ARTIST

FP7-317859

Advanced software-based seRvice provisioning and migraTIon of legacy Software

Deliverable D8.2

Methodology and Techniques for Model Discovery

Editor(s):	Hugo Bruneliere & Javier Canovas
Responsible Partner:	INRIA
Status-Version:	Final - V1.0
Date:	28/03/2014
Distribution level (CO, PU):	PU

(cc) BY-SA

Project Number:	FP7-317859
Project Title:	ARTIST

Title of Deliverable:	Methodology and Techniques for Model Discovery
Due Date of Delivery to the EC:	31/03/2014

Workpackage responsible for the Deliverable:	WP8 – Legacy Product Analysis by Reverse Engineering
Editor(s):	Javier Cánovas, Inria Hugo Brunelière, Inria
Contributor(s):	Inria, TUWien, ATOS, Spikes, Sparx
Reviewer(s):	Domenico Presenza (ENG)
Approved by:	All Partners
Recommended/mandatory readers:	WP6, WP8, WP9

Abstract:	Detailed specification of the global methodology and underlying model-based techniques to be considered for discovering complete low-level initial models out of existing software artefacts. To be then used as base for implementing such a dedicated support for the Java & C# technologies (cf. D8.2.1, D8.2.2 and D8.2.3).
Keyword List:	Model Driven Reverse Engineering (MDRE), Model Discovery, MoDisco
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/

(cc) BY-SA

Document Description

Document Revision History

Version	*Date*	*Modifications Introduced*	
		Modification Reason	*Modified by*
v0.1	24/01/2013	First draft version (from WD8.2 as internally submitted on M6)	Javier Cánovas (Inria), Hugo Brunelière (Inria)
v0.2	07/03/2014	Integration of all the new content as collaboratively elaborated and collected using a shared Google document. Use of the proper ARTIST template document.	Integration by Hugo Bruneliere (Inria) Content from Hugo Brunelière (Inria), Javier Cánovas (Inria), Yosu Gorroñogoitia (ATOS), Bram Pellens (Spikes), Alexander Bergmayr (TUWien), Horst Kargl (Sparx)
v0.3	12/03/2014	Version ready for internal review	Hugo Bruneliere (Inria), Javier Cánovas (Inria)
v1.0	28/03/2014	Final version integrating comments from internal reviewer - Domenico Presenza (ENG)	Hugo Bruneliere (Inria), Javier Cánovas (Inria)

(cc) BY-SA

Table of Contents

Table of Contents 4
Table of Figures 5
Table of Tables 5
Terms and abbreviations 6
Executive Summary 7
1 Introduction 9
1.1 About this deliverable 10
1.2 Document structure 10
2 Model Discovery in (Model-Driven) Reverse Engineering 11
2.1 Reverse Engineering 11
2.2 Model-Driven Reverse Engineering (MDRE) 11
2.3 Focus on Model Discovery 13
3 Requirements for Model Discovery in ARTIST 14
3.1 Base Requirements 15
3.2 WP9/Forward Engineering-related Requirements 15
3.3 User Interface Requirements 16
3.4 Cross-WPs/Activities Requirements 16
4 Support for Model Discovery in ARTIST 16
4.1 Overall Methodology for Model Discovery 16
4.2 Concrete Technical Support for Model Discovery 19
5 Model Discovery Experiments in ARTIST 21
5.1 Experimenting with the Java PetStore example 21
5.1.1 Overview of the Model Discovery process 21
5.1.2 Model Discovery from the J2EE 1.3 version 22
5.1.3 Model Discovery from the Java EE 6 version 22
5.2 Experimenting with the DEWS use case 23
5.2.1 Overview of Model Discovery activities 23
5.2.2 Initial UML and XML Model Discovery 23
5.2.3 Further Treatment by Transformation 23
5.3 Experimenting with the LoB use case 24
5.3.1 Generic UML Class Diagram Discovery 24
5.3.2 SharePoint Data Model Discovery 24
5.3.3 Package Dependency Discovery 24
6 Conclusions 25
7 References 26

(cc) BY-SA

Table of Figures

FIGURE 1: THE MODEL DISCOVERY PHASE 17
FIGURE 2: MODEL DISCOVERY - A POSSIBLY TWO-STEP APPROACH..... 18
FIGURE 3 MODEL DISCOVERY EXPERIMENT FOR THE JAVA PETSTORE BASED ON J2EE 1.3 22
FIGURE 4 MODEL DISCOVERY EXPERIMENT FOR THE JAVA PETSTORE BASED ON JAVA EE 6 22

Table of Tables

(cc) BY-SA

Terms and abbreviations

ADM	Architecture Driven Modernization
ASTM	Abstract Syntax Tree Metamodel
ATL	ATL Transformation Language
EC	European Commission
FE	Forward Engineering
KDM	Knowledge Discovery Metamodel
MDRE	Model Driven Reverse Engineering
OMG	Object Management Group
RE	Reverse Engineering
UML	Unified Modelling Language

(cc) BY-SA

Executive Summary

As stated by its name, ARTIST deliverable D8.2 is about proposing and describing a methodology and corresponding techniques for dealing with the Model Discovery step in the context of a Model Driven Reverse Engineering (MDRE) phase within a more global model-based migration process. The proposed Model Discovery approach and supporting tooling is positioned according to the current situation in the RE/MDRE area and targets a set of core requirements that have been identified notably in the context of the ARTIST project. While continuously under development/improvement, the supporting techniques are also concretely deployed and used within several ongoing experiments on actual ARTIST use cases.

This document and the described Model Discovery step are perfectly integrated within the whole ARTIST methodology for migrating existing applications to the Cloud. Thus it actually corresponds to the task *MODELDISCO*, as part of the *Application Discovery & Understanding* activity in the *Migration* phase (cf. ARTIST deliverable D6.2.2 – ARTIST Methodology M24). This task is concretely supported by several already existing, extended and/or newly developed components constituting the so-called ARTIST Model Discovery Toolbox (cf. ARTIST deliverable D8.2.1 and latest versions to come). The main objective of this practical methodology and implementing components is to correctly feed the next task of the ARTIST process, namely *MODELUNDER* – Model Understanding, with the required set of models for the remainder of the migration process to be accomplished as expected.

This deliverable starts by setting up the overall context of Model Discovery according to the state-of-the-art in Reverse Engineering (RE) in general and MDRE in particular (the ARTIST methodology being a model-based approach). Then, relying on this evaluation of the current situation and also from the specificities of ARTIST (notably its use cases), it exposes the main requirements for Model Discovery in such an industrial context of software migration. The core of the document details the proposed methodology to tackle this problem, as well as the associated technical support developed and promoted in ARTIST to concretely handle it. In order to illustrate the actual experiments and related developments performed so far in the project, the deliverable also describes the work realized during the first year and a half of the project about applying this Model Discovery methodology and corresponding techniques. Finally, the document ends by summarizing the provided content and opening some further progresses to be accomplished during the remaining year and a half of the ARTIST project.

The main result brought by this deliverable is the proposed practical and general methodology for Model Discovery as well as its mapping to a concrete implementing toolbox (cf. ARTIST deliverable D8.2.1 and latest versions to come for more details on the actual tooling support). Up to our knowledge and to the state-of-the-art elaborated in the context of this document, we have not been able to find trace of a similar generic MDRE approach to deal with the particular problem of Model Discovery. Available solutions and tools are usually following ad-hoc methodologies and are very often quite oriented to specific software technology or migration scenario(s). Thus, this deliverable provides a step in the direction of having a more systematic approach for Model Discovery independently from the input technologies or targeted scenarios.

For the second phase of the ARTIST project, the work in this area of Model Discovery will mainly consist in improving the practical tooling support to this methodology. Thus, the Model Discovery Toolbox (cf. ARTIST deliverable D8.2.1) will be upgraded and extended with new components according to the needs. This will result in two new versions of this toolbox to be

(cc) BY-SA

delivered respectively at M24 (D8.2.2) and M30 (D8.2.3). In order to perform this, the already submitted Taxonomy of Legacy Artefacts (cf. ARTIST deliverable D8.1) will be used as a valuable information provider regarding the more relevant ways to implement the currently missing support. Notably, one of the objectives is to enhance the actual applicability of the approach by extending the corresponding tooling support with some more components targeting the C#/.NET environment and related technologies.

(cc) BY-SA

1 Introduction

Reverse engineering is almost as old as computer science itself. Initially targeting hardware analysis [1], it quickly extended its scope to mainly focus on software systems [2]. Then, following the spectacular expansion and advent of software from the end of the 80s, RE has been usually regarded in the context of dealing with already existing systems which are often still running critical operations for companies.

Model-Driven Engineering (MDE) is a more recent paradigm where models are considered as first-class artefacts within software engineering processes (e.g., development, maintenance, evolution). In this context, the use of models is generally intended to allow raising the level of abstraction and automation, thus helping to better deal with complexity and to improve both productivity and quality of corresponding processes.

While MDE was initially mainly used in forward engineering processes, where software artefacts (e.g., source code) can be generated from models (e.g., specification), corresponding principles and techniques are also now more and more used in the context of reverse engineering processes, where an already existing software system must be analysed and further understood.

Within the context of model-based migration or modernisation processes (such as proposed by ARTIST), it is thus normal to follow an approach relying on the well-known Horse-shoe model [3] which generally splits such a given process into three distinct but complementary phases:

1. *Reverse Engineering*, which consists in obtaining models representing the software system to be modernised;
2. *Restructuring*, which is about transforming these models into new ones representing the target modernised system;
3. *Forward Engineering*, where the modernised system is being generated from the updated or refined models.

What follows focuses more particularly on the Model-Driven Reverse Engineering (MDRE) aspects, as the application of Model-Driven Engineering (MDE) principles and techniques to reverse engineering problems.

The main goal of MDRE is to allow producing relevant model-based views on existing software systems in order to facilitate their understanding and then further manipulation (e.g., to perform the remainder of the modernisation process). Thus, a standard MDRE process is normally composed of two main steps:

1. Model Discovery, where the required initial low-level models are extracted from the existing software artefacts;
2. Model Understanding, where the previous raw models are “digested” (analysed/treated) to obtain models at a level of abstraction suitable for the next step(s) of the process.

While the model discovery phase largely depends on the source technology to analyse (e.g., the software system’s programming languages), the model understanding phase is more related to the goal of the overall modernisation process itself (e.g., software analysis, migration, etc.). In this present document, a stronger focus is put on the Model Discovery phase as the initial step of any MDRE process.

(cc) BY-SA

1.1 About this deliverable

This deliverable describes the Model Discovery methodology and corresponding techniques as developed, deployed and used within the context of ARTIST.

1.2 Document structure

This deliverable first introduces the current state-of-the-art in the area (Section 2) before summarizing the main requirements as expressed by the ARTIST use case providers (Section 3). Then the ARTIST support for Model Discovery is concretely detailed (Section 4), including the overall methodology and related tooling support as provided by the ARTIST Model Discovery Toolbox (cf. ARTIST D8.2.1 and latest versions for more details). Finally, some concrete experiments performed during the first year and a half of the project are briefly described (Section 5) before concluding on next steps and further work regarding these Model Discovery aspects (Section 6).

(cc) BY-SA

2 Model Discovery in (Model-Driven) Reverse Engineering

The objective of this section is to introduce the main concepts and current state-of-the-art in Model Discovery by situating it according to its "parent" (Model-Driven) Reverse Engineering. Thus this section goes from general Reverse Engineering to Model-Driven Reverse Engineering, as the application of MDE principles and techniques to Reverse Engineering, to finally focus on more specific Model Discovery aspects.

2.1 Reverse Engineering

In contrast with forward engineering, reverse engineering is commonly defined as "the process of examining an already implemented software system in order to represent it in a different form or formalism and at a higher abstraction level" [2]. The key notion is the one of representation, which can be associated to the concept of *model* in the large sense of the word. The objective of such representations is to have a better understanding of the current state of a software system, for instance to correct it (e.g., to fix bugs or ensure regular maintenance), update it (e.g., to align it with the constantly evolving organizational policies and rules), upgrade it (e.g., to add new features or additional capabilities), reuse parts of it in other systems, or even completely re-engineer it. This is happening now more than ever, due to the need of not only satisfying new user requirements and expectations but also for adapting existing software systems to emerging business models, adhering to changing legislation, coping with technology innovation and preserving the overall system structure from deteriorating [4].

Clearly, given that reverse engineering is very often a time-consuming and complex process, any reverse engineering solution that could (semi)-automate the process would bring precious help to the users (e.g., software architects or engineers) and thus facilitate its larger adoption. Nevertheless, such a solution would need to face several important problems:

- Technical heterogeneity of the existing software systems;
- Structural complexity of these software systems;
- Scalability of the developed solution;
- Adaptability/portability of this solution.

Previous attempts in the 90s to build (semi-)automated solutions were first based on object-oriented technologies [5]. Several proposals and tools in that area, e.g., aimed at program comprehension [6], appeared around at that time. Among the many proposals, some focused on the extraction and analysis of relevant information from existing source code or software components in general [7] whereas others focused on relational databases [8], compiled code or binary files [9], etc. However, these works were quite specific to a particular software technology or a given reverse engineering scenario.

Following the emergence of Model Driven Engineering (MDE) [10], corresponding concepts and approaches have been used in order to propose alternative ways of building effective reverse engineering solutions: this is the so-called Model-Driven Reverse Engineering (MDRE), as presented in the next section.

2.2 Model-Driven Reverse Engineering (MDRE)

Model-Driven Reverse Engineering (MDRE) is the application of MDE/model-based principles and techniques to the reverse engineering domain [11]. At the beginning, models (in the MDE sense) were mainly considered to specify systems prior to their implementation (forward engineering). Instead, MDRE proposes to first obtain models from already existing software

(cc) BY-SA

implementations. These models are then used as the starting point for all the following reverse engineering activities. Moreover, higher-level views of the system (e.g., design models) or of its domain (e.g., application domain models) used in forward engineering are particularly relevant to improve the related maintenance and evolution/migration processes. Thus, there is still naturally an inherent complementarity between forward engineering and reverse engineering, especially when homogeneously treated and combined using models (such as in the ARTIST overall methodology for instance).

More generally, MDRE formalizes the representations (as models) that are derived manually or (semi-)automatically from existing software systems in order to ensure a (better) common understanding of their actual contents. Therefore, MDRE directly benefits from the genericity, extensibility, coverage, reusability, integration and automation capabilities of MDE technologies to provide a good support for reverse engineering.

MDRE is regarded as one of the fundamental applications of MDE [12]. The growing interest around it motivated the OMG to launch the Architecture Driven Modernization (ADM) Task Force [13] with the main objective to propose a set of standard metamodels useful for modernisation projects, i.e., technical migrations from old or obsolete technologies to more recent ones. Based on the work from this task force, a few proposals such as [14] intent to combine these standards in a methodological framework. MDRE is also useful for software analysis purposes (e.g., as proposed by the Moose platform [15]). More generally, MDRE is required when dealing with Model Driven Software Evolution [16], i.e., global scenarios including any kind of possible modification on legacy systems (structural, functional, maintenance, etc.) and not only pure (technical) modernisation.

In all cases, a MDRE phase is needed first in order to obtain the required models from the considered systems so that they can then be analysed, modified or evolved. An MDRE process is normally composed of two main phases:

1. **Model Discovery**, where initial low-level models are extracted from the existing software artefacts. The term low-level models is used when their level of abstraction is quite the same as the software artefact they are representing, that is, there is a direct systematic correspondence between software artefact elements and model elements. No further data or content analysis is realized at this step. For instance, a Java code model will represent every Java element in the code (statements, variables, classes, etc.), or a generic UML Class model will describe each Java package, class, attribute as corresponding UML elements. The next sections in this document will elaborate more on this phase.
2. **Model Understanding**, where the previously discovered models are deeper processed and/or refined to obtain the higher-level models needed for the next steps of the process, e.g., forward engineering (cf. ARTIST D8.3 for more details). In this case, higher-level models represent digested information from the software artefact, thus usually requiring a change in the representation format and data/content analysis process. For instance, following the previous example, generic UML models could be processed to obtain UML Component model as higher-level models. Building such component models require a further transformation/analysis of the information contained in the previously discovered low-level models (e.g. using some heuristics or knowledge bases).

Model Discovery is a metamodel-driven phase in charge of representing the legacy system as a set of models with no loss of required information. On the other hand, the Model Understanding phase is completely model-based: it takes as input the models from the

(cc) BY-SA

previous phase and generates the required output (models) thanks to a chain of model transformations. While the components to use in the model discovery phase largely depend on the source technology to analyse (i.e., the metamodels to employ are for instance adapted to the legacy system programming languages), the components in the model understanding phase are more related to the actual objective of the overall reverse engineering process itself. Such a goal could be simply system comprehension (e.g., of its architecture or implemented functionalities), further analysis (e.g., business rule extraction, non-functional property verification), re-engineering (e.g., technical refactoring or migration), etc.

As just stated earlier, the remainder of this document focuses on the Model Discovery aspects while ARTIST deliverable D8.3 describes more specifically the Model Understanding step.

2.3 Focus on Model Discovery

Now putting the emphasis more particularly on the Model Discovery phase, there have already been several initiatives (e.g., via frameworks, tools, environments) intended to provide some support regarding the related discovery aspects.

For instance, several past works have described how to migrate from a particular technology to another one using dedicated components and mappings, e.g., from procedural COBOL to object-oriented COBOL [17], from COBOL to Java [18], from COBOL to Java EE technologies [19], from Mainframe to Java EE [20] or from Oracle Forms to Microsoft .NET [21]. These approaches/tools (such as Sparx Enterprise Architect [22], to be also used and extended in ARTIST) use specific and/or proprietary parsers, grammars, metamodels, etc. that can be considered as model discoverers. However, the genericity and reusability of these components in other contexts (e.g., other paradigms, legacy technologies or target environments) can be quite limited. Moreover, they are generally not (fully) open source and quite often rely on proprietary platforms (e.g., for their actual execution).

There also exist several more specific tools allowing discovering different types of models out of various kinds of software artefacts. Columbus [23] is offering parsing capabilities from C/C++ source code and allows serializing the obtained information using different formats (e.g., XML, UML XMI). JaMoPP [24] or SpoonEMF [25] are providing alternative (but complete) Java metamodels and corresponding model discovery features from Java source code. Complementary to this, some other kinds of software artefacts can also be relevant in a Model Discovery context. For example, execution traces captured during the running of a given system have been used to generate UML2 sequence diagrams showing dynamic views on this system [26]. In [27], available Web service descriptions have been expressed as UML models to be able to later compose them for future integration in a different framework. In [28] a tool able to obtain models out of JSON data is presented. All these components can also be seen as potentially reusable model discoverers in the context of ARTIST.

As far as more generic reverse engineering platforms/environments are concerned, one of the first to appear was [3] that tried to integrate together various architectures and code re-engineering tools. It is based on a common inter-operation schema called CORUM to be considered as a "standard" for the different tool providers in the area. However all discovered models have to conform to this CORUM “metamodel”, and the solution is not proposing different base metamodels or allowing plugging other ones into the framework if required for a given software technology.

Appearing a bit more recently, Moose [15] is a well-recognized platform providing a toolbox dedicated to the building of various software and data analysis solutions. Thus the Moose Suite is proposing a set of tools built-up around this same platform, all relying on a common

(cc) BY-SA

fixed core named FAMIX that is dedicated to the representation of object-oriented systems. Again, relying on a single common core can be seen as a limitation in some cases. On the contrary, the ARTIST intent is to propose a more open approach where technology-specific metamodels and related model discoverers (e.g., covering object-oriented technologies but not only) are used to ensure the required Model Discovery support as part of the overall process. If actually needed by the next steps in a given process, a more generic metamodel, e.g., UML to be able to reuse already existing capabilities, could be considered later on by transforming the specific models to such more generic ones.

Based on the widespread Eclipse platform and its Eclipse Modeling Framework [29], the MoDisco open source project [30] proposes a framework intended to facilitate the elaboration of MDRE solutions. It more particularly provides a set of reusable components for Model Discovery, including base (but complete) metamodels for different software technologies (Java, XML, JSP) and related model discoverers. It also provide some initial support for the generic discovery of standard models such as KDM [31] or UML [32] ones. Thus, and as initiated by one of the ARTIST partners, the MoDisco Model Discovery support is going to be reuse and extended intensively within the context of the project (cf. notably some components from the ARTIST Model Discovery Toolbox as presented in D8.2.1 and latest versions).

Taking into account the previously presented state-of-the-art and also our experiments of concretely applying MDRE in real projects (both from a research and industrial perspective), we believe MDRE has interesting capabilities to address efficiently reverse engineering problems as mentioned before. More specifically, the main challenges MDRE solutions must be able to overcome concerning the Model Discovery phase are the following:

- **To avoid information loss due to the heterogeneity of software systems.** To ensure the quality of the overall MDRE process, and thus validate its actual relevance; during the Model Discovery phase it is very important to be able to retrieve as much information as required from software systems that are often technically heterogeneous.
- **To manage scalability and complexity. Software systems are usually huge and complex systems.** Scalability of MDRE techniques, and more particularly of Model Discovery ones, must be improved to be able to load, query and transform in an effective way the very large models usually involved in MDRE processes.
- **To adapt/port existing solutions to different needs.** Many Model Discovery solutions are still technology- or scenario- dependent, meaning that they target a very concrete legacy technology or reverse engineering scenario. Progress must be done in the development of some generic or base Model Discovery solutions that, even if they still keep some specific parts, are more largely reusable in various contexts and for a minimized cost.

3 Requirements for Model Discovery in ARTIST

Based on the state-of-the-art in this area as presented before, on our past experience on MDRE in general (and Model Discovery in particular) and also on the ARTIST use cases descriptions, a set of requirements have been identified concerning the potential needs in terms of Model Discovery capabilities/features within the ARTIST methodology and related tooling support.

(cc) BY-SA

The requirements have been classified into four categories: (1) base requirements, which we consider crucial to perform properly the ARTIST methodology; (2) forward engineering related requirements, which are needed by the tasks of WP9; (3) UI requirements, which are aimed at making easier the use and understanding of the developed tooling; and (4) cross tasks related requirements, which are needed for other tasks/WPs within the ARTIST methodology.

In the remainder of the section, we introduce the set of identified requirements. Note that this initial requirement list should not be considered as a "contract" per se, since some of them are already (partially or totally) supported by existing tools/frameworks that may be reused and some others may not be (fully) addressed according to their assigned priority or constraints on efforts for the different involved partners.

3.1 Base Requirements

These requirements are considered as fundamental for the correct operation of the Model Discovery task of the ARTIST methodology (i.e. *MODELDISCO*).

- **Req1 - Support for raw model discovery**. The provided model discoverers have to allow obtaining sufficiently complete, if not full, low-level (or raw) representations of the artefacts as base models. Thus, it would be necessary to define a set of low-level or raw metamodels for the considered technologies. In the context of ARTIST, being able to get (at least) code models (e.g., Java or C#) and XML models is needed.
- **Req2 - Support for different software technologies.** Models have to be discovered out of artefacts from software systems relying on different (sometimes heterogeneous) technologies, languages or frameworks. In the context of ARTIST, the support for Java (i.e., J2EE/Java EE-based applications) and C# (.NET-based applications) is required.
- **Req3 - Generic approach for not specifically supported artefacts.** Even when a very specific technology support is not available, the required models have to be able to be discovered anyway. In these cases, alternative discoverers may be reused. For instance, in the context of ARTIST, the reverse engineering of configuration files may reuse the discoverer for XML-based files. In some cases, the discovery of models from not natively supported artefacts may need the development of more specific solutions.

3.2 WP9/Forward Engineering-related Requirements

These requirements can be perceived as extensions of the base ones in order to feed WP9 - Forward Engineering with the needed information (i.e., models) for enabling the rest of the ARTIST process. They also correspond to some more advance capabilities in terms of Model Discovery.

- **Req4 - Support for generic UML model discovery.** From the discovered raw models (cf. previous section), UML models have to be obtained via transformations implementing a direct/systematic generic mapping from the corresponding low-level metamodels (e.g., Java) to the UML metamodel.
- **Req5 - Support for both structural and behavioural discovery.** Also from the discovered raw models, more generic models have to be obtained (via transformation) to represent both structural and behavioural aspects of the reverse engineer artefacts, for instance respectively supporting the generation of UML Class and UML Activity models.

(cc) BY-SA

3.3 User Interface Requirements

These requirements are more directly related to the user experience when performing Model Discovery activities. They correspond to features expressed in terms of (graphical) interface to improve the usability and efficiency of the Model Discovery tooling support itself.

- **Req6 - Management of big complex models.** As models resulting from Model Discovery are often huge and complex, an appropriate viewer/navigator has to be provided in order to allow the user to browse them as efficiently as possible (e.g., using features such as searching, filtering, etc.).
- **Req7 - Easy access to the discovery features.** As software projects themselves can also be big and complex, an appropriate and simple way (ideally in a few clicks for instance) to launch the discoverers from the artefacts composing these projects has to be offered (e.g., using UI components such as pop-up windows, wizards, etc.).

3.4 Cross-WPs/Activities Requirements

The main purpose of this category is to introduce requirements relevant not only in the context of Model Discovery but also more generally in other activities from the ARTIST methodology and processes.

- **Req8 - Support for traceability between initial artefacts and corresponding models.** A Model Discovery process (and the ARTIST process in general) deals with several artefacts and models relating together. Thus, being able to keep track of the handled elements from the initial software artefacts to the migrated ones passing by the different intermediate models is an important feature.

4 Support for Model Discovery in ARTIST

The objective of this section is to introduce the Model Discovery support as actually thought and provided within the context of the ARTIST project and related overall methodology. First, the main principles and concepts associated with Model Discovery are presented to the reader. Then, it is described how this methodology for Model Discovery is (or is planned to be) technically supported by the current version of the ARTIST Model Discovery Toolbox.

4.1 Overall Methodology for Model Discovery

We believe a satisfying Model Discovery support must provide the following global characteristics in order to fulfil the identified requirements:

- **Genericity** based on technology-independent standards (i.e., metamodels such as UML) and customizable model-based components where specific technology support is ensured by additional dedicated features that can be plugged into the more generic ones (cf. **Req3**, **Req4**);
- **Extensibility/Coverage** relying on a base support that is potentially extensible in order to cover different software technologies according to the possible heterogeneity of the considered existing software system (cf. **Req1**, **Req2**, **Req5**);
- Direct **(re)use** and **integration** of both the directly provided components and possibly external ones, but also of all the discovered models (e.g., sharing with other steps of the overall process) (cf. **Req6**, **Req7**, **Req8**);
- Facilitated **automation** of (at least parts of) the Model Discovery processes thanks to the already available MDE techniques, notably by chaining predefined sets of discovery components with model transformations, etc. (cf. **Req7**).

(cc) BY-SA

To achieve them, the base idea is to follow the strategy of switching very early from the heterogeneous world of software systems (in terms of used technologies, languages, frameworks, etc.) to the more homogeneous world of models. Thus, we can directly benefit as soon as possible from the interesting properties of MDE and the full set of already available technologies (e.g., for manipulating the models discovered from the software systems).

The main principle is to quickly get initial models representing the artefacts of the software system without losing any information required for the next steps of the process. These “raw" models are sufficiently accurate to be used as a starting point of the considered MDRE scenario (cf. next Model Understanding task for instance), and do not represent any real increase of the abstraction or detail levels. Thus, they can be seen as (full or partial) abstract syntax graphs that bridge the syntactic gap between the worlds (i.e., technical spaces [33]) of source code, configuration files, etc. and the world of models. From this point on, any reverse engineering task to be performed on the system can be done with the same result using these models as a valid input representation. Therefore, we have reduced the heterogeneity of the reverse engineering process to a modelling problem.

Now going more to the details, to be able to represent the software system without losing any of the required information, the metamodels employed at this stage can be metamodels of a low abstraction level and that closely resemble the source technology such as a general programming language (e.g., Java, C#, C++), a Domain Specific Language or DSL (e.g., SQL, HTML), an exchange format (e.g., XML, JSON), a file format (e.g., Microsoft Word or Excel), etc.

The metamodel-driven software components allowing generating these raw models are called model discoverers, where each discoverer targets a specific (software) technology. The overall principle of a model discoverer is shown on Figure 1.

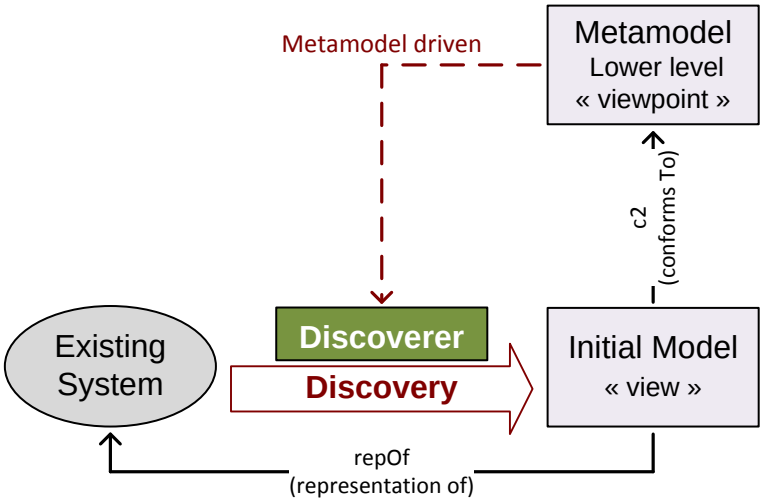

Figure 1: The Model Discovery phase

First the metamodel (defining the viewpoint the system will be observed from) has to be designed, depending on the kind of legacy system that is concerned. For some well-known technologies like Java/JEE, appropriate metamodels can already be available (e.g., see the ones provided via the ARTIST Model Discovery Toolbox). This metamodel is the main factor ensuring the quality and completeness of the discovery phase in charge of creating instances of this metamodel from the legacy system. The model resulting from the execution of the built discoverer will conform to this base metamodel.

(cc) BY-SA

Note that even if some discoverer models were created during the development process, they could already be outdated compared to the actual state of the software system. In that case, model discovery can provide a useful complementary support to validate the relevance of these models. Thus, discovered models can be used to study how the system has evolved (e.g., detecting the elements that changed, such as variable or methods names in the source code).

As described by the ARTIST Methodology (cf. section 5.1 of ARTIST D6.2.2), Model Discovery can start with the pre-task of classifying the software artefacts composing the system to be treated (cf. ARTIST Task *MODELDISCO.A1*). Thus, a better vision of this system can be obtained which can notably guide the definition of the actual model discovery process. In particular, a taxonomy (cf. ARTIST D8.1 Taxonomy of Legacy Artefacts) has been defined to facilitate a proper analysis of the software system to be reverse engineered. The taxonomy includes several commonly shared dimensions and provides recommendations to identify their potential impact notably on Model Discovery activities.

Depending on the cases, that is, the software technology to be considered as input for the model discovery, the process itself can be split up into two complementary steps (cf. Figure 2):

- **Injection**. Focusing on bridging between the technical spaces of the software system and the technical space of MDE (a.k.a. *modelware*), e.g., using parsers, APIs or other techniques to access the content of software artefact(s) and then create the low-level model accordingly (cf. ARTIST Task *MODELDISCO.A2*);
- **Transformation.** An additional syntactic (or structural) mapping inside the MDE technical space to produce the actual initial low-level model from the result (i.e., a model) of the previous injection step, if required (cf. optional ARTIST Task *MODELDISCO.A3*).

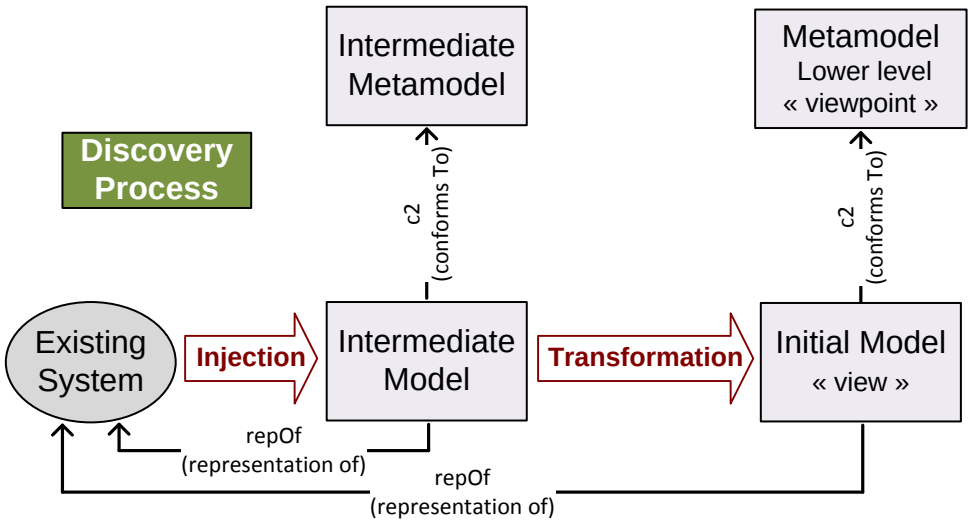

Figure 2: Model Discovery - A possibly two-step approach

These two steps are not necessarily atomic operations performed at once and can rather be composed of several sub-operations possibly iterative and interactive, notably chains of model transformations in the second (optional) step. The length of this chain may depend on the complexity of the discovery process and on the availability of some intermediate metamodels that can be used to simplify it.

For instance, considering the case of discovering the model corresponding to a set of source code files in Java, the discovery process implementation can vary:

(cc) BY-SA

- The **two-step approach** can be strictly applied: a generic (intermediate) model of the abstract syntax tree can be obtained first from the program (based on a generic metamodel such as OMG ASTM [34], e.g., in [35]) and then transformed into a language-specific (low level) model (i.e., a Java model);
- A more **direct approach** can be considered: visit the abstract syntax tree of the program (e.g., using a dedicated API or the result of a previous parsing) in order to build directly the language-specific (low level) model (i.e., the Java model), the transformation step being integrated in the injection one.

There are different arguments to help deciding between these two implementation options, the use of the ARTIST Taxonomy of Legacy Artefacts (cf. ARTIST D8.1) being of some help in that decision process. In the first case, considering such a two-step process usually allows reducing the complexity of each individual step (cf. the well-known "divide and conquer" overall approach) and thus can facilitate the discoverer's global implementation. The first part of the discoverer (the injector) becomes generic and can be directly reused by similar model discoverers. Nevertheless, when these benefits are not so relevant for the reverse engineering scenarios interesting to the user, one can decide to go for the second option which involves less technological complexity (i.e., less number of artefacts to think about) and offers sometimes better performance. In any case, the initial models obtained at the end of both alternatives have to represent the same system at the same (expected) level of details.

4.2 Concrete Technical Support for Model Discovery

The actual Model Discovery support in ARTIST is provided via the so-called ARTIST Model Discovery Toolbox (cf. ARTIST deliverable D8.2.1 and latest versions for more details about this toolbox). It is basically constituted of a set of reusable components intended to cover both the Java/JEE and C#/.NET software technologies (partially of course, but nevertheless sufficiently for the context of ARTIST). The next paragraphs presents how these various components follow the overall methodology for Model Discovery as introduced before in this document. However, in order to avoid unnecessary information duplication, all technical and implementation details can be directly found from corresponding ARTIST deliverable D8.2.1.

First of all, the different provided Model Discovery components match the main characteristics (as generalized from the expressed requirements) that have been enumerated previously:

- Concerning **genericity**, some general standards from the OMG are supported and notably UML. Thus, UML (Class or Activity) models can be discovered from Java source code. This can be performed directly from a previously obtained Java models or indirectly via an intermediate KDM models (which is another OMG standard). Moreover, a generic XML support is offered in order to allow reverse engineering from all XML documents, independently from the XML schemas they conform to. Such a base support for XML has notably been reused in the context of several JEE frameworks configuration files or Microsoft SharePoint data files.
- Software technology **coverage** is addressed, obviously partially, via the already provided support for different base technologies (currently Java and XML, but this will be extended for C# and some related .NET technologies in the second phase of ARTIST). Thus, it is possible to get both low-level/raw models but also more generic ones from any Java project (cf. previous point). In addition, the offered support is covering both structural and behavioural aspects. In terms of **extensibility**, the current coverage can be extended either by developing and integrating brand new model discoverers or by relying on (and so extending accordingly) other already existing ones.

(cc) BY-SA

- Then, concerning **reuse** and/or **integration**, the platforms which these components are based on contribute to facilitate the process. For instance, the open source Eclipse/EMF environment allows easily building and packaging new metamodels and associated model discoverers, model transformations, etc., and so does Sparx Enterprise Architect but relying on their own internal (proprietary) parsers and representations. In any case, the overall integration is ensured via the use of a common representation format for interchanging models between the two worlds, namely UML models serialized in EMF XMI.
- Finally, **automation** of the Model Discovery processes is made possible in different ways. The execution of model discoverers is most of the time fully automated and simply requires some initial parameters from the users. The provided user interface (either in the Eclipse workbench or Sparx Enterprise Architect) allows easily setting these parameters and then launching the automated discovery process. The automation of more elaborated model discovery processes can be achieved from the host platform: e.g., in the Eclipse environment via either the MoDisco workflow facilities or directly via Java code; a similar kind of support being available in the Sparx Enterprise Architect context.

As far as the structure of the Model Discovery process is concerned, the current version of the ARTIST Model Discovery Toolbox provides components implementing both approaches (either the one-step or the two-step one, cf. previous subsection):

- Sparx Enterprise Architect-based components (mostly for UML model discovery so far) generally follow a one-step approach as they allow directly obtaining the initial UML models from the source project without any intermediate step or model. However, to influence the mapping between the source code and the UML model, it is possible to define an nBNF grammar (an extended form of a BNF grammar). The nBNF grammar contains information about what syntactic elements are mapped to corresponding UML constructs.
- Eclipse/EMF- (and MoDisco-) based components are more heterogeneous from this perspective. Some of them are following the one-step direct approach (such as the ones for raw Java or XML) while some others (notably the more generic ones targeting UML) actually rely on these first ones: e.g., the UML Class discovery from Java is performed using the Java model resulting from the Java model discoverer (either directly or via another KDM intermediate step).

Finally, related to the support still currently missing in the ARTIST Model Discovery Toolbox, the main new components to be elaborated would concern the support for the C# language and more generally (parts of) the .NET technology.

Thus, in order to come to equal terms with the Java support in ARTIST, the ideal situation would be to provide a C# Model Discoverer comparable to the already existing Java Model Discoverer. This would allow having a complete one-to-one mapping of the source code towards a C# Model. For instance, the current support for model discovery from C# code in Enterprise Architect is not covering external class references. This is a real limitation since many applications based on the .NET framework (or on frameworks on top of it) use such external (platform) classes and/or libraries. Also, the integration effort between Enterprise Architect and Eclipse has to be continued, notably to better relate the C# source code (e.g., in Enterprise Architect) with corresponding models (e.g., in Eclipse). An additional feature would be the discovery support for the C#/.NET *attributes* (similar to the Java *annotations*), for instance using (a collection of) profiles. Useful applications could concern the .NET framework itself, the Entity Framework, the SharePoint platform and so on.

(cc) BY-SA

5 Model Discovery Experiments in ARTIST

This section briefly describes the first experiments that have been performed around Model Discovery during the first half of the project, preparing the field for more extended applications on the ARTIST use cases in the second half of the project. It has been kept voluntarily short in order to do not unnecessarily repeat content that will be elaborated with more details in future deliverables (e.g. describing actual use case implementations). The main purpose of this section is to simply show how the Model Discovery task of the ARTIST methodology has already been applied to various material and scenarios (cf. previous section for more information on the Model Discovery methodology itself). It also gives a practical taste on how the first set of components from the ARTIST Model Discovery Toolbox have been concretely used so far (cf. ARTIST D8.2.1 for more details on these components themselves).

5.1 Experimenting with the Java PetStore example

The Java PetStore[1] reference application has been introduced early in the ARTIST project to enable performing initial experiments within a more controlled environment. It is considered as a test scenario in the context of the ARTIST project. For that reason, the Java PetStore application was first migrated manually to the Google App Engine[2]. Then, the migration of the Java PetStore persistency layer was automated via a first version of a generic transformation chain implemented at that time. Thus a reverse engineering phase has been applied first, before the "cloudified" application code to be generated in a forward engineering phase from the discovered and refined domain models.

5.1.1 Overview of the Model Discovery process

In the performed reverse engineering phase, initial low-level models have been discovered from the Java PetStore application. In fact, the focus was set on structural aspects as the migration of the persistency layer was the main objective. Furthermore, as two different versions of the Java PetStore have been considered – one version is based on J2EE 1.3 while the other version is based on Java EE 6 – the models resulting from the respective Model Discovery step are slightly different (as relying on different Java platforms). For instance, considering the implementation of the persistence layer, the Java PetStore version based on Java EE 6 extensively make use of the Java annotation mechanism to express the application entities to be persisted. This is not the case for the Java PetStore version based on J2EE 1.3[3]. In the latter case, additional descriptor files (i.e., XML-based configuration files) are employed in addition to the Java source code to describe these persistent entities.

Independently from the different Java PetStore versions, the two-step approach (cf. Figure 2) has been generally applied in order to get base UML representations from the input software artefacts. These UML Class models are then considered as input for Model Understanding. In this particular context, the focus has been set on generating a UML Class model (a domain model) that covers only the essence of the PetStore domain, i.e., 12 classes with properties and relationships in-between, from the initially discovered base UML Class model containing more than 100 classes (cf. ARTIST D8.3 for more details on these Model Understanding aspects).

---

[1] The Java PetStore was originally introduced by Sun in 2001 (http://www.oracle.com/technetwork/java/javaee/tech/index.html). It is a simple Web-based reference application based on Java Enterprise Edition. Today, several versions of the Java PetStore are available.

[2] The rationale behind selecting the Google App Engine rather than another provider is basically explained by its native support for applications based on the Java Enterprise Edition.

[3] Java annotations were introduced after the release of J2EE 1.3.

(cc) BY-SA

5.1.2 Model Discovery from the J2EE 1.3 version

Figure 3 gives an overview of the conducted experiment within the context of the Java PetStore implementation that is based on the J2EE 1.3 platform.

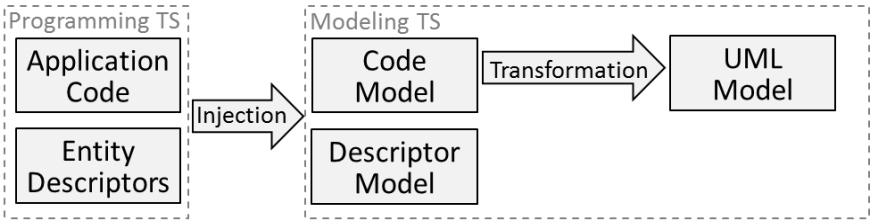

Figure 3 Model Discovery experiment for the Java PetStore based on J2EE 1.3

The *Application Code* and respective *Entity Descriptors* are first injected as models, thanks respectively to the Java Code Model Discoverer and Generic XML Model Discoverer and Aggregator from the ARTIST Model Discovery Toolbox. By this step, a complete model of the Java code (including a fine-grained description of all Java statement, e.g., variables, loops, conditions and assignments) is generated as well as a representation (as a single XML model) of the associated descriptor files. Relying on the initially discovered *Code Model*, a base UML (Class) model is generated by via the Generic UML Class Model Discoverer (reusing its provided *Java-to-UMLClass* transformation). The *UML Model* discovered at this point basically represents the injected application code. Both the *UML Model* and the *Descriptor Model* serve as basis for further "domain" analysis throughout Model Understanding (as discussed in ARTIST D8.3).

5.1.3 Model Discovery from the Java EE 6 version

Quite similarly to the content of the previous subsection, Figure 4 shows an overview of the conducted experiment with the Java PetStore that is based on the Java EE 6 platform this time.

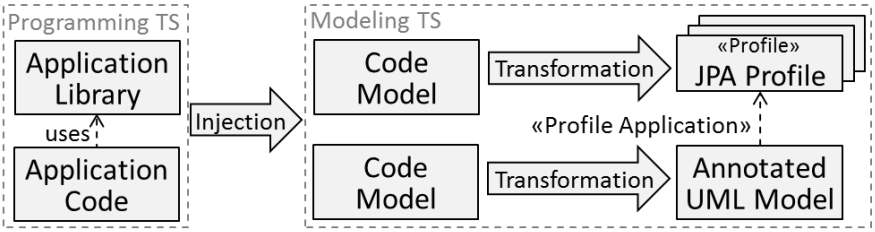

Figure 4 Model Discovery experiment for the Java PetStore based on Java EE 6

The main difference with the experiment previously presented is that the use of Java annotations now makes the source code sufficient to obtain the domain information later required for Model Understanding. Thus, model discovery from associated descriptor files is no longer needed in this particular scenario. However, in addition to the Java code model and related base UML Class model as in the previous case, a UML Profile is here necessary to provide the corresponding annotations in terms of stereotypes (e.g. for representing framework- or library-specific information such as persistence classes, cf. explanations just after). To generate the required UML Profile – in the present case, the focus is set on discovering a UML Profile for the Java Persistence API, though the used Java PetStore version actually relies on annotations from more than 10 different *Application Libraries* – the Generic UML Profile Discoverer is applied. Moreover, the enhanced version of this model discoverer is also used for directly obtaining the *Annotated UML Model* (with the discovered profile applied on).

(cc) BY-SA

5.2 Experimenting with the DEWS use case

5.2.1 Overview of Model Discovery activities

So far, the DEWS use case makes mainly use of the Eclipse-based components from the ARTIST Model Discovery Toolbox. More particularly, in the context of Model Discovery, it reuses MoDisco model discoverers and related ATL model-to-model transformations when needed. While such model discoverers are used to obtain initial models from the DEWS software system (e.g. from its Java source code and associated configuration files), model transformations are then used to refine them and/or later during the Model Understanding step. For instance, such more advanced transformations are used to perform (semi-)automated component detection (cf. ARTIST D8.3 for more details on Model Understanding activities).

5.2.2 Initial UML and XML Model Discovery

Several model discoverers from the ARTIST Model Discovery Toolbox are practically used and applied to obtain the set of initial models required to start working with the DEWS use case in this modernisation context.

Thus, the Java Code Model Discoverer is first used to obtain complete Java Models from the Java source code of the DEWS application. Then, using the associated Generic UML Class Model Discoverer (implementing via a direct transformation from Java to UML Class), corresponding UML Class models are produced from the previously discovered Java models. It is important to note that, since the use case is technically split into three different projects (i.e., *core*, *swe* and *gitews*), the initial UML discovery process is launched for each individual project. The result is the generation of three different UML Class models which will have to be combined into one single UML Class model (cf. next subsection).

Also, in addition to discovery from Java source code, the generic XML Model Discoverer is also used to get an XML model from the different *plugin.xml* project descriptor file of one of the project composing the DEWS application (describing the required project dependencies).

5.2.3 Further Treatment by Transformation

As explained before, three different UML Class models (one per project from the DEWS use case) have been generated using the Generic UML Class Model Discoverer. However, elements from one of the DEWS project are sometimes related to elements from another one. Moreover, from a deployment and extensibility perspective, it is easier to deal with one single UML Class model in the remainder of the process rather than with multiple (even if interconnected) ones. Thus, in order to facilitate Model Understanding coming right after as well as the later modernisation process (i.e., forward engineering), additional model transformations have been implemented to perform this merging step. In particular, several model-to-model transformations have been defined to obtain higher-level models from the Data and RCP (Rich Client Platform) components of the application. These transformations are mainly PSM-to-PIM ones that produce UML models profiled with stereotypes representing information about some identified applied patterns, e.g. *Observer* such as used in the DEWS use case (cf. D8.3 for more details about further Model Understanding activities).

(cc) BY-SA

5.3 Experimenting with the LoB use case

5.3.1 Generic UML Class Diagram Discovery

The LoB/SpikesTogether use case first required using of the Enterprise Architect (*EA*)-based components from the ARTIST Model Discovery Toolbox to reverse engineer the *SpikesTogether* source code and obtain corresponding UML Class models. These UML Class models have been explored through *EA* on a package-per-package basis.

To use the Eclipse-based ARTIST toolset, these UML Class models have then been exported to a XMI-compliant UML format. In *EA*, it is possible to export these models to an *EA* variant of UML-XMI. However, the current version of this exporter still gives some warnings/errors since the UML-XMI in *EA* deviates from the UML-XMI format used in Eclipse. To solve this problem, an intermediate tool, the *EnArXMISpy*, was used to export the data to yet another tool developed by Sparx in charge of completely rebuilding the UML inside the Eclipse environment, thus obtaining a finally valid XMI-compliant UML model. This feedback has been provided to Sparx in order to improve the support for Eclipse XMI-compliant UML models in *EA* in next versions.

5.3.2 SharePoint Data Model Discovery

In this case, the Generic XML Model Discoverer as available from the ARTIST Model Discovery Toolbox has been used in order to obtain XML models from various SharePoint related XML files contained in the *SpikesTogether* project (i.e., *listinstance*, *contenttype* and *sitecolumns* files). These XML files contain *SpikesTogether* specific extensions on the default SharePoint data model. A model transformation has then been defined for each kind of XML model, thus allowing transforming each one of them into other models conforming to more specific metamodels. Next, another set of transformations are in charge of merging the previous models into a single one, thus providing an integrated data model for the whole application. At last, it provides a base data model to be used as input for the Model Understanding and/or further forward engineering activities.

Following a similar approach, models have also been discovered from the so-called Resource Files (files with *resx* extension). This was needed in the transformation of the SharePoint data model but can also be used standalone since the resource files are used elsewhere in C#/.NET applications.

5.3.3 Package Dependency Discovery

The *EA Package Dependency Generator*, a component to extract and visualize dependencies between the different packages inside *SpikesTogether* has also been used as an initial input for obtaining a Component Model. The component model was then transformed, manually at the moment, into a component model within the Eclipse environment (cf. Model Understanding). Even if other tools (such as *Altova UModel* and *Visual Studio*) have been used in this context to complement the information given by the *Package Dependency Generator*, it could possibly be added to next version of the ARTIST Model Discovery Toolbox.

(cc) BY-SA

6 Conclusions

Reverse Engineering is the very first phase of any software migration process, such as the ones supported by the ARTIST project and its model-based migration methodology (and corresponding tooling support). More particularly, in Model Driven Reverse Engineering (MDRE) processes, an initial step called Model Discovery is first necessary. It is in charge of (automatically) obtaining the needed base models from the software artefacts composing the existing system to be later migrated (to the Cloud). This deliverable has been focusing on the methodology and techniques for Model Discovery and its concrete implications within the context of the ARTIST project. Thus the overall context of Model Discovery has been first introduced, by notably performing a review of the state-of-the-art in this area. Based on this, the actual need for Model Discovery has been motivated considering also the requirements as expressed by the different ARTIST partners involved in such Model Discovery activities. According to these requirements, a more detailed methodology and its related supporting tooling have also been described (cf. also ARTIST deliverable D8.2.1 and latest versions). Finally, some concrete experiments around Model Discovery realized during the first year and a half of the ARTIST project have been summarized.

While the current state-of-the-art and proposed methodology for Model Discovery can be considered as already stable and finalized at this point in time, the actual support for Model Discovery as implemented within ARTIST is still under evolution. As a consequence, its development will continue during the second year and a half of the project. Thus, two more versions of the ARTIST Model Discovery Toolbox will be produced and delivered respectively at ARTIST M24 (D8.2.2) and M30 (D8.2.3). They will notably complement the current tooling support as already provided by the initial version of the toolbox already delivered at ARTIST M12 (D8.2.1)

As further work in terms of Model Discovery tooling, the focus will be mainly put on trying to fulfil a majority of the requirements that have been identified and expressed in this document. This will notably involve some more implementation/development effort. In this context, the already submitted Taxonomy of Legacy Artefacts (cf. ARTIST deliverable D8.1) will be used as a way to provide valuable information regarding the more suitable solutions to implement the missing support. It is also important to note that extra-requirements could be added throughout the development of the project if really needed, when applying the various Model Discovery components to the ARTIST use cases for instance. The development of this new support (e.g., for the C#/.NET world more particularly) will not only benefit to Model Discovery itself but also to all potential users of the ARTIST MDRE approach (cf. also the Model Understanding step as depicted in ARTIST deliverable D8.3).

(cc) BY-SA

7 References

[1] M. Rekoff Jr., "On reverse engineering", *IEEE Transaction on Systems, Man and Cybernetics* 15 (1985) pp. 13-17.

[2] E. J. Chikofsky, J. H. Cross II, "Reverse engineering and design recovery: A taxonomy", *IEEE Software* 7 (1990) pp. 13-17.

[3] R. Kazman, S.S. Woods, S.J. Carriere: "Requirements for integrating software architecture and reengineering models: Corum ii". in: *Proceedings of the Working Conference on Reverse Engineering*, Honolulu, Hawaii, U.S.A., October 12-14 (1998) pp. 154-163

[4] G. Canfora, M. Di Penta, L. Cerulo, "Achievements and challenges in software reverse engineering", *Communications of the ACM* 54 (2011) pp 142-151.

[5] H. Chapman, P. A. Hall, "Software Reuse and Reverse Engineering in Practice", Chapman and Hall, Ltd. London, UK, 1992.

[6] M. L. Nelson, "A survey of reverse engineering and program comprehension", in: *ODU CS 551 Software Engineering Survey*.

[7] G. Canfora, A. Cimitile, M. Munro, "Re2: Reverse-engineering and reuse re-engineering", *Journal of Software Maintenance: Research and Practice* 6 (1994) pp. 53-72.

[8] W. Premerlani, M. Blaha, "An approach for reverse engineering of relational databases", in: *Proceedings of the Working Conference on Reverse Engineering*, Baltimore, Maryland, USA, May 21-23 (1993) pp. 151-160.

[9] E. Eilam, "Reversing: Secrets of Reverse Engineering", Wiley Publishing, 2005.

[10] S. Kent, "Model driven engineering", in: *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 286-298.

[11] S. Rugaber, K. Stirewalt, "Model driven reverse engineering", *IEEE Software* 21 (2004) pp. 45-53.

[12] J.-M. Favre, "Foundations of model (driven) (reverse) engineering : Models - episode i: Stories of the fidus papyrus and of the solarus", in: J. Bezivin, R. Heckel (Eds.), *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, IBFI, Dagstuhl, Germany, 2005.

[13] OMG Architecture Driven Modernization (ADM), http://adm.omg.org

[14] L. Favre, "Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution", IGI Global - Premier Reference Source, 2010.

[15] T. Girba, *The Moose Book*, Self Published, 2010.

[16] A. van Deursen, E. Visser, J. Warmer, Model-driven software evolution, in: *1st Workshop on Model-Driven Software Evolution* (MoDSE 2007), Amsterdam, The Netherlands.

[17] H. M. Sneed, "Migration of procedurally oriented Cobol programs in an object-oriented architecture", in: *International Conference on Software Maintenance* (ICSM 1992), Orlando, Florida, U.S.A.

[18] H. M. Sneed, "Migrating from Cobol to Java", in: *International Conference on Software Maintenance* (ICSM 2010), Timisoara, Romania.

[19] F. Barbier, S. Eveillard, K. Youbi, E. Cariou, "Model-driven reverse engineering of Cobol-based applications", in: R. Vogel (Ed.), *Proceedings of the Tools and Consultancy Track. 5th European Conference on MDA Foundations and Applications* (ECMDA-FA 2009). Enschede, The Netherlands, June 23 - 26, 2009, pp. 36-51.

[20] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, J.-M. Jezequel, "Model-driven engineering for software migration in a large industrial context", in: *Model Driven Engineering Languages and Systems*, volume 4735 of Lecture Notes in Computer Science, 2007, pp. 482-497.

[21] L.F. Andrade, J. Gouveia, M. Antunes, M. El-Ramly, G. Koutsoukos: Forms2Net – "Migrating Oracle forms to Microsoft.NET." In: *Generative and Transformational*

(cc) BY-SA

Techniques in Software Engineering, pp. 261–277 (2006)

[22] Sparx Enterprise Architect, http://www.sparxsystems.com/products/ea/

[23] R. Ferenc, A. Beszedes, M. Tarkiainen, T. Gyimothy, "Columbus - reverse engineering tool and schema for C++", in: *International Conference on Software Maintenance* (ICSM 2002), IEEE Computer Society, 2002, pp. 172-181.

[24] F. Heidenreich, J. Johannes, M. Seifert, C. Wende, "Closing the gap between modelling and Java", in: *International Conference on Software Language Engineering* (SLE'09), Springer, 2009, pp. 374-383.

[25] SpoonEMF2, http://soft.vub.ac.be/soft/research/mdd/spoonemf2

[26] L. C. Briand, Y. Labiche, J. Leduc, "Toward the reverse engineering of UML sequence diagrams for distributed Java software", *IEEE Transaction on Software Engineering* 32 (2006) 642-663.

[27] W. Sun, S. Li, D. Zhang, Y. Yan, "A model-driven reverse engineering approach for semantic Web services composition", in: *Proceedings of the 2009 WRI World Congress on Software Engineering* (WCSE 2009). Denver, Colorado, USA, October 4, 2009, volume 3, IEEE Computer Society Washington, DC, USA, 2009, pp. 101-105.

[28] J.L. Cánovas Izquierdo, J. Cabot: "Discovering Implicit Schemas in JSON Data". In: *International Conference on Web Engineering* (ICWE'13) Volume 7977., LNCS (2013) pp. 68-83.

[29] Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf

[30] H. Bruneliere, J. Cabot, F. Jouault, F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering", in: *Proceedings of the ASE '10 IEEE/ACM International Conference on Automated Software Engineering*. Antwerp, Belgium, September 20 - 24, 2010, ACM New York, NY, USA, 2010, pp. 173-174.

[31] OMG Knowledge Discovery Metamodel (KDM), http://www.omg.org/technology/kdm/

[32] OMG Unified Modeling Language (UML), www.uml.org

[33] I. Kurtev, J. Bézivin, M. Aksit: "Technological Spaces: an Initial Appraisal". In: *International Symposium on Distributed Objects and Applications* (2002) pp. 1-6

[34] OMG Abstract Syntax Tree Metamodel (ASTM), http://www.omg.org/spec/ASTM/

[35] J. L. Canovas Izquierdo, J. Garcia Molina, "An architecture-driven modernization tool for calculating metrics", *IEEE Software* 27 (2010) pp. 37-43.

(cc) BY-SA