Final version published in Requirements Engineering Journal (2005)
DOI 10.1007/s00766-005-0022-5
www.springerlink.com

COTS Tenders and Integration Requirements

Søren Lauesen, slauesen@itu.dk
IT University of Copenhagen, Rued Langgaardsvej 7, DK-2300 Copenhagen S

Abstract. When buying COTS-based software, the customer has to choose between what is available. The supplier may add some minor parts, but rarely everything the customer wants. This means that the customer cannot write down his requirements and expect that they can all be met. A scoring system is necessary rather than traditional mandatory requirements. Requirements for integrating the new COTS system with other systems are particularly hard because suppliers may integrate in different ways and with different other systems. A related problem is that once the new COTS system is purchased, the COTS supplier may have a de-facto monopoly. Only he can expand the system or integrate it with other systems.

The traditional way to purchase COTS is to iteratively find the right product. However, in a tender process this is not possible, and another solution is necessary.

Experience shows that customers fail to deal with these issues adequately. As an example they may believe that asking for open interfaces is sufficient to guard them against monopoly. In this paper we analyze the problems and show ways to deal with them. We illustrate the problems and solutions with real-life examples from Electronic Patient Recording systems (EPR).

Keywords: COTS, tender process, integration requirements, risk reduction, open-target requirements, supplier monopoly, third-party requirements.

1. Background

Software is increasingly bought as COTS - Commercial Off The Shelf. COTS products range from simple components (e.g. sort procedures or user interface components), through complex operating systems and middleware (e.g. CORBA-based), to tool packages (e.g. MS Office) and complex application packages (e.g. ERP systems such as SAP or BAAN).

The COTS term should not be taken too literally. In practice, the COTS supplier may not only deliver the software off the shelf, but also extend it in customer-specific ways, for instance integrate it with other systems. He may also offer consultancy, data conversion, user training and support. Morisio & Torchiano (2002) give a comprehensive taxonomy of COTS products.

The ability for two products to cooperate is called *interoperability*, and requirements dealing with it are called interoperability requirements. When two products cooperate closely, we say that they are *integrated*. There are many degrees of integration as we will see below.

In this paper we will look at cases where the supplier delivers one or more COTS application packages, typically including middleware. He adds wrappers, adapters and glue code to integrate his products with the customer's legacy systems. He may be willing to extend his COTS products in ways that meet specific customer needs, if they also seem suitable for future customers.

COTS products may be acquired in many ways. Here are some examples:

Components for in-house development

The customer's IT-organization looks around for suitable COTS components, buys them and embeds them in their own systems. Semancik & Conger (2002) give an example where the customer (NASA) built an autonomous file and distribution system. Their approach was to gather information about potential components from the Internet, local experts, and peers. They got trial versions from the vendors and tried out the products. Gradually they developed selection criteria and narrowed down their search to a few vendors.

Components for complex products

The customer is a product developer. He integrates the COTS components into his own product. Helokunnas and Nyby (2002) give an example where the customer (Nokia) finds suitable suppliers, negotiates detailed interface requirements with them, and establishes long-term partnerships.

Tools packages

The product is truly off-the-shelf and cannot be modified. Being a tool, it is not embedded in other systems but may be loosely connected with them. Maiden & Ncube (1998) report on the lessons learned in selecting a software tool. A key lesson was that selection criteria were an exploratory exercise - you cannot define the requirements based solely on perceived customer needs.

Business applications, company customer

The customer is a large company who wants a business application. Examples are ERP systems and banking systems. The customer will usually ask the supplier to do the necessary integration with his legacy systems. The acquisition approach may be similar to the approaches above: The customer gradually develops selection criteria, tries out the systems, and narrows down the search. He may negotiate conditions and long-term relationships with the supplier.

COTS tender for public organizations

The customer is a public organization who wants an application such as a health care system or a payroll system. In the EU, public acquisitions have to follow strict rules to protect against corruption. All suppliers must have equal opportunities. For contracts above a certain amount (around 150,000$), this is achieved through a tender process where the customer specifies the requirements and the selection criteria. The suppliers submit their proposals. The customer chooses between the proposals - based on the selection criteria. Prices, requirements and other conditions cannot be negotiated once the tender process has started.

This situation is much more difficult because you cannot develop selection criteria gradually. In principle, you could to try out the systems prior to the tender process, but the systems are so complex that it is not feasible.

In this paper we look at solutions for the tender situation. In particular, we look at requirements for integrating the new system with legacy systems and systems that the customer may acquire later. Although the solutions were developed for the public tender situation, they are also useful for other kinds of COTS acquisition.

2. Research in the area

Much research has been done about COTS in general, particularly about the vendor side - how COTS products are developed. Much less has been done about COTS software acquisition in practice. Brownsword et al. (2000) outline a process for COTS software acquisition. Albert & Brownsword (2002) explain that COTS solutions comprise much more than the COTS product itself. Sai (2003) reports about acquisition of a budget system to integrate with the existing accounting system. They all stress the iterative narrow-down approach (risk-based) and suggest that it can be handled in a semi-structured way (e.g. RUP-based). However, the tender situations we discuss don't allow iteration of this kind.

Until recently little research was done about the customer's attempts to integrate COTS products. Bansler & Havn (1994) and Boehm & Abts (1999) report on some of the difficulties and lessons from COTS integration in practice. Boehm & Abts give the important advice of early inspection of the COTS products and their integration abilities, rather than trusting the supplier's claims (a risk-driven approach). They also advice to choose open architectures to better deal with future products. Balk & Kedia (2000) discuss a specific project with integrating nine COTS products. The integration was done by the customer's IT staff, and all the COTS products had already been used by the customer as stand-alone systems. Uncertainties about their functionality were thus reduced, and the requirements were not contractually strict. The integration required around 3,700 work hours. Hornstein & Willoughby (2002) report about acquisition problems in NASA and how history had created artificial distinctions of component types. Lewis and Wrage (2004) report on a project that integrated several COTS products using XML. They give wonderful examples of why XML - as an open standard - only solves a small part of the integration problems. These publications are useful background information, but don't help us express the requirements.

Gorton & Liu (2002) discuss acquisition of COTS middleware and warn of the vague vendor specifications of the products, and the lack of compatibility in spite of "open interfaces". They explain in detail about their narrow-down approach when helping clients to select a middleware product. Their approach is inspiring, but since we look at application acquisition, the middle-ware requirements are not directly useful.

Interoperability in general is covered well, for instance with the many works on open source and various schemes and protocols for integration and reuse of software modules. Guo (2000) and Wileden & Kaplan (1999) give good overviews of the technological solutions. Bao & Horowitz (1996) and Saur et al. (2000) show two typical approaches to integrating largely incompatible products. Yakimovich et al. (1999) made a classification of the integration technologies and their compatibility, and applied it to typical COTS products. Davis et al. (2003) give a taxonomy of interoperability conflicts. Although these publications don't mention requirements, they have been a source of inspiration for the solutions below.

The literature shows the highly technical issues involved. The customer in the public tender situation doesn't want to deal with them. He wants the supplier and third parties to take care of them, and the requirements must ensure this.

3. Research method

The basis for the present work was studies over a couple of years of five COTS tenders. The first was a business application (ERP system) for a shipyard, the next two were hospital systems for payroll and roster planning, the next a hospital system for patient administration, and the last one was a meter reading and billing system for a municipality that supplied power, water, etc.

The main focus in these studies was how to deal with requirements to the user interface. At that time this was the most problematic area. As part of the first hospital studies, we developed the Task & Support approach for requirements. The idea was to describe the user tasks without going into detail with who does what (similar to use cases, but without being explicit about the actors). The requirement is then to support these user tasks. The supplier describes for each task and subtask how he will support the user. The customer compares these descriptions, rates the solutions, and selects the winner.

The approach was first validated by expert judgement with one of the hospitals and with three suppliers. More than a year later, the hospital used the approach in a tender for patient administration. They reported excellent results and a reduction of tender costs to around one fifth. The approach and the results have been published elsewhere (Lauesen, 2002, 2003).

The last of the five studies (power and water supply) was intended to validate the Task and Support approach in another context. This project was not quite as successful for many reasons, for instance an unskilled use of the Task and Support approach, and excessive user involvement (Lauesen & Vium, 2004, 2005). However, it became apparent that the most problematic area had now shifted to integration requirements. In retrospect and when talking to suppliers, it was obvious that integration requirements had always been problematic, but now a much larger part of the project was about integration.

At first I had great difficulties coming up with reasonable integration requirements for a COTS tender. For some time I thought it was impossible. The Task and Support approach allowed us to compare user interfaces according to how well they supported the user tasks. Since most of the user interface exists, we can make the comparison before signing the contract. However, the approach was not suited for comparing integration solutions, because they show up very indirectly in the user interface, for instance through response times and data actuality. These aspects are not possible to inspect before the contract is signed. However, in cooperation with the customer, one of the suppliers, and my colleague J. P. Vium, I gradually developed the first outline of the solution presented in section 5. It was not really validated with experts at this point in time.

At the end of the study, I was fortunate to be involved as a consultant in two tender processes for EPR (Electronic Patient Recording). Here, the integration requirements were an even larger issue than with the power and water system. I had to elaborate the solution, and got some feedback from the hospital teams. They said that it looked okay. However, they were not really involved in a constructive way. Their focus was much more on the user interface aspect, user involvement (excessive), and the national EPR strategy. One of the hospitals had already purchased an integration platform and wanted to pursue the integration issue on their own. I continued the cooperation with the other hospital, however.

Since the experience from the power and water supplier was that integration *was* a critical issue, I visited the potential EPR suppliers (five of them) and talked to the expert groups that most likely would be involved in the tender. I showed them the outline of the integration requirements and asked whether it would be possible for them to reply to these requirements, whether the requirements seemed fair, and whether they allowed them to describe the advantages of their own solution.

They had several comments that caused some additional changes to the integration requirements. This is discussed further in sections 5.6 and 5.7. The hospital just took the integration requirements and patched them into a lot of other requirements that their IT department had made. And off the requirements went to the official bidding! I had hoped to study the results half a year later, but the project ran into many problems related to other issues. For instance there were legal complaints about not using the government standard contract, later the parties agreed to move the entire system to a different platform, etc. As a result, 15 months later the system is not yet operational and the parties keep matters top secret.

4. Integration problems in COTS tenders

The customer wants to integrate the new COTS product with several systems that he has already, and later with systems not known yet. In this section we will see why it is hard to specify requirements that can deal with this. We don't pretend to cover all the problems in the area, but only those we have experienced as serious. The problem list could probably grow forever.

We will illustrate the problems with real-life examples from acquisitions of EPR systems (Electronic Patient Recording systems). An EPR system records data about patients, their diseases (diagnoses), the treatments (patient services), and notes about why things are done and what the plans are. It must be possible to transfer data electronically to other health organizations or to statistical databases. Traditional patient records are largely text. Structured data such as measurements and diagnoses are often embedded in the text, and for this reason hard to use for overviews of the patient's state or for statistics. A good EPR system must help the clinical staff record all data in a more structured way, and be able to show the data in many ways.

One of the big issues is that there are thousands of different patient services. They include laboratory tests, medicine prescriptions, preparation for surgery, X-ray pictures, CT scanning, food for the patient, psychological services, and so on. Each type of service may have its own data format. From a computer science point-of-view we are dealing with a class, *service*, that has thousands of subclasses.

Furthermore, many services are handled by separate sub-systems (*production systems*) more or less automatically. As an example, some laboratory tests are fully automated. Batches of sample glasses with bar codes are inserted in the analyzer, and a bit later the test results are available in the production system's database. A good EPR system must integrate its own database and functionality with a score of specialized production systems. These systems don't even have a common communication protocol.

4.1. Difficult to explore existing products

Large COTS applications are too complex to try out in an exploratory manner. Typically the supplier would spend weeks setting up the system, and the customer cannot do it himself. The customer might work closely together with the potential suppliers one by one to study their products, but this too is time consuming. There is also a risk that the customer bases his requirements on one particular system that he studied closely. This will unintentionally exclude other suppliers. Although the customer is aware of this problem, he cannot figure out how to phrase his requirements in a supplier-independent way, and in the public area it easily leads to accusations of not treating the suppliers equally.

The customer may study how successful the products are in other organizations, and many customers actually do this. However, the products develop so fast that this may give a false picture of what is available now. The result is that the exploratory narrow-down approach suggested in literature doesn't work well in the tender case. Even if it did, the customer would still have to go through a tender process and justify the final selection of the winner through compliance with requirements and selection criteria.

4.2. Mandatory requirements don't work

To run a tender process, the customer must elicit requirements in the traditional fashion. However, he cannot express the requirements in the traditional hard way, such as

R1: The EPR system shall not store the results in its own database, but retrieve them from Labsystem X when needed.

I have often seen such requirements in practice. The problem is that this may be feasible for some EPR products, but for other EPR products it may either be impossible or cause very long response times. Such a requirement would unintentionally exclude a lot of suppliers.

Some analysts rightly say that R1 is a solution - not a requirement. To find the true requirement, we can ask the customer why he wants this solution. We might then replace R1 with this requirement:

R2: The EPR system must ensure that there is no difference between the shown results and the latest data in Labsystem X.

This requirement is too strict however, and it unnecessarily limits the possible solutions. In general it is impossible to find a supplier who can meet all the requirements that the customer dreams up. The customer wants to select the supplier that is closest to his wishes (see also the discussions in Sai 2003, and Perrone, 2004). The solution is to use *open-target requirements*, where the customer specifies his demands and expectations, and the supplier specifies how he can meet the demands. We explain more on this in section 5.1.

4.3. Product scope is fuzzy

The customer may not be sure how large a system he wants. Should it include middleware or should it use the platform that the customer has already? In the EPR case, should it include an X-ray system or should he get one from a third party? The best choice varies from one COTS system to another. Somehow the requirements must allow the suppliers to offer these things or integrate with a third-party product.

Some customers believe they can cut the entire system into pieces and purchase the pieces one by one. As an example, one hospital first purchased a middleware system. Next they wanted to purchase a series of smaller systems one by one that had to run on this middleware, for instance an X-ray system and a medicine system. Finally they planned to purchase the most important system - the EPR system - which would bind the other systems together from a user perspective.

Tempting as this may seem, experience shows that applications and middleware are so closely related that a system cannot easily be ported from one middleware system to another - even if they all claim to follow an open standard such as CORBA and J2EE (Gorton & Liu, 2002; Liu & Gorton, 2003). As a result, the customer may soon end up in a situation where no COTS product is available as the crucial EPR system. Selecting and buying middleware first, is a good strategy if the rest is developed from scratch. But if the rest is to be COTS products, the approach doesn't work well.

4.4. High-risk areas handled too late

Once the contract is signed, the supplier should just deliver as promised. There may be a long period of time where the supplier develops glue-ware and extends his system, and at the end the customer will use the system. This is the ideal. However, it often happens that the supplier cannot fulfill his promises no matter how hard he tries. Typical trouble areas are performance (response time) and integration with other products. As an example, the supplier's present installations have around 30 users and perform well with them. But when facing the planned 2000 users, the response time skyrockets. Or the integration to system X sounded easy, but in practice it didn't work as expected.

Up front most customers don't care about this. What is the problem, they ask. If the supplier doesn't deliver as promised, he won't get his money; he may even have to pay a penalty. These customers assume that the supplier is just lazy and money will cure him. The fact may be that the supplier is unable to solve the problems. Such projects can drag on for years. The customer never gets an adequate system and the supplier loses fortunes. Court cases don't cure anything.

The root problem here is that the parties too late face the high-risk areas. The supplier tends to delay the hard parts and deliver the easy ones first. Ideally, the customer should ask for proof of the high-risk areas before selecting the winner. Often this is expensive, and it is unrealistic that all the proposers can do it at proposal time.

4.5. The COTS supplier gets a monopoly

Once the customer has got a complex system, he may want to extend it in various ways, for instance integrate it with new systems or modify the user interface. At this point the COTS supplier may have a monopoly. Only he can extend the COTS system. Some suppliers actually exploit this situation and charge unfair prices for extensions.

If COTS was a truly open market, the customer could just find another vendor - in the same way as you can switch from Ford to Toyota. Unfortunately, it is not so easy with complex software applications. It often takes a couple of years to replace an application - primarily due to organizational changes.

Customers try to prevent the monopoly by asking for "open interfaces" to the COTS product. Typically, the supplier will reply: "Yes, our system has open interfaces - it uses CORBA". Much later the customer realizes that although CORBA is used, the detailed formats and meanings of messages and API-calls are not described, and the supplier considers this information proprietary (see Lewis and Wrage, 2004, for a discussion of formats and semantics in "open interfaces"). In order to prevent

	Problems				
Solutions	Difficult to explore existing products	Mandatory requirements don't work	Product scope is fuzzy	High-risk areas handled too late	COTS supplier gets monopoly
Open target with structured questions	All requirements	All requirements			
Middleware and complex applications first			Selection criteria		
Optional sub-products			15-1 to 15-2		
Degrees of integration	15-3 to 15-10	15-3 to 15-10			
Technical interface for third party					16-1 to 16-8
User interface modifica-tion by third party					17-1 to 17-9
Trial period after contract to prove high-risk areas				Selection criteria	

Figure 1. The connection between problems and solutions
The cells refer to the requirements or contract parts that illustrate the solution. As an example, *Difficult to explore existing products* is handled by *Open target* requirements and *degrees of integration*. Examples of the latter are shown as requirements 15-3 to 15-10.

the monopoly, more than "open interfaces" are needed. Below we will show requirements that better guard against monopoly.

5. Dealing with the problems

In this section we will show solutions to the problems above. Figure 1 gives an overview of how the problems relate to the solutions. We will again illustrate the principles with an EPR acquisition.

5.1. Open target with structured questions

As explained in section 4.2, mandatory requirements don't work because even the best supplier may not meet all the requirements. The solution is to use *open-target requirements*, where the customer specifies his demands and expectations, and the supplier explains how he can meet the demands. This technique has been thoroughly described in Lauesen (2002). As an example, we could specify an integration requirement in this way:

R3: The system should share data with Labsys X. The customer expects that the latest results are always shown in the EPR system. The supplier is asked to explain his solution.

One supplier may reply that he always retrieves the data from Labsys X, another that he keeps a copy but updates it on a daily basis or at user request. Some suppliers may even say that they don't integrate with Labsys X at all, but they suggest another lab system that they integrate with.

Is it possible to verify open-target requirements? No, not in the traditional sense where a requirement is either met or not. Instead, the customer will give each supplier a score for how well he meets R3. He will give these scores when he assesses the proposals. The scores are the basis for selecting the winner.

However, if the requirements are too open, the supplier explanations may be long sales talks that are hard to compare. This may happen with R3 above, but the risk is higher if we omit the expectation about *latest results*:

R4: The system should share data with Labsys X. The supplier is asked to explain his solution.

We often see customers specify such requirements. The supplier has no idea what he is expected to reply and writes a sales talk. We need more structured questions to make sure we can compare solutions. All the requirements below are examples of this.

15. Laboratory system

The customer expects integration with his present system, Labsystem X, but may consider changing to a new lab system. The technical interfaces to Labsystem X are specified in Appendix ...

Degree of integration:	Example solution or offered solution:
1. The vendor offers an alternative system that meets the functional requirements in section ...	
2. The vendor offers integration with Labsystem X as specified in points 3 to 10 below.	
3. The user starts Labsystem X through the EPR system, then logs into Labsystem X, specifies the patient ID and requests the lab service through X's screens.	
4. As possibility 3, but the user doesn't have to log in and specify the patient ID.	The user and patient ID are transferred automatically.
5. The user requests lab services through the EPR system's screens in the same way as for other services.	The EPR system uses the API interfaces to Labsystem X.
6. States and results of the lab service are visible in the EPR screens in the same way as for other services.	
7. The EPR system can warn users about pending lab results in the same way as for other services.	
8. The EPR system shares database with Labsystem X. In this way EPR data and lab data are always identical.	
9. Lab data is periodically transferred from Labsystem X to the EPR system (replicated databases).	
10. Data for a single patient is transferred at user request from Labsystem X.	

Figure 2. Section 15 of an EPR requirements specification

Specifies optional integration with an existing customer system and an optional inclusion of a different system.

5.2. Purchase middleware and complex applications first

The customer may in principle buy some middleware first, then buy applications one by one. However, as explained in section 4.3, applications and middleware are closely related in practice and porting an application to the customer's middleware is hard, particularly for complex applications.

The general solution is to allow the supplier to deliver as much as possible, but give high priority to the complex applications. This vastly reduces the integration problems. The right choice in the EPR case would be to acquire a middleware product combined with the crucial EPR system. We have expressed this through the selection criteria (Figure 6), which give high priority to support of the basic clinical tasks (the EPR system) and to the ability for third party to extend the system. The details of the third-party ability is explained further in sections 5.6 and 5.7.

5.3. Optional sub-products

When the product scope is fuzzy, we need a way to allow the supplier to influence the scope. The solution is to use open-target requirements and let the supplier specify which of them he will meet and how.

Figure 2 shows section 15 of an EPR requirements specification. It allows the supplier to specify to what extent the customer's existing laboratory system will be integrated with the new EPR system. As an example, requirement 1 allows the supplier to offer an alternative lab system. The supplier writes his reply in column two. He may say "No, I don't offer an alternative system" or he may say "yes" and give a price for this option. Requirement 2 allows him to offer integration with the existing lab system. There are thus four basic options: the supplier offers only an alternative system; he offers only an integration; he offers both (and the customer can choose); or he doesn't deal with the lab system (a third party who knows the existing lab system might do the integration).

If the supplier delivers an alternative lab system, he delivers a larger product than if he just integrates with the old one. However, the integration may be so costly to him, that the larger product is cheaper than the integration.

5.4. Degrees of product integration

Figure 2 is also an example of how to avoid the mandatory requirements by means of structured questions. The supplier is not asked to do the integration in a specific way. He is asked to specify how close the integration will be. Requirement 3 is the weakest kind of integration. The supplier just allows the user to switch over to the lab system, but doesn't interfere in any other way. Requirements 4 and 5 represent solutions that are safer and easier to use. Requirements 6 and 7 state that from the user's point of view, the lab services look like any other patient service.

Requirements 8 to 10 allow the supplier to specify the actuality of data. When he fills in column two, he may go into more detail with the solution. These requirements look like requirements for a specific design, but they are alternative designs that the supplier can relate to. This helps the customer compare the proposals in a uniform way.

What if the supplier has a solution that we didn't know or didn't ask about? This is not a serious problem. The supplier just explains his solution in a right-hand box. Comparison is a bit harder - that is all.

The figure shows two examples where the customer has pre-filled the fields in the second column. He has mentioned an example solution, but it is not a requirement. The supplier may replace it with the solution he offers. Experience is that these suggestions help the supplier understand what the customer expects. They may also allow the supplier to explain that his product exceeds the customer's expectation.

Which degrees of integration should we describe in the table? The example deals with what is important in this project. Guo (2000) and Yakimovich et al. (1999) have more generic lists of integration dimensions, and our choice here was inspired by their list. Gorton & Liu (2002) explain about criteria for middleware products. They refer to their 150 item list of requirements to consider. It is proprietary, however. Such lists are precious extracts of long experience!

5.5. Integrating with future products

In the future, the customer may want to integrate the COTS-based system with other systems. What is involved in such an integration? Figure 3 shows the components involved.

Let us first look at integration with an existing system, for instance the existing LabSys-X. It will usually not talk the same "language" as the supplier's COTS product, so the supplier has to insert some *Glue code* or *Adapters* that translate the messages or files sent between the two systems. This may be a very complex piece of code, and other aspects are involved too, for instance controlling the run-time environments of the two systems.

In addition to the technical interfaces between the two products, some changes are usually needed to the user interface of the supplier's product. It may be buttons on the user interface that make the systems interchange data, or it may be new screens that show data from the existing LabSys-X. The supplier will do all of this as part of the delivery.

In general, integration with another system may be only the data-sharing. This just involves the technical interface. Or it may include integration with the existing user interface too.

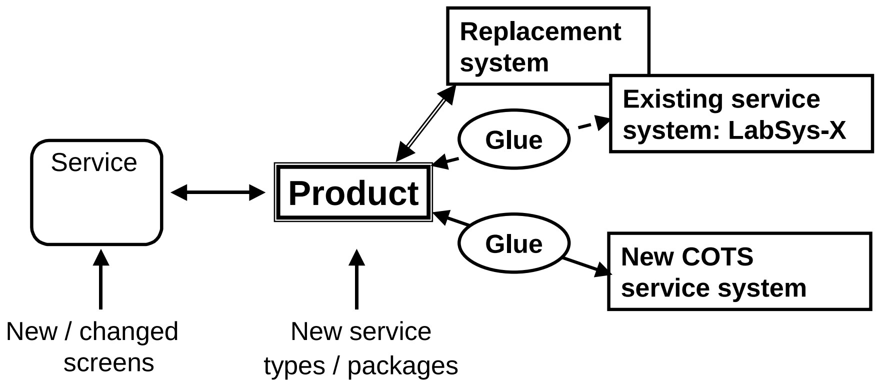

Figure 3. The architecture of integration.
Data integration and related user interface modification.

16. New production systems

The customer expects that new production systems can be integrated with the EPR system by a third party. A new production system might for instance be another laboratory system, a booking system, or an expert system that uses clinical data about a patient for searching international medical databases.

A production system may be tailor-made or a COTS system where third party adds glue code or adapters for integration with the EPR system.

Requirements:	Example solution or offered solution:
Interfaces - EPR in the server role:	
1. A new production system can retrieve and update data in the EPR system. This data should include data described in section ...	Might be done with messages, an API call or an SQL-query.
2. A new production system can invoke functionality in the EPR system. The functionality includes reporting warning events and printing results on department printers monitored by the EPR system.	
Interfaces - EPR in the client role:	
3. The EPR system can retrieve and update production data from a new production system.	Might be offered as messages, an API call or an SQL-query.
4. The EPR system can use functionality in a new production system. The functionality includes requests for service, warnings about pending requests, and print of requests.	
Documentation and rights:	
5. The technical interfaces to the EPR system (e.g. messages and API calls) and the format of data that can be retrieved or updated in the EPR system must be documented. The documentation must be understandable to a third-party software house and found suited for the intended development purpose.	The vendor should specifiy how much training is needed to understand the documentation. Sample documentation should be submitted as part of the proposal.
6. The customer and third parties must have the right to use the documentation and the technical interfaces.	
7. Third parties must have the right to extract and use data from the EPR system with the permission of the customer.	
8. Response times for the various interface functions should be specified, including their dependencies of database sizes and hardware platforms.	

Figure 4. Requirements specifying to what extent a third party can extend the system.

When we in the future try to integrate with a new COTS system, the principle is the same. We need Glue code and maybe user interface changes too. Who can do this? If only the supplier can do it, he has got a monopoly and can charge the customer accordingly. The alternative is that a third-party software house does it. Then there will be at least two integration providers, and the customer can compare their proposals. A special case is that the customer's IT department serves as a "third party".

However, a third party is not automatically able to do the integration. It has to be ensured through the requirements. For the technical interfaces between the products, the requirements must cover the technical aspects, plus the human ability and legal rights to do the integration. For the user interface something similar is needed, but the requirement details are different. We will look at the two interfaces one by one.

5.6. Technical interface for third party use

Figure 4 shows the requirements for a technical interface in an EPR tender. The primary concern is to integrate with future production systems that handle some kind of patient service, and this is what the example deals with. As explained in the introductory part of the requirements, the concept of a "production system" is very wide and would allow many kinds of systems.

Requirement 1 says that a new system can retrieve and update all data specified for the EPR system. Many systems can be integrated in this way. The supplier will in column two explain to what extent he can meet this requirement.

Requirement 2 specifies EPR functionality that must be available to a new production system. Very little is required because most of the communication is done by means of the data exchanged according to requirement 1. (Some more is required in the real case, for instance exchange of security information, but we have omitted it here.)

Requirements 1 and 2 specify what the EPR system should do in its role as a server for a new production system. Requirements 3 and 4 specify what the EPR system should do in its role as a client that requests services from a new production system.

One thing is that the two systems in principle must be able to exchange data and invoke functionality from each other. Another thing is whether a third party is able to make the systems do it. Requirements 5 through 8 aim at this.

Requirement 5 says that the technical interfaces must be documented. A requirement for documentation is very common, but it is also very common that the documentation actually supplied is useless. How can the customer guard against this? Requirement 5 says that it must be understandable and adequate for a software house. This requirement will be verified by having a software house check it. The software house will also check that the documentation matches what the system actually does. In order to reduce the risk, the supplier is asked to supply sample documentation as part of the proposal.

Some customers have experienced that although the technical interfaces are documented, neither customer, nor third parties are allowed to use the interfaces. In cases where the supplier also operates the system, he may even claim a right to the data stored on behalf of the customer. Requirements 6 and 7 guard against this. (In many cases these requirements are considered part of the contract rather than part of the requirements. This is not important as long as they are somewhere.)

Finally, requirement 8 asks for the response time to be specified for each of the functions on the technical interfaces.

Expert judgement

The first outlines of these integration requirements were presented and discussed with five potential suppliers, one by one. We will explain some of the more interesting comments.

Suppliers usually strongly resist that third parties integrate with their product. They find that the technical stability of their system is at stake (and maybe also their monopoly). What was the reaction during the presentation? It turned out that all suppliers accepted the need for third parties. Some of them even said that they knew they had refused earlier, but things had changed. They had to be competitive and open up.

Generally, they found the requirements above suitable for the purpose. One supplier had a system with an open interface that however was very hard to learn. He insisted that some weeks be allowed for the third party to learn the interface. Another supplier explained that any programmer would be able to use their open interface right away. The result was that we in the right-hand box for requirement 5 asked the supplier to specify how much training was needed.

Requirement 8 was more of a problem. The suppliers understood the need for knowing the response times. If the figures are unknown, it is impossible to assess what the interfaces can be used for in practice. However, they would hate to reply to it (some of them probably wasn't sure they would be able to do so). We kept the requirement, but as for all other requirements, the target is open and the supplier may reply that he doesn't have the figures. A supplier who can supply the figures will score higher than one who doesn't, but it is not sure he will win for just this reason.

5.7. User interface modifications by third party

Figure 5 shows the requirements for future modifications of the user interface. The primary concern is to let the user interface grow as hospital departments acquire new production systems and invent new ways of doing things.

Requirements 1 to 5 specify the functionality of tools to modify and expand the user interface. These tools should preferably be used by hospital staff, e.g. super users. Requirement 6 specifies that a third party must be able to add new types of user interface components. Requirement 7 calls for developing screens not only for PC's but also for PDA, mobile phone, etc.

Requirements 8 and 9 specify the rights to use the tools and the documentation. They are similar to the requirements for integrating new production systems.

In addition to these requirements, we had requirements for supporting the maintenance cycle. They included the need for a test version with realistic data, and means to distribute the new version in such a way that the relevant users are made aware of the new screens and services, while other

17. User interface modifications

The EPR system maintains patient data and service data and provides access through user screens. The customer expects that his own staff or a third party can modify the screens and add new screens. The vision is a kind of "Visual Basic for hospitals". The customer also expects that his own staff can define new types of services, for instance packages of simple services.

Requirements:	Example solution or offered solution:
Interfaces	
1. Staff can define new types of services and associated screens. The service may consist of several simpler services (see examples in section ...). It should be possible for super users to do much of this.	A program-like data declaration of the new service, combined with an interactive screen painter.
2. A screen can show and update data for a specific service including the constituent services.	
3. A screen can show computed data (views) based not only on the specific service, but all data described in the data model (section ...).	
4. Screens can activate functionality in the EPR system and in production systems integrated with it. Functionality includes requests for service, warnings, and prints.	
5. Screens can be composed from *controls* including text boxes, function buttons, tables, tab-sheets, pictures and graphs. The colors of the controls should be able to reflect data and service states.	
6. Third party can add new control types for use in the screens.	
7. Screens can be defined for several types of equipment, for instance PC, Tablet PC, PDA, mobile phone.	
Documentation and rights:	
8. The tools for composing new screens and defining new service types must be documented. This includes formats and explanation of data that may be shown in the screens. The documentation must be understandable to the intended audience (customer's super users and/or IT staff).	The vendor should specifiy how much training is needed to understand the documentation. Sample documentation should be submitted as part of the proposal.
9. The customer and third parties must have the right to use the documentation, the tools and the data in the EPJ system.	

Figure 5. Requirements specifying the modifiability of the user interface.

users are not troubled with these additions. In a hospital, there are so many medical specialities that only a small proportion of users need to know about new services. We don't go into details with these requirements since they are not related to the integration issues as such.

Expert judgement

When I reviewed these requirements with suppliers, some of them were scared of the very ambitious requirements. I had to reinforce several times that none of the requirements were mandatory. The purpose was to structure the supplier's description of what he could offer.

In addition to these concerns, the suppliers pointed out some ambiguities, and they suggested additional requirements (e.g. requirement 7 about PDA's and mobile phones).

5.8. Step-wise narrow-down and a trial period

While the criteria for commercial acquisition can be modified as new insights are obtained, this is not possible in public tenders. A tender approach narrows down the suppliers according to predefined steps and criteria. A typical tender process proceeds in this way:

Pre-qualification. The customer announces that a tender will be made in a certain area, for instance health care. No requirements are announced at this point, but the selection criteria for the pre-qualification are stated. They will typically be: *The supplier's financial status and track record, refer-*

Selection criteria

1. Efficient support for the basic clinical tasks.
2. Ability for the customer and third parties to extend the EPR system.
3. Adequate response times for the specified number of transactions and users.
4. Integration with the customer's existing systems.

The customer considers these criteria the most important for long-time success of the system. They are also considered the most risky areas. Failure in one of these areas will be very hard to repair late in the project.

The supplier should as far as possible demonstrate how far the criteria are met in the present system. This may be done during the presentation meeting when the customer evaluates the proposal. If it cannot be done at this time, the criteria must be demonstrated during the first six weeks after signing the contract.

Figure 6. The selection criteria from the tender documents.

ence customers, experience in the application area (for instance health care). Suppliers are invited to apply for pre-qualification.

The customer selects a limited number of suppliers, typically five. In this stage of the process, the effort for suppliers as well as customer is modest compared to the effort of replying to the real requirements.

Tender. Around a month later, the customer sends the requirements specification, the final selection criteria, and other tender material to the pre-selected suppliers. The suppliers get between 26 and 40 days to send a proposal. The customer selects the winner from an assessment of the proposals. As part of this, each supplier has a presentation session where he can explain and demonstrate his proposal.

At this point the traditional narrow-down process is finished. In many ways it works all right. However, the supplier has only *explained* what he will deliver, but he has not *proved* that he can do it. To reduce the risks for both parties, the supplier should deal with the high-risk areas very early.

Trial period. The solution is to encourage the winner to prove already at proposal time that he can do as promised. For areas where it would be too costly for him to prove it at this time, he is given a short period to do it after signing the contract. As an example, he may prove that he can meet the response time requirements by setting up an environment where 2000 users are simulated. Although this is not the final verification of the response time requirements, it is sufficient evidence at this point. The supplier can include the cost of the test in his proposal. Notice that such a trial period makes sense for COTS-based systems, which largely exist already. It would not be realistic for systems developed from scratch.

We have seen a variant of this approach where the customer selected two winners and paid both of them around $100,000 to go through the trial period. At the end of the period, he selected one of them to do the rest of the project. This is another step in the narrow-down process. For each step more work is needed to reach the next step, but fewer suppliers have to do it.

Selection criteria. Which criteria should be used for the selection? In principle the answer is easy: look at the major risks. A major risk is something that can wreck the project completely. Examples are inability to provide adequate response times, or inability to extend the system without the supplier having a monopoly. It is important to select suppliers that provably eliminate these major risks. Minor risks, such as some missing functionality, can be overcome by the supplier - at least if money is at stake - and need not enter the selection criteria.

Figure 6 shows an example of how the selection criteria and the trial period can be expressed in an EPR tender. Each criterion is a summary of many requirements. As an example, criterion 1 (*Efficient support for the basic clinical tasks*) comprises the scores for how well the EPR system supports a few basic user tasks. Criterion 2 (*Ability for the customer and third party to extend the EPR system*) comprises the third-party requirements in Figures 4 and 5.

Discarding the best supplier too early. A step-wise selection of this kind has the disadvantage that the best supplier - everything considered - may have been discarded early in the process. As an example, a new supplier entering the application domain with a superior product, may be discarded already in the pre-qualification stage. While the more exploratory narrow-down approaches could deal with this situation, the tender process cannot.

6. Conclusion

Buying COTS products that integrate with other products is a difficult business. Based on practical experience from tender processes, the paper suggests these improvements of current practice:

1. Use open-target requirements that allow the supplier to explain his solution in a semi-structured way.
2. Specify degrees of integration and ask the supplier which degrees he can provide.
3. Specify requirements that ensure that a third party can extend the product. The requirements must cover the interface functionality, the usefulness of the documentation, and the right to use the documentation and the interfaces.
4. Use a short trial period immediately after signing the contract to get evidence that the supplier can handle the high-risk areas of the project.

Acknowledgements

I want to thank Jens-Peder Vium, Henrik Willumsen, David Simonsen, Niels R. Larsen and Henrik Lindholm for some of the ideas behind this paper, Andrew Gabb for assuring me that I was dealing with important problems, and Torben Elin for comparing the approach with what he - as a customer - had been trying to do.

7. References

Albert, C. & Brownsword, L.(2002): Meeting the challenges of Commercial-Off-The-Shelf (COTS) products. In: J. Dean & A. Gravel (Eds.): International Conference on COTS-Based Software Systems, ICCBSS 2002, LNCS 2255, pp. 10-20.

Balk, L.D. & Kedia, A. (2000): PPT: A COTS integration case study. International Conference on Software Engineering, ICSE 2000 , pp. 42-49.

Bansler, J.P. & Havn, E.C. (1994): Information systems development with generic systems. In: Walter R.J. Baets (ed.), Second European Conference on Information Systems. Nijenrode University Press, 1994, pp. 707-715.

Bao, Y. & Horowitz, E. (1996): Integrating through user interface: A flexible integration framework for third-party software. Proceedings of COMPSAC '96. IEEE Computer, 1996, pp. 336-342.

Boehm, B. & Abts, C. (1999): COTS integration: Plug and Pray? IEEE Computer, January 1999, pp. 135-38.

Brownsword, L., Oberndorf, T. & Sledge, C.A. (2000): Developing new processes for COTS-based systems. IEEE Software, July/August, 2000, pp. 48-55.

Davis, L., Flagg, D., Gamble, R. and Karatas, C. (2003): Classifying interoperability conflicts. International Conference on COTS-Based Software Systems, ICCBSS 2003, LNCS 2580, pp. 62-71.

Gorton, I. & Liu, A. (2002): Streamlining the acquisition process for large-scale COTS middleware components. In: J. Dean & A. Gravel (Eds.): International Conference on COTS-Based Software Systems, ICCBSS 2002, LNCS 2255, pp. 122-131.

Guo, Jiang (2000): Interoperability technology assessment. Elsevier Science, Electronic notes vol. 65 No. 4., 2000.

Helokunnas, T. & Nyby, M. (2002): Collaboration between a COTS integrator and a vendor. In: J. Kontio & R. Conradi (Eds.): European Conference on Software Quality, ECSQ 2002, LNCS 2349, pp. 267-273.

Hornstein, R.S. & Willoughby, J.K. (2002): Realizing the potential for COTS utilization: a work in progress. In: J. Dean & A. Gravel (Eds.): International Conference on COTS-Based Software Systems, ICCBSS 2002, LNCS 2255, pp. 142-150.

Lauesen, S. (2002): Software requirements - styles and techniques. Addison-Wesley, 2002.

Lauesen, S. (2003): Task Descriptions as Functional Requirements. IEEE Software 2003, March/April, pp. 58-65.

Lauesen, S. and Vium, J.P. (2004): Experiences from a tender process. International Workshop on Requirements Engineering, REFSQ'04, 2004.

Lauesen, S. and Vium, J.P. (2005): Communication gaps in a tender process. Requirements Engineering Journal, ??, 2005.

Lewis, Grace A. and Wrage, Lutz (2004): A case study in COTS product integration using XML. International Conference on COTS-Based Software Systems, ICCBSS 2004, LNCS 2959, pp. 41-52.

Liu, A. & Gorton, I. (2003): Accelerating COTS middleware acquisition: The i-Mate process. IEEE Software, March/April 2003, pp. 72-79.

Maiden, N.A. & Ncube, C. (1998): Acquiring COTS software selection requirements. IEEE Software, March-/April, 1998, pp. 46-56.

Morisio, M. & Torchiano, M. (2002): Definition and classification of COTS: a proposal. In: J. Dean & A. Gravel (Eds.): International Conference on COTS-Based Software Systems, ICCBSS 2002, LNCS 2255, pp. 165-175.

Perrone, Vito (2004): A wish list for requirements engineering for COTS-based information systems. International Conference on COTS-Based Software Systems, ICCBSS 2004, LNCS 2959, pp. 146-158.

Sai, Vijay (2003): COTS acquisition evaluation process: the preacher's practice. International Conference on COTS-Based Software Systems, ICCBSS 2003, LNCS 2580, pp. 196-206.

Saur, L.D., Clay, R.L. & Armstrong, R. (2000): Meta-component architecture for software interoperability. IEEE Computer, November, 2000, pp. 75-84.

Semancik, S.K. & Conger, A.M. (2002): The standard autonomous file server, a customized, off-the-shelf success story. In: J. Dean & A. Gravel (Eds.): International Conference on COTS-Based Software Systems, ICCBSS 2002, LNCS 2255, pp. 234-244.

Wileden, J.C. & Kaplan, A. (1999): Software interoperability: Principles and practice. In: Proceedings of Software Engineering 1999, ACM, pp. 675-676.

Yakimovich, D., Bieman, J.M. & Basili, V.R. (1999): Software architecture classification for estimating the cost of COTS integration. In: International Conference on Software Engineering, ICSE '99, ACM, pp. 296-302.