CoinsExtractor: The Architects’ Buddy in Identifying Conceptual Interoperability Constraints

Hadil Abukwaik  
University of Kaiserslautern  
Gottlieb-Daimler-Strasse 47  
67663 Kaiserslautern, Germany  
abukwaik@cs.uni-kl.de

Mohammed Abujayya  
University of Kaiserslautern  
Gottlieb-Daimler-Strasse 47  
67663 Kaiserslautern, Germany  
abujayya@rhrk.uni-kl.de

Dieter Rombach  
University of Kaiserslautern  
Gottlieb-Daimler-Strasse 47  
67663 Kaiserslautern, Germany  
rombach@cs.uni-kl.de

ABSTRACT
Identifying and resolving conceptual mismatches between two software systems are crucial for their successful interoperation. However, it is a tedious and time-consuming task for software architects to manually provide interoperability-relevant information about their software systems, especially in the case of large software systems. This paper demonstrates CoinsExtractor, a novel tool that aids architects in accomplishing this task effectively and efficiently. CoinsExtractor enables architects to determine the interoperable parts of their software system, to automatically extract the interoperability-relevant information about these parts from the existing UML documents of the system, and finally to document the extracted information in a standard way to be shared with third-party clients. To better support the conceptual analysis, CoinsExtractor categorizes the extracted information based on the concepts it presents.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques—Computer-aided software engineering (CASE); D.2.11 [Software Engineering]: Software Architectures; D.2.12 [Software Engineering]: Interoperability

General Terms
Documentation

Keywords
Conceptual interoperability, interoperability analysis, conceptual mismatches, information extraction, research tool

1. INTRODUCTION
Achieving successful interoperability [7] between two existing software systems requires identifying and resolving not only their technical mismatches, but also the conceptual ones. Such conceptual mismatches (e.g., architectural constraints, usage context, desired qualities, etc.) can lead to meaningless or incorrect results that, consequently, cause worthless technical integration or expensive rework.

Current interoperability-supporting approaches lack the focus on the conceptual aspects of interoperable software systems [3]. Besides, owners of such systems are like to publicly share their API documentations which are rich in technical information. These API documents may not expose all relevant architectural and conceptual information, which is rather hidden in in-house architectural artifacts like the UML diagrams. As a result, conceptual interoperability analysis cannot be properly performed and conceptual mismatches may not be detected. On another hand, proactively providing the relevant architectural information about interoperable software systems requires a considerable amount of manual effort from the software architects in capturing, structuring, and publishing these information.

To address these issues, we have previously proposed the Conceptual Interoperability Analysis (COINA) framework [2] that assists architects, at design time, in extracting and sharing the conceptual constraints of their interoperable software systems in a standard way. In this paper we demonstrate a novel tool, CoinsExtractor, with a number of features to make our framework applicable in practice and to take some burden off the architects’ shoulders.

CoinsExtractor supports architects with an easy-to-use interface to effectively determine the interoperable elements (i.e., data, functions, and components that are part of an interoperation with other systems) of their software system. The tool then automatically extracts the relevant Conceptual Interoperability Constraints (COINS) about the determined interoperable elements from existing UML diagrams. Also, CoinsExtractor enables architects to efficiently review, update, approve, or delete the automatically extracted COINS. Finally, the tool creates the COINS Portfolio that is a standard ready-to-share document associating each interoperable element with its COINS. The Portfolio categorizes the COINS according to the aspect they address (i.e., syntax, semantic, structure, dynamic, context, and quality) to facilitate the conceptual interoperability analysis task.

2. MOTIVATING EXAMPLE
In this section we outline a short example of the employment of the CoinsExtractor tool.

Consider a Smart Tractor (S1) that has been developed...
with a goal to make its software system interoperable (i.e., it is able to exchange data and services successfully and meaningfully [7]) with other software systems (e.g., farmer mobile apps, farm management systems, etc.). To achieve this goal, the developers of S1 create its API documentation and share it publicly for interested third-party clients.

At some point in time, owners of another software system like the Smart Farm (S2) get interested in integrating an instance of S1 into their ecosystem. The software architect of S2, Adam, is responsible for identifying and resolving any mismatch with S1. After the manual reading and analyzing of the text in S1’s API documentation for all data and functionalities, Adam could identify the explicitly expressed technical constraints of S1 (e.g., the RemoteSteering functionality receives NewDirection as GPS coordinates and sends a confirmation/error code, all in XML format). However, from the technical-oriented API documentation, Adam could not infer the conceptual constraints of S1 that are hidden or mistakenly forgotten in its unshared UML diagrams. Possible examples are: (1) a use case diagram of S1 states that the AutoSteering functionality is assumed to be invoked by ActualDriver setting in the tractor in the field and not by RemoteDriver; (2) a sequence diagram of S1 shows that the interaction type with the LiveTracking functionality is assumed to be synchronous; or (3) an annotated UML note on a sequence diagram for the ObstacleDetection functionality mentions preferences on the resolution of the field image input. Apparently, missing such information would not hinder the technical interoperation, but would eventually introduce unexpected conceptual mismatches and expensive rework. To overcome such an issue, interoperability analysts of the third-party clients, like Adam, might send inquiries to the owners of S1 asking for further conceptual information. However, this solution leads to repeated costs for S1 owners in handling sent inquiries, besides the waiting time that clients have to pay. Another solution would be that the architects of S1 extract and share the conceptual constraints related to exchanged data or functionalities by their interoperable software systems. Obviously, applying this solution manually is not trivial, time-consuming, subjective, and could be error-prone, especially for large systems. Therefore, a tool supporting architects in providing such conceptual constraints of their interoperable systems is essential!

### 3. THE COINS EXTRACTOR TOOL

In this section, we present the CoinsExtractor tool’s features, design, implementation, and scalability analysis.

#### 3.1 Approach and features

The tool’s semi-automatic approach for sharing the conceptual constraints of interoperable systems includes three main phases: setup, extraction, and finalization [2]. The tool takes the software system’s UML diagrams as an input and produces the COINs Portfolio document, which includes the conceptual constraints about the interoperable elements, as an output. This is implemented by the following features:

**Identification of interoperable elements.** CoinsExtractor allows architects to annotate any component, class, use case, or actor with an “Interoperability Type” property. This property declares whether the element plays role in interoperations with other software systems. There are three possible values for the interoperability property: interoperating party (i.e., external software system which interoperates with the system), exported element (i.e., element that the system provides to an interoperating party), or imported element (i.e., element that the system acquires from an interoperating party). Once architects determine an element as interoperable, it appears in the system’s list of interoperable elements (see Figure 1), which steers the next features. This step requires a one-time manual effort to capture architects’ knowledge, which will be reused in all future interoperations.

**Automatic extraction of interoperable elements’ COINs.** This feature saves the architects’ effort in selecting only relevant pieces of information, which need to be shared about the interoperable elements, from the whole UML document. This also preserves consistency in what is being shared and how about the interoperable elements on the project level and among projects. The tool parses the UML input and checks it against our predefined COIN extraction rules [2]. These rules cover extraction from component, deployment, class, use case, and sequence diagrams. Found COIN instances are saved in a list of candidates, where each is detailed according to our designed COIN sheet. This sheet specifies: interoperability element name, interoperability level [10], constraint’s category, value, and weight. It has also optional fields (e.g., formal notation, consequent issues, and comments) that architects can fill as needed. Figure 2 shows a COIN instance and its detailed description.

![Figure 1: Example of a list of interoperable elements of a software system.](image)

![Figure 2: Example of an extracted COIN.](image)
Table 1: COINs’ sources and categories

<table>
<thead>
<tr>
<th>Source diagram</th>
<th>COIN name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Component Layering</td>
<td>Structure</td>
</tr>
<tr>
<td>Component</td>
<td>Component distribution</td>
<td>Structure</td>
</tr>
<tr>
<td>Deployment</td>
<td>DB distribution</td>
<td>Structure</td>
</tr>
<tr>
<td>Deployment</td>
<td>Component distribution</td>
<td>Structure</td>
</tr>
<tr>
<td>Class</td>
<td>Structural multiplicity</td>
<td>Structure</td>
</tr>
<tr>
<td>Class</td>
<td>Inherited constraints</td>
<td>Structure</td>
</tr>
<tr>
<td>Use Case</td>
<td>Allowed actors</td>
<td>Context</td>
</tr>
<tr>
<td>Use Case</td>
<td>Usage multiplicity</td>
<td>Context</td>
</tr>
<tr>
<td>Use Case</td>
<td>Inherited constraints</td>
<td>Structure</td>
</tr>
<tr>
<td>Sequence</td>
<td>Interaction type</td>
<td>Dynamic</td>
</tr>
<tr>
<td>All diagrams</td>
<td>Natural language constraints</td>
<td>NA</td>
</tr>
</tbody>
</table>

Automatic categorization for extracted COINs. The tool provides two views for the extracted COINs. The first is diagram-based where architects can navigate the extracted COINs based on the diagram they have been retrieved from. This helps architects in tracing the source of interoperability constraints in their systems (see the pane tree in Figure 2). Table 1 lists the different conceptual constraints that the CoinsExtractor covers, their categories, and their source UML diagram. The second view is element-based where COINs can be navigated according to the interoperable element they relate to. This view is also used in the final shared Portfolio to help third-party clients focus on the COINs of the elements they want to interoperate with.

Manual filtering for automatically extracted COINs. The architects have the option to manually approve or delete the automatically extracted COINs. Furthermore, they can review the COINs’ sheets and edit them as needed.

Manual adding for COINs. In the case of missing constraints, the tool enables architects to manually add more COIN instances to any of the interoperable elements.

Automatic generation for the COINs Portfolio. The CoinsExtractor relieves architects from the manual gathering and classification of all extracted and manually added COINs. It generates the final, ready-to-share, web-based COINs Portfolio seen in Figure 3. The Portfolio has two parts. The first dedicated to the actual COINs of exported elements that third party clients need to pay attention for when deciding to interoperate with the software system. While the second part is about the expected COINs from imported elements that architects need to look for when selecting a software system to interoperate with. In fact, the Portfolio’s both parts share the same goal of supporting the conceptual interoperability analysis in detecting conceptual mismatches early in the project.

3.2 Design

The CoinsExtractor has a multi-layered architecture as shown in Figure 4. The Presentation layer allows the interaction between the tool and architects using it through windows for identifying the interoperable elements, filtering extracted COINs, and adding more ones. It also visualizes the results of such actions in a table of interoperable elements and a table of COINs. The second layer is the Business that includes: (1) The Business logic which is responsible for processing the input UML diagrams and for extracting the COINs from them. It contains the working units of the tool like, the COINs Extractor and the Portfolio Generator. Additionally, we have the plan to implement the Portfolio Formalizer and Mapper units in the near future. (2) The Business Entities are the predefined COIN templates (which can be extended to cover more constraints) and data structures needed for the tool’s functionalities (e.g., the Node position which is used to determine elements distribution). The Data Access layer reads input from the UML database and writes results into the output file.

Figure 3: Example of a system’s COINs Portfolio.

Figure 4: Architecture of the CoinsExtractor (CE).
5. CONCLUSION AND FUTURE WORK

In this paper we have presented the CoinsExtractor tool that supports performing the conceptual interoperability analysis between two software systems. It helps software architects in extracting the conceptual constraints about their interoperable systems. The tool also documents the extracted constraints in a standard way to be shared with interested third-party clients. From a technical point of view, we implemented the CoinsExtractor, using C# .NET, as an Add-On for the Enterprise Architect tool.

Based on our ongoing research, we aim to extend the tool in the future with additional components: (1) the formalizer which transforms the COINs Portfolio into a formal notation, and (2) the Mapper which maps two systems’ formal Portfolios to find their conceptual mismatches that should be resolved to enable their interoperation. We are also planning to evaluate the tool with practitioners to test our hypotheses about its effectiveness, efficiency, and acceptance.

6. ACKNOWLEDGEMENTS

This work is part of the PhD research of Hadil Abukwaik under the supervision of Prof. Dieter Rombach. We thank Mohammed Abufouda, Dr. Matthias Naab, and Christian Wolschke for their constructive comments.

7. REFERENCES

APPENDIX

A. COINS EXTRACTOR PRESENTATION

The CoinsExtractor tool is a complete academic-research tool that we implemented as an Add-On for the Enterprise Architect tool. We plan to extend the tool with further features serving the goal of supporting industrial architects in their conceptual interoperability analysis task.

A YouTube video demoing our tool within a toy example is available at: https://www.youtube.com/watch?v=7iGLS66fWWk.

This tool is available on request from its page: (http://abukwaik.com/site/projects/coins-extractor).

The following subsections show a brief description of what the CoinsExtractor presentation will incorporate.

A.1 Constraints extraction in CoinsExtractor

We will give an introduction of the problem space and the proposed solution approach implemented by CoinsExtractor. Each step of the proposed approach is presented with emphasis on the support that the new tool provides to it.

A.2 Features

We will illustrate the novel features included in the CoinsExtractor and its practical benefits. This includes explaining the required input for each feature and the produced results.

A.3 Modeling

We will give a brief presentation of the multi-layered architecture of the CoinsExtractor tool and explanation for the taken design decisions. This tool is built under the academic research and aimed to serve architects in industry.

A.4 Implementation

We will justify the technology choices we made and how we implemented the tool as an Add-On for the Enterprise Architect tool.

A.5 Execution

We will run a demonstration of the tool on a toy example due to the limited availability of industrial architectural documents. A selected number of the tool’s features will be shown.

A.6 Future Work

we will state our planned enhancements for the tool that aims at better supporting the conceptual interoperability analysis. Feedback from the audience will also be collected to improve the tool.