Author
Heikki Kontio
heikki.kontio@helsinki.fi

Current trends in software industry: COTS Integration

Table of Contents

Terms and abbreviations

ADL	Architecture Description Language
AIS	Archware-based information systems
ANSI	American National Standards Institute
API	Application Programming Interface
CBSD	Component-Based Software Development
COTS	Commercial-off-the-shelf
E2E	End-to-End
HTTP	Hypertext Transfer Protocol
ISO	International Standards Organization
MMS	Multimedia Messaging Service
ORB	Object Request Broker
SOAP	Simple Object Access Protocol
WSI	Web services infrastructure
WWW	World Wide Web

1 Introduction

The purpose of this document is to have a brief look at the COTS (Commercial-Off-The-Shelf) concept and its status in the current technology environment. We will also be looking at the ArchWare architecture approach to COTS integration. The author of this text will also give some personal insights into the COTS concept based on own experiences during the last few years.

2 COTS: The Concept

A commercial off–the-shelf (COTS) item is one that is sold, leased, or licensed to the general public. It is offered by a vendor trying to profit from it, supported and evolved by the vendor who retains the intellectual property rights, available in multiple, identical copies and finally, used without modification of the internals. [04]

Following are some examples of COTS-based components.

- Northrop Grumman B-2 Spirit: a multi-role stealth bomber, capable of delivering both conventional and nuclear munitions. The basis for the acquisition of B-2 was U.S. Air Force's requirement to acquire a strategic bomber capable of stealth [05].
- Nokia MMS Center: a server-side software platform, which enables mobile phone users to send multimedia messages (MMS) to each other. Can be bought as a "black box" and integrated into mobile phone operator's existing production environment.
- Standard car battery: there are several vendors across the globe who offer standardized car batteries, which can be installed to cars without any major effort.
- Deep Space Network Program at the NASA Jet Propulsion Laboratory [01]
- Lewis Mission at NASA's Goddard Space Center [01]
- Boeing's new 777 aircraft with 4 million lines of COTS software (!) [01]
- Air Force Space and Missile System Center's telemetry, tracking, and control (TT&C) system called the Center for Research Support (CERES) [01]

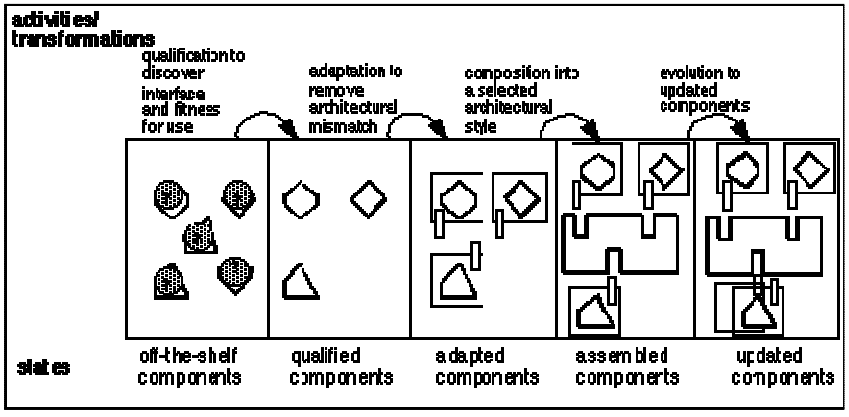
The main point here is that a COTS component doesn't necessary have to be a software component; it can be everything from rubber boots to fighter planes. The concept still stays the same.

3 Integrating COTS components

Basically acquiring COTS components should ease up the development process of a new IT service quite a lot, because the development team doesn't have to "reinvent the wheel" for some particular feature anymore. However, it is really rare that a COTS component can be just installed straight away to the existing production environment. Many times COTS components make several assumptions about architectural issues. When these assumptions conflict or don't match, the simplicity benefits of a COTS component soon vanishes, as a requirement arises to start integrating the component to the environment with "glue" – customized software, that is. The root of the problem is many times the lack of source code and access to the developers. [06].

Indeed, in an IT service, it is really rare to find pure COTS architecture from the starting point to the end (E2E, end-to-end). In such cases the whole structure is more usually than not implemented by a single vendor, using proprietary techniques which are straight away compatible with each other. As this is mostly not the case in the real world, purchasing and customizing COTS components has become an integral part of the entire software development life cycle. [06]

Below is a picture describing the process how to acquire COTS based components.



Picture 1. Activities of the Component-Based Development Approach. [01]

The first phase is component qualification. It is a process of determining "fitness-for-use" of previously-developed components, which are applied to a new system context [01]. At the same time all possible candidates for the component can be checked out in the marketplace. Qualification can also mean to check out the development and maintenance processes for the software, which are extremely important in safety-critical applications, such as life support systems.

Two phases belong to the component qualification: discovery and evaluation. In the discovery phase the properties for the component at hand are reviewed. These properties include such things as functionality, interfaces, use of standards, component reliability etc. Many times it's also useful to include other aspects - such as the vendor's financial situation or market share. If, for example, an improved version of the existing software is required but the vendor is on the verge of collapse, then it is reasonable to have a look at other alternatives as well.

The evaluation phase is when a selection is made from a peer group of products. The International Standards Organization (ISO) describes general criteria for product evaluation, while others describe techniques, which take into account the needs of more specific application domains.

Component adaptation is the next phase. Because individual components are written to meet different requirements, and they are based on differing assumptions about their context, they must often be adapted when used in a new system. Components should be adapted based on rules that ensure conflicts among components are minimized; however, conflicts usually are unavoidable and will cause some teething problems when the components have been put into production. The degree to which a component's internal structure is accessible suggests different approaches to adaptation:

- white box: access to the source code is allowed. This allows high-level adaptation to every need.
- grey box: no source code access, but the component provides developers with APIs making it possible to attach the component to other systems dynamically
- black box: no source code, no API's no nothing (double negative written on purpose)! Just the executable binary code.

It should be noted here that if a component is offered and integrated as a white box, the in-house made developments might become a maintenance burden in the long term.

Assembling components into systems: the components must be integrated through some well-defined infrastructure. This infrastructure provides the binding that forms a system from the disparate components. For example, in developing systems from COTS components, several architectural styles are possible:

- database, in which centralized control of all operational data is the key to all information sharing among components in the system
- message bus, in which components have separate data stores coordinated through messages announcing changes among components
- object request broker (ORB) mediated, in which the ORB technology provides mechanisms for language-independent interface definition and object location and activation. [01]
- no centralized architecture for information exchange at all: information exchange is implemented project-specifically for each purpose

System evolution: at first, component-based systems may seem relatively easy to upgrade since components are the unit of change. To repair an error, an updated component is swapped for its defective equivalent, treating components as plug-replaceable units. Similarly, when additional functionality is required, it is embodied in a new component that is added to the system. However, this is a way too optimistic view of system evolution. Replacement of one component with another is often a time-consuming and arduous task since the new component will never be identical to its predecessor and must be thoroughly tested, first in isolation and then in combination with the rest of the service components. Wrappers must often be checked and rewritten, and side-effects from changes must be found and assessed. [01]

4 ArchWare: An Active-Architecture Approach to COTS Integration

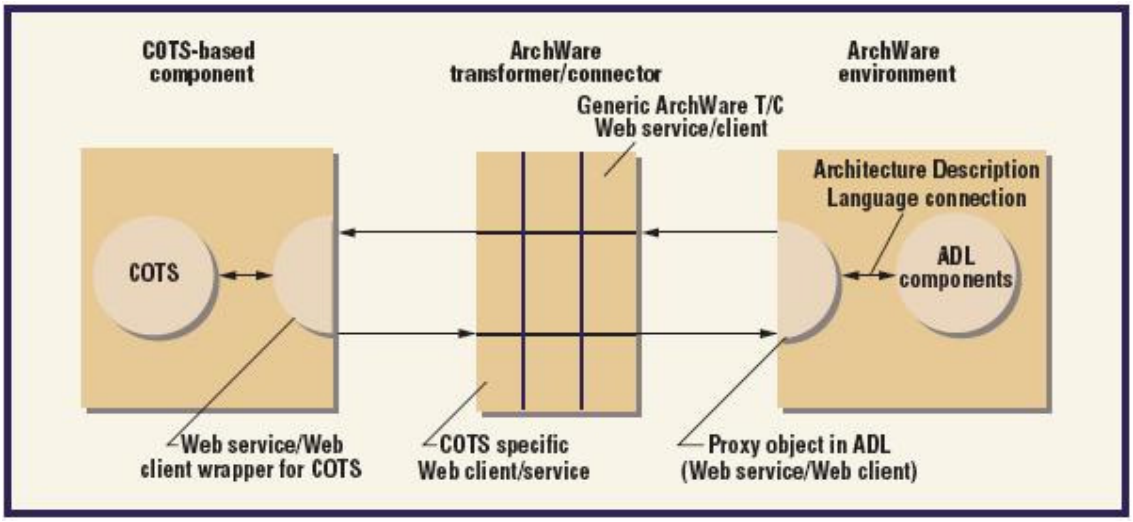
The Archware framework was developed by Brian Warboys, Bob Snowdon, R. Mark Greenwood, Wykeen Seet and Ian Robertson from the University of Manchester with Ron Morrison, Dharini Balasubramaniam, Graham Kirby and Kath Mickan from the University of St. Andrews. It supports COTS-based software systems, which can then better respond to both predicted and emergent changes arising from the use of COTS products. It is a process-centered approach to implement the core of the system as a network of evolvable, cooperating processes [07]. The active architecture tries to 1) capture each component's role within the IT service and 2) identify how the service can incorporate new COTS components or replace existing ones as the process network evolves.

The ArchWare framework makes it possible to integrate COTS components into adaptable distributed software systems, so called ArchWare-based information systems (AIS). AIS has 4 basic elements:

1. COTS components
2. ArchWare transformer/connectors (T/Cs): they provide the "wrappers", which capture the COTS component's role in the system.
3. ArchWare Architecture Description Language (ADL) components, which form the architecture model describing the COTS integration.
4. users!

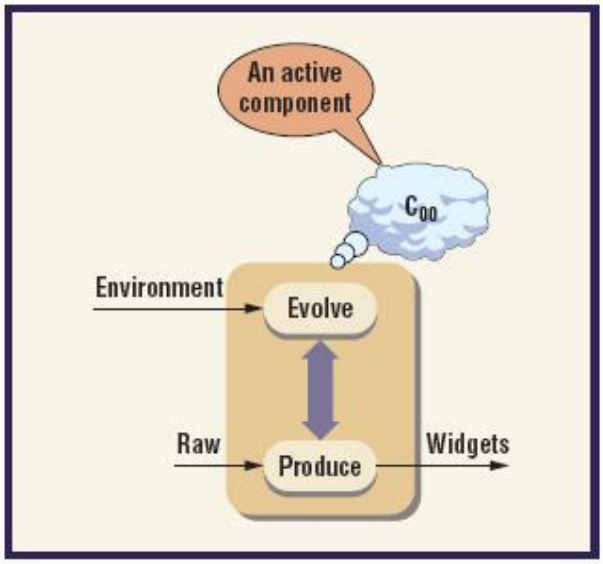
The ArchWare framework's architecture model (or to put it more simply, the way of work) is to create an interface layer, which is connected from one side to the COTS component and from the other side to AIS system (or whatever system entirety, that is). The model is active: it maintains its state (read: its interface syntax) when business logic on the top of it changes, and can also control and guide changes in the business logic. The ArchWare framework doesn't place any constraints on the COTS components, so its users can use whatever component they choose to.

In the center of the framework is the transformer/connector. Basically what is does is to work as a filter between the COTS component and the ADL component (see figure 2). It is a piece of software, which is e.g. able to monitor the COTS component regarding its proper functioning and it is also used to transfer messages (commands, queries etc.) between the components which it connects.



Picture 2. Integrating COTS and ADL components with a transformer/connector [07].

Users can interact with the system in two ways. They can use the COTS component directly, or they can interact with the ADL component. The ArchWare environment provides the users with an interface to the T/Cs. The interface technology is using Web services infrastructure (WSI). It is used to implement the interactions between COTS components and T/Cs and between T/Cs and ArchWare environments. It should be noted that interfaces, which eventually lead to COTS components, don't have to offer every functionality of the component. They are rather a chosen subset of functionalities and can also add some extra value to the interface feedback (such as subscriber information, terminal information etc.).



Picture 3. An ArchWare component's basic structure.

Basically ArchWare components are process components. In the picture above are described two main sections: Produce and Evolve.

- Produce (P) represents the component's production or operational behavior—that is, the behavior that fulfills the component's purpose. For example, if an Arch-Ware component is supposed to transform input "raw" to output "widgets," P fulfills this transformation.
- Evolve (E) represents the component's management behavior and is responsible for ensuring P's effectiveness in circumstances requiring change. So, E affects P.

Basically what this is, is a process description and is highly complicated to implement in pure source code, as organizations, architectural directions etc. change all the time. So this model is actually more efficient in the social than purely technical field [07].

5 Personal Experiences

- Use of ArchWare-like software engineering for many years
- We're using lots of COTS-based software and the trend is growing
- We provide Web Services –based interfaces for internal component use; the services feature a selected subset of the COTS components' functionalities
- We also add value from other systems to the interface feedback, so we can offer better service to the components. Added value is e.g. mobile subscription information, mobile terminal identification etc.
- The interfaces work as a glue, which enables us to integrate many vendor-independent components to each other, thus creating the final service for end customers
- The concept works well, but it is really tough to maintain clear architectural lines. Reasons vary.
- Sometimes vendor evaluations are made very well by the book, sometimes they are purely political decisions causing great deal of headache to engineers
- The cheapest solution is not always the best – neither is the most expensive so it requires experience and careful judgments to find a suitable solution. Many times this is very challenging due to time and budget constraints.

6 Summary

The usage of COTS components has risen all the time during the last decade and it's still growing. At the same time they have been matured in such way that they offer the possibility to better customization and integration to customers' specific needs. However, even if there's no need to reinvent the wheel, the demand for customer-specific versions and "glue" which attaches the COTS component to already existing environment is rising. The knowledge how to integrate COTS components into complex infrastructures will be of great importance in the future.

7 References

[01] Carnegie Mellon Software Engineering Institute: Component-Based Software Development

[02] Southwest Research Institute (SwRI)

[03] Ernagan, Farcas, Farcas, Krüger, Menarini: A Service-Oriented Blueprint for COTS Integration: the Hidden Part of the Iceberg. IWICSS'07

[04] U.S. Department of Defense: Commercial Item Acquisition: Considerations and Lessons Learned, June 26 2000

[05] FAS.org: B-2 Advanced Technology Bomber (http://www.fas.org/man/docs/peds_98/0604240f.htm)

[06] Egyed, Müller & Perry: Integrating COTS into the Development Process. IEEE Software, July/August 2005

[07] Warboys, Snowdon, Greenwood, Seet, Robertson et. al: An Active-Architecture Approach to COTS Integration. IEEE Software, July/August 2005