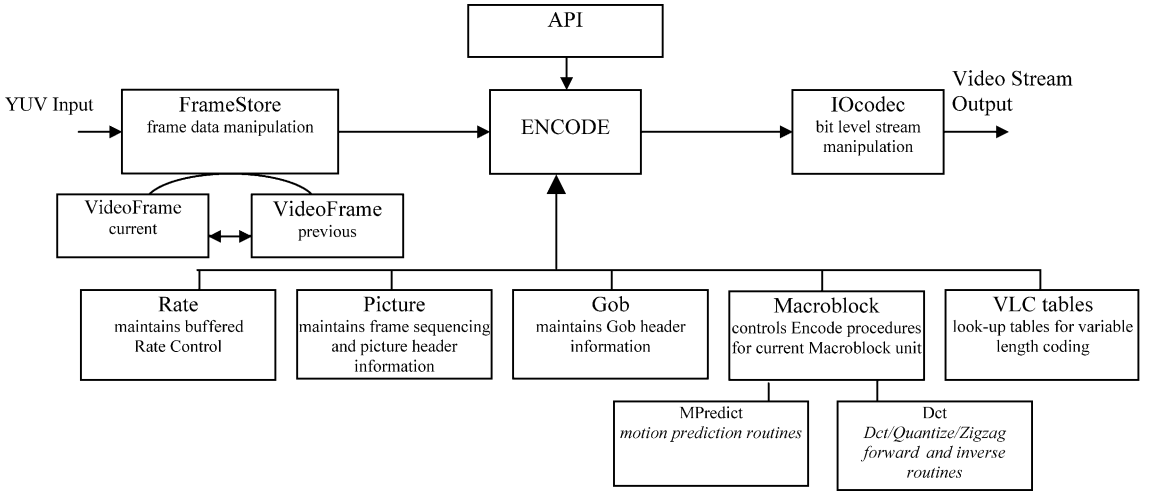


Fig. 3. H.263 architecture configured for encoding.

and maintaining high resolution. Refer to Fig. 3 for a possible architecture.

4.2.2. fR2 (component 2)

This software component for the Windows NT OS supports encoding/decoding the two picture formats in H.261 (CIF and SQCIF). The system shall be able to do so in real time and have a small memory footprint. Refer to Fig. 4 for a possible architecture.

4.2.3. fR3 (component 3)

The Canon VC-C4 system is an analogue pan, tilt, zoom camera. It includes a 16× zoom, brighter lens, center mounted for a smoother pan/tilt, a quieter, precision pan/tilt mechanism with no shaking and minimal vibration, intelligent image processing enabling the codec to achieve higher compression performance, field of view 3–47.5° (65° with wide-angle lens adapter), 460 TVL of resolution, height of 3.5″ and weight of 1 lb. The VC-C4 has five ports: S video, NTSC video and two RS232 ports (used to control the camera from a PC or daisy chain up to eight cameras). No architecture is available for this system.

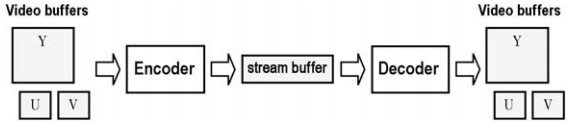


Fig. 4. H.261 codec architecture.

4.2.4. fR4 (component 4)

This head mounted display (HMD) takes NTSC or VGA, and automatically selects between NTSC and VGA depending on which is connected. It is 240 down by 320 across pixels: not full VGA resolution. It supports gray scale, 2D video. The system shall provide stereo audio. Refer to Fig. 5 for a possible architecture.

4.2.5. fR5 (component 5)

This head mounted display (HMD) supports color, 3D video in S-video format and stereo headphones and a head tracking capability. The display features

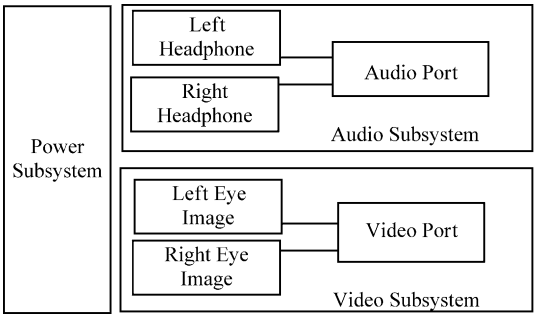


Fig. 5. 2D HMD architecture.

include SVGA resolution (800 × 600), field of view: 26° diagonal, image size: 76″ at 13′ and 24-bit color depth. The cable to connect i-glasses to S-video source is 19 feet long S-video connector. Built-in headphones use stereo RCA for Audio. Refer to Fig. 6 for a possible architecture.

4.2.6. fR6 (component 6)

QuickTime 5 by Apple Computer is a software component for the Macintosh, NT, linux and BSD OS that supports over 50 digital file formats. The input file formats include AIFF, AU, Cubic VR, DLS, DV, GIF, JPEG/JFIF, MIDI, MPEG-1 (Playback and Streaming), MP3 (MPEG-1, Layer 3), Virtual Reality (VR) and Wave. The video codecs include DV NTSC and PAL, H.261, H.263). The export file formats include AIFF, AU, DV stream, JPEG/JFIF, MIDI, TIFF and WAV. The sound compressors include 24-bit integer, 32-bit floating point, 32-bit integer, 64-bit floating point, ALaw 2:1 and AU. The video effects include animation, blur, edge detection and zoom. No architecture is available for this system.

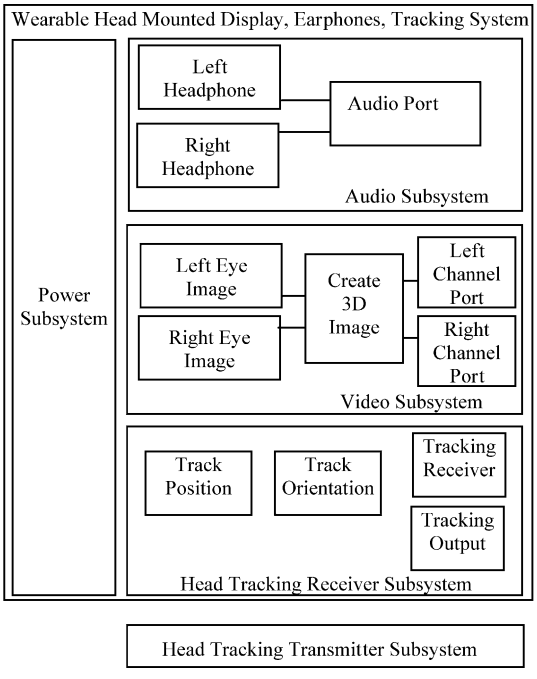


Fig. 6. 3D HMD architecture.

4.3. Steps 3, 4, 5. Search, match, select for each nR

The architect chooses the foreign architectures (i.e., components), whose requirements match the native requirements. The selected components are maintained either as run- or design-time alternatives.

4.3.1. nR1 ≪ fR1.1

Native requirement nR1 is a good match to part of foreign requirement fR1. The foreign requirements are bigger in scope. The foreign component achieves this goal in real time and maintains high resolution. The native system can switch to this component at run-time when memory resources are readily available.

4.3.2. nR1 ≫ fR2.1

Native requirement nR1 matches very closely to part of foreign requirement fR2. The scope of the foreign component is less than the native requirement. The component supports H.261 (a subset of H.263). It offers poorer resolution and reduced memory needs. The native system can switch to this component at run-time when memory resources are scarce.

4.3.3. nR2 ~ fR5

Native requirement nR2 matches very closely to the foreign requirements fR5. The foreign component displays color, 3-D images in SVGA, provides stereo audio, and a head tracking capability. With its requirement matching the native requirement well, this component becomes a run-time alternative (a design-time alternative, if it is not feasible to support switching HMD systems).

4.3.4. nR2<>fR4

Native requirement nR2 does not match well with fR4. The foreign component only supports 2D gray scale display in sub-VGA resolution and does not provide a head tracking capability.

4.3.5. nR3 ≫ fR3

Part of native requirement nR3 is a good match to foreign component fR3. The pan, zoom, and tilt camera requirement is an excellent match. The parts of the native requirements for the microphone and the delivery of synchronized audio and video to the head mounted display are not addressed.

In summary, the relationships the architect identifies up to this point in the example between the native and foreign requirements are: nR1 ≪ fR1.1, nR1 ≫ fR2.1, nR2 ~ fR5, nR2 <> fR4, nR3 ≫ fR3.

4.4. Step 6. Compose the architectures

The architectural alternatives (or their parts) are composed into the native architectural design (or its parts). Each native requirement is represented as a case statement:

```
{case nR1@now of

   fR1.1: fA1.1
   fR2.1: fA2.1
   Otherwise time for design-time evolution}
{case nR2@now of

   fR5: fA5
   otherwise: time for design-time evolution}
{case nR3@now of

   fR3: fA3
   . . ./* other foreign components to map to native
   requirements for a microphone and the syn-
   chronization of audio and video*/
   otherwise: time for design-time evolution}
```

5. Conclusions

This paper has presented preliminary work on Adaptable COTS-Aware Software Architecting (ACASA), a systematic methodology for using COTS components for developing adaptable software architectures. Adaptability in the architecture is considered at design- and run-time. In the current ACASA, adaptability of the architecture is satisfied by changing the use of one foreign architecture (or its parts) to another (or its parts). The methodology is clearly requirement-driven, in that the basis for selecting foreign architectures is how closely their requirements match the native requirements.

Several lines of future work include the investigation of the relevance relation—defining the equivalence classes of excellent, good, similar and poor. More matching techniques are also needed, for both design-time component selection and run-time comparison of the native requirements against the foreign requirements whose architectures act as basis for run-time adaptation. Another line of future research concerns exploration of other adaptability enhancing techniques, including meta-level monitoring and control of requirements changes and corresponding selection of appropriate architectural parts. Consideration of other domains is also needed so as to determine the weaknesses and strengths of the ACASA approach. Additionally, tool support is needed for facilitating the construction of a knowledge base with finer-grained architectures and their components.

References

[1] B. Boehm, Requirements that handle IKIWISI, COTS, and rapid change, IEEE Computer 33 (7) (July 2000) 99–102.

[2] B. Boehm, D. Port, M. Abi-Antoun, A. Egyed, Guidelines for the Life Cycle Objectives (LCO) and the Life Cycle Architecture (LCA) deliverables for Model-Based Architecting and Software Engineering (MBASE), TR USC-CSE-98-519, USC-Center for Software Engineering.

[5] D. Carney, S.A. Hissam, D. Plakosh, Complex COTS-based software systems: practical steps for their maintenance, Journal of Software Maintenance: Research and Practice 12 (6) (Nov.–Dec. 2000) 357–376.

[6] L. Chung, K. Cooper, Towards a model-based COTS-Aware Requirements Engineering process, MBRE '01, Nov. 2001, 52–63.

[7] L. Chung, K. Cooper, S. Lee, F. Shafique, A. Yi, Towards Adaptable COTS-Aware Software Architecting, SERP'02, CSREA Press, Las Vegas, Nevada, June 24–27, 2002, pp. 38–43.

[8] L. Chung, K. Cooper, A knowledge-based COTS-Aware Requirements Engineering approach, Proceedings of SEKE, 2002, 175–182.

[9] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, Boston, MA, 2000.

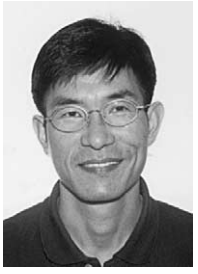
[10] J.C. Dean, Ensuring the capability of COTS products, 23rd Annual International Computer Software and Applications Conference, IEEE Computer Society Press, 1999, pp. 96–97.

[11] Federal Information Processing Standard (FIPS) Publication 183, Integration Definition for Function Modelling (IDEF0), December 21, 1993.

[12] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison Wesley Longman, USA, 1999.

[13] P. Kruchten, Modeling component systems with the unified modeling language, International Workshop on Component-Based Software Engineering, April 25–26, 1998, Kyoto, Japan, Electronic Publication.

[14] J. Mylopoulos, L. Chung, S.S.Y. Liao, H. Wang, E. Yu, Extending object-oriented analysis to explore alternatives, IEEE Software, (Jan./Feb. 2001) 2–6.
[15] J. Mylopoulos, L. Chung, B. Nixon, Representing and using nonfunctional requirements: a process-oriented approach, IEEE Transactions on Software Engineering 18 (6) (June 1992) 483–497.
[16] C. Ncube, N. Maiden, Guiding parallel requirements acquisition and COTS software selection, Proceedings of the IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, 1999, pp. 133–140.
[18] J. Steur, Defining virtual reality: dimensions determining telepresence, Journal of Communications 42 (4) (1992) 73–93.
[20] E. Yu, Modelling strategic relationships for process reengineering, DKBS-TR-94-6, The University of Toronto, Canada, Dec. 1994.



Dr. Lawrence Chung is currently an Associate Professor in the Department of Computer Science at the University of Texas as Dallas. He received his PhD from the University of Toronto, and is currently on the Editorial board of the Requirements Engineering Journal. He is the principal author of the book Non-Functional Requirements in Software Engineering, which is being adopted in extending object-oriented analysis to goal- and agent-oriented analysis. His research interests include Requirements Engineering, Software Architecture, Electronic Business Systems and Conceptual Modeling.

Dr. Cooper is an Assistant Professor in Computer Science at the University of Texas at Dallas. She holds a BASc, an MASc and a PhD, all in Electrical and Computer Engineering from The University of British Columbia, Canada. Her research interests include the use of COTS components, metrics, empirical analysis and formal methods in requirements engineering and software architecture. Dr. Cooper also has industrial experience in the requirements specification and architectural design of complex, large-scale, software-intensive systems. Before joining UTD, she worked as a senior systems engineer on Motorola's GPRS core network project developing system requirements and extending the product architecture.

Stephen Kinwai Lee was born in Hong Kong. He received his BBA and MBA degrees from the University of Louisiana at Monroe in 1991 and 1993, respectively. He is currently pursuing his MS degree in Computer Science at the University of Texas at Dallas. His prior experience includes financial auditing of various governmental entities and telecommunication companies. He was also involved as an analyst in the software development process.

Faisal Shafique is currently studying Masters in Computer Science (Software Engineering) at the University of Texas at Dallas. Faisal Shafique did Bachelors in Mechanical Engineering from University of Engineering and Technology at Taxila, Pakistan. His research interests include Component-Based Software Engineering, CAD/CAM and Software Architectures.

Ms. Anna Yi is currently a PhD student in the Department of Computer Science at the University of Texas at Dallas. She received her BA degree from the University of Texas at Austin. Her experience includes taking a webmaster position for a professional organization and participating in a distance learning course development. Her research interests include model-driven dynamic GUI development, business and design patterns and nonfunctional requirements.