Round Tripping in Component Based Software Development

Yves Wautelet  
Faculteit Economie & Management  
Hogeschool-Universiteit Brussel  
Brussels, Belgium  
yves.wautelet@hubrussel.be

Sodany Kiv, Vi Tran, Manuel Kolp  
Louvain School of Management  
Université catholique de Louvain  
Louvain-la-Neuve, Belgium  
{sodany.kiv, vi.tran, manuel.kolp}@uclouvain.be

Abstract—Several approaches have been proposed in literature to develop COTS-based software systems. Most of these focus on a functional and non-functional evaluation to select one or more products needing to be customized for specific requirements. Developing on the basis of components is however constraining for the software users as well as for software developers willing to provide tailored solutions. In this paper we propose a component-based development process using a two level components selection procedure with feedback loops. Analysts select one or more components to fulfill the identified goals and softgoals then a multi-agent system (MAS) managing the business logic delegates them the functional execution. Runtime feedback allows analysts to take decisions for component customization, abandonment or the inclusion of new ones so that equilibrium is found by round tripping between the analysis stage and the MAS execution.

Keywords-component-based software development; iterative life cycle; i* based software development; outbound logistics;

I. INTRODUCTION

COTS-based software development has become a strategic field for building large-scale and complex systems. However, since components are generically developed they do not entirely meet their client specific needs. In this paper we use COTS-components as low level functional providers interfaced with a MAS and develop a software process for such developments using a two level selection procedure with feedback loops. Moreover, the use of a MAS allows to build the business logic outside the software component so that they can be more easily integrated into the new software system. Also, by delegating the functional execution on a lower level basis, we have the ability to round-trip between component integration decision (at analysis level) and integrated component selection (at runtime). This approach allows:

• to integrate both functional and non-functional aspects in the components’ “evaluation for integration” process;
• to make the components’ “evaluation for integration” process continuous during the project life cycle;
• to compare the integrated components performance on a low-level functional basis to take the best performing solution on the basis of cost and QoS ontologies;
• to iteratively seek for an equilibrium;
• to ease the components integration process by building the business logic outside the components so that these require less customization;
• to ease the system adaptation process when facing changing requirements. Indeed, only the MAS’s business logic has to be adapted then.

The paper is structured as follows. Section II introduces the theoretical foundations, section III overviews the process application on the development of a collaborative software and finally section IV concludes the paper.

II. THEORETICAL FOUNDATIONS

This section points to the relevant MAS concepts, depicts the development process as well as the supporting software architecture.

A. MAS Concepts

A complete specification of the relevant MAS concepts in the context of wrapping over a COTS components including the dialogue procedure (through the concept of capability), and the transaction managed by the mediator agent can be found in [1].

B. Software Process

Figure 1 depicts the software development process in the form of a workflow using the semantics defined by the Software Process Engineering Meta-Model (SPEM, [2]). The process is structured as follows:

• At analysis stage:
  – an organizational model using a classic i* (i-star, [3]) approach allows to formalize the goals and tasks pursued by involved actors;
  – then, on the basis of an NFR goal graph [4] crossing the available components’ functional and non-functional attributes with the identified requirements a first component selection for integration is performed (more information about the analysis procedure can be found in [5]);
  – Components are thus evaluated at analysis level on the basis of non-functional aspects as well as high-level functional ones for eventual integration in the system under development;
At design stage, the business logic deduced from organizational models is integrated into a MAS acting as a wrapper over integrated COTS-components;

- At runtime, integrated components are competing (and judged by a mediator agent) to execute low-level functionalities requested by the MAS agents;
- Runtime feedback allows analysts in their high-level functional analysis:
  - to take further component integration decisions (removal of under-performing ones or to the integration of new ones);
  - when no existing solution can be found for obtaining MAS equilibrium, deciding to develop custom code (notably through the customization of existing integrated components).

C. Architectural Layers

Figure 2 generically depicts the supporting software architecture. It distinguishes 3 layers:

- The Graphical User Interface (GUI) layer where the services the application has to offer are composed by users through sets of tasks and goals;
- The Multi-Agent System (MAS) layer where the subprocesses (tasks or goals) are resolved following a defined business logic (encapsulated into the MAS) in the form of sequences of capabilities (called resolution paths). The communication with the lower layer is managed by a mediator agent (see [1]);
- The Capabilities layer where components advertise their ability to fulfill a capability request directly to the mediator agent and, when this agent orders it, components execute the capability.

III. ILLUSTRATIVE EXAMPLE

This section presents the application of the proposed methodology on a software development for supply chain management. Indeed, the aim of the project is to develop a collaborative platform for outbound logistics (OL) actors. Since space is limited we will focus on the selection of Transportation Management Systems COTS components.

A. Application Domain

Outbound logistics is the process related to the movement and storage of products from the end of the production line to the end user. In the context of this paper we mostly focus on transportation. The actors of the supply chain play different roles in the outbound logistic flow. The producer will be a logistic client in its relationship with the raw material supplier, which will be considered as the shipper. The carrier will receive transportation orders from the shipper and will deliver goods to the client, while relying on the infrastructure holder and manager. In its relation with the intermediary wholesaler, the producer will then play the role of the shipper and the wholesaler will be the client. The idea underlying the software development is to favour these actors’ collaboration. Indeed, collaborative decision will tend to avoid local equilibriums (at actor level) and wastes in the global supply chain optimisation, giving opportunities to achieve the greatest value that the chain can deliver at lowest cost. More information on the applicative package development as well as a broader definition of the involved actors can be found in [6].
In the example, the purpose is to develop a collaborative platform for outbound logistic actors where COTS components are notably used for specific purposes in the overall system; we more precisely integrate:

- **Fleet Management Systems** (FMS);
- **Warehouse Management Systems** (WMS);
- **Enterprise Resource Planning** (ERP) Systems;
- **Transportation Management Systems** (TMS).

### B. Analysis

1) **Organizational Analysis:** Figure 3 documents an i* diagram issued of organizational modelling. This strategic dependency diagram (SDD) depicts the relevant actors and their respective goal, task and resource dependencies. Those dependencies are further analysed in the strategic rationale diagram (SRD) which is not presented here due to a lack of space. It provides a more detailed level of modelling by looking inside actors. Each outbound logistic actor depicted above (as well as the final client) and the third party component we want to integrate in the global system are all represented as actors. These diagrams allow to get a first representation of the system to be at high abstraction level.

2) **Goals Formalization:** The NFR goal graph of Figure 4 represents both the functional and non-functional aspects that the TMS should fulfill. Non-functional requirements includes Global Optimization (of the supply chain), Collaboration, Flexibility and Security. After posing these non-functional requirements as goals to satisfy, we tend to refine them into sub-goals through AND and OR decompositions and with the help of the SRD. The goals that the component should reply to are depicted in the SRD and represented as operationalizing goals in the NFR goal graph. The interdependencies among the goals are also studied as shown in Figure 4. The desirability of different goals is illustrated in Figure 5.

3) **Component-level Based Selection:** Searching for COTS products in the marketplace and acquiring their documentations is not the focus of this paper. For illustration in this case study, we suppose that COTS product are selected after short-listing the available products and evaluating the pre-selected products.

### C. Design: Multi-Agent System Design

Based on the SRD and goal graph, we now can design the MAS as well as its behavior. We illustrate the realization path of the Transport load and route optimization goal. On the goal graph of Figure 4, we can notice **Selecting most...**
adequate transport is the way to realize the Transport load and route optimization goal. Figure 6 presents the sequence diagram documenting the evoked goal’s success scenario.

![Figure 6](image)

**Figure 6.** The resolution path of Select Most Adequate Transport goal.

**D. Runtime: Mismatches Identification**

<table>
<thead>
<tr>
<th>Capability</th>
<th>Inferred Definition</th>
<th>Calling Agent</th>
<th>Target</th>
<th>Present in COTS design?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateLogisticRequest</td>
<td>Constructor of the Logistic Request</td>
<td>Shipper</td>
<td>Logistic Request</td>
<td>Yes</td>
</tr>
<tr>
<td>EvaluateLogisticRequest</td>
<td>Compare the alternative routes to fulfill the logistic request according to the departure and destination points</td>
<td>Carrier</td>
<td>Logistic Request</td>
<td>No</td>
</tr>
<tr>
<td>SelectAlternativeRoute</td>
<td>Select the alternative route based on the provided services (services are the theoretical transportation offer provided by the carriers)</td>
<td>Carrier</td>
<td>Service</td>
<td>No</td>
</tr>
<tr>
<td>RecordServicesRequired</td>
<td>Record the required capability on each specific connection of the route</td>
<td>Carrier</td>
<td>Logistic Request</td>
<td>Yes</td>
</tr>
<tr>
<td>DefineShipmentDate</td>
<td>Define the shipment date</td>
<td>Carrier</td>
<td>Shipment</td>
<td>Yes</td>
</tr>
<tr>
<td>NotifyShipmentDate</td>
<td>Notify the shipment data to the logistics manager, shipper, and final client</td>
<td>Carrier</td>
<td>Carrier</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Figure 7.** Capabilities required to fulfill the Select Most Adequate Transport goal.

Figure 7 depicts the capabilities list required to fulfill this goal as well as the involved agents. At the lecture of the table, the reader will notice that the capabilities EvaluateLogisticRequest and SelectAlternativeRoute are the only ones, involved in the Select most adequate transport goal that are not fulfilled through a message already implemented into the component. Consequently, the two evoked capabilities have to be developed in an existing component (customization) or a new component should be included in the system through the analysis procedure. This information is the first raw data for the feedback loop.

**E. Feedback Loop**

Figure 8 depicts the mismatches of the selected COTS and their resolutions in aggregated terms i.e. at component level and no more at capability one. Further technical resolution and other iterations cannot be documented here due to a lack of space.

**Table: Mismatches documentation of the selected COTS.**

<table>
<thead>
<tr>
<th>Mismatch</th>
<th>Mismatch Type</th>
<th>Impact</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport load and route optimization</td>
<td>Process</td>
<td>Very strong impact on the success of the project since it concerns the very high desired goal.</td>
<td>Tailoring COTS product</td>
</tr>
<tr>
<td>Event handling</td>
<td>Process</td>
<td>Strong impact on the success of the project since it concerns the high desired goal.</td>
<td>Tailoring COTS product</td>
</tr>
<tr>
<td>Use non-standard data types</td>
<td>Differ</td>
<td>Some non-standard data types are used but it is acceptable.</td>
<td>Ignoring</td>
</tr>
</tbody>
</table>

**IV. Conclusion**

COTS-components can seldom be used as such into an specific software project and one need adequate methodologies to deal with code reuse possibilities. Flexibility both in component design and project integration are in that perspective from primary importance. That is why we proposed a flexible requirements-driven method for component-based development where COTS are used as low-level functional providers with the specific business logic issued of the custom analysis stage is developed in a MAS acting as a wrapper over integrated components. Moreover, the iterative approach allows reporting lacks at runtime so that analysts can take decisions to seek better MAS performance.

**REFERENCES**


