ARTIST

FP7-317859

Advanced software-based seRvice provisioning and migraTIon of legacy Software

Deliverable D8.1

Taxonomy of legacy artefacts

Editor(s):	Hugo Bruneliere, Javier Cánovas
Responsible Partner:	INRIA
Status-Version:	Final version - v1.2
Date:	25/09/2013
Distribution level (CO, PU):	PU

Project Number:	FP7-317859
Project Title:	ARTIST

Title of Deliverable:	Taxonomy of legacy artefacts
Due Date of Delivery to the EC:	30/09/2013

Work package responsible for the Deliverable:	WP8 - Legacy Product Analysis by Reverse Engineering
Editor(s):	Inria (Hugo Bruneliere, Javier Cánovas)
Contributor(s):	Inria (Hugo Bruneliere, Javier Cánovas, Guillaume Doux) Fraunhofer (Oliver Strauß) TECNALIA (Leire Orue-Echevarria) From case studies: ATOS (Yosu Gorroñogoitia) ENG (Stefania D'Agostini), SPIKES (Bram Pellens) ATC (Ilias Spais)
Reviewer(s):	ATC (Ilias Spais)
Approved by:	All Partners
Recommended/mandatory readers:	WP5, WP9

Abstract:	This document proposes a detailed classification of the main different types of legacy artefacts according to their corresponding characteristics and specific properties. The purpose of this taxonomy is to be then used as a base to guide the design and implementation of the Model Discovery technologies (cf. D8.2.x).
Keyword List:	Taxonomy, Legacy Artefact, Model Discovery

Document Description

Document Revision History

Version	*Date*	*Modifications Introduced*	
		Modification Reason	*Modified by*
v0.1	09/11/12	Initialization of the document with a first ToC and some example sections	Javier Cánovas & Hugo Bruneliere, Inria
v0.2	26/11/12	First draft version including some initial content to Sections 3, 4 & 5	Guillaume Doux & Hugo Bruneliere, Inria
v0.3	07/12/12	Contributions from Fraunhofer & TECNALIA	Oliver Strauss (Fraunhofer) & Leire Orue-Echevarria Arrieta (TECNALIA)
v0.4	20/12/12	Second draft version with upgraded content (notably in Section 4)	Hugo Bruneliere, Inria
v0.5	17/01/13	Integration of comments from Jordi Cabot, Inria	Hugo Bruneliere & Javier Canovas, Inria
v0.5.1	12/02/13	Adding of a new dimension as proposed by TU Wien	Hugo Bruneliere, Inria
v0.6	15/05/13	Integration of comments from TECNALIA	Hugo Bruneliere, Inria
v0.7	15/07/13	Third draft version with upgraded content all along the document (e.g. in Section 6)	Javier Cánovas, Inria
v0.8	25/07/13	Executive summary + upgrades in the different sections	Hugo Bruneliere, Inria
v0.9	14/08/13	Fourth draft and final input from Inria, waiting input from partners to compile the last version	Javier Cánovas. Inria
v1.0	10/09/13	First tentative final version integrating contributions from partners.	Javier Cánovas, Hugo Bruneliere and Guillaume Doux. Inria
v1.1	24/09/2013	Internal Review	Ilias Spais (ATC)

v1.2	25/09/2013	Final version including comments from the internal review	Javier Cánovas

Table of Contents

Table of Figures

Table of Tables

Terms and abbreviations

FE	Forward Engineering
MDE	Model Driven Engineering
MDRE	Model Driven Reverse Engineering
RE	Reverse Engineering
TS	Technical Space

Executive Summary

The ARTIST project is about proposing concrete solutions to the problem of migrating legacy software to Cloud environments. In this context, Reverse Engineering (RE), as the process of obtaining different relevant representations of the legacy system, is the first action to be performed in order to accomplish the actual migration. Promoted by the ARTIST approach, Model Driven Reverse Engineering (MDRE) is the application of Model Driven Engineering (MDE) techniques to the realization of RE. Thus, the produced representations of legacy systems are actually models, which conforms to different metamodels and have different levels of abstraction depending on the needs.

In order to get the required models, the key inputs are the very often numerous and heterogeneous legacy artefacts composing a given legacy software. Thus, it is fundamental to have a correct knowledge of the concerned (type of) legacy artefacts in terms of properties and characteristics. These different artefacts are consumed by so-called *Model Discoverers* (cf. D8.2.1 – "Components for Model Discovery") producing the initial models out of them. Thus, having a global classification of the different kinds of legacy artefacts that can be encountered can be very helpful in the context of RE.

The present document is providing a general taxonomy for classifying legacy artefacts according to several complementary dimensions. The identified dimensions are currently the following:

- Technological Content – *Technical Spaces*
- Origin – *Manual vs. Generated*
- Purpose – *Code vs. Documentation*
- Consumer – *System vs. End User*
- Organization – *Structured vs. Unstructured*
- Nature – *Static vs. Dynamic*
- Size – *Small vs. Large*
- Opacity – *White Box vs. Black Box*
- Architectural Layer – *Conceptual Content*
- Environment – *Supporting Tooling*

This deliverable is presenting in detail each one of these dimensions, notably explaining their meaning and relevance in a MDRE context. The taxonomy is also accompanied by a simple classification framework, based on an easy-to-fill template, thus allowing its application to a given legacy software. As real life illustrations, this taxonomy has already been used on the four different ARTIST use cases, thus already offering a solid panel of legacy systems in terms of heterogeneity.

The benefits brought by the use of such a taxonomy can vary depending on the targeted objective. For software architects, it can allow having a more accurate vision of the current state of their legacy system and thus facilitating its global (technical) understanding. For modernization engineers, it can provide a precious help in the process of identifying or selecting the most appropriate model discoverers in their particular context. For Model Discovery experts, it can give detailed indications guiding the model discoverer design and implementation. More generally, we also foresee other potential benefits in other areas (e.g., in the actual migration, to specialize transformations according to the nature of the software artefact; or in Forward Engineering, to study the target system of the migration).

1 Introduction

The Model Driven Engineering (MDE) paradigm emphasizes the use of models to raise the level of abstraction and automation in the development of software [1]. Abstraction is a primary technique by which human minds cope with complexity, whereas automation is the most effective method for boosting productivity and quality.

In MDE, the approach used to increase the level of abstraction is that of defining modelling languages whose concepts closely reflect the concepts of the problem domain, thus facilitating understanding and hiding implementation technologies. On the other hand, automation allows models expressed in high-level abstraction to be transformed into equivalent computer programs or other models more suitable for design analysis. The research community has embraced MDE, partly because they see an opportunity to provide significant improvements in the development of software.

There are generally three major applications of MDE [2]: (1) (model driven) forward engineering where systems are created from models, (2) (model driven) reverse engineering where models are extracted from systems and (3) system-model coexistence in time and synchronization. This document addresses specificities of Model Driven Reverse Engineering (MDRE).

A MDRE process is mainly composed of two steps [3]: (1) discovery of low-level models from the existing system artefacts and (2) transforming the extracted models into higher-level ones, which will be used in the rest of the process. As a mandatory first step, model discovery is therefore fundamental in MDRE.

Existing software systems are composed of a (large) number of heterogeneous software artefacts taking different forms: source code, configuration files, raw data, documentation and many others. Thus, it is crucial for a MDRE process (where reverse engineering techniques are applied) to deal with the different software artefacts composing the legacy software system.

In order to better identify the different types of legacy artefacts composing software systems as well as their potential impact on the MDRE process, this document describes a taxonomy and related methodology helping to classify them.

1.1 Document structure

- **Section 2.** This section motivates the need of a taxonomy of legacy artefacts in the context of MDRE, thus giving an overall view of the main advantages it provides.
- **Section 3.** The taxonomy along with its dimensions are presented in this section.
- **Section 4.** This section presents a classification framework to help developers to apply the taxonomy.
- **Section 5.** This section includes four case studies where the taxonomy has been applied, thus illustrating its application in within the context of the project.
- **Section 6.** This section discusses some lessons learned after applying the taxonomy to the previously commented case studies.
- **Section 7.** The report finalizes with conclusions and some further work in the context of the ARTIST project.

2 Model Discovery in a MDRE process

Model discovery is the action of obtaining raw models from the software artefacts of the legacy system to be reverse engineered. These models are actually *initial models* because they are exact and direct representations of the system (i.e., there is no abstraction gap). Thus, each one of these models conforms to a metamodel that represents concepts at a low abstraction level for the required artefact, for instance, programming language metamodels (e.g., Java or C++), file's format metamodels (e.g., Microsoft Word), etc.

Models obtained during the model discovery normally become the starting point for MDRE processes, thus this step is fundamental in the great majority of MDRE scenarios. After the model discovery step, the next step of the MDRE process focuses on deriving higher level models (called *derived models*) of the system, thus providing a more specific view of those parts of the software system which are more interesting for the remainder of the process. This second step is called *model understanding* and allows notably getting models describing either functional or non-functional features of the (legacy) software system. **Error! Reference source not found.** presents a big picture relating these two phases, as composing a full MDRE process. In this document we will focus on the first step involving model discovery.

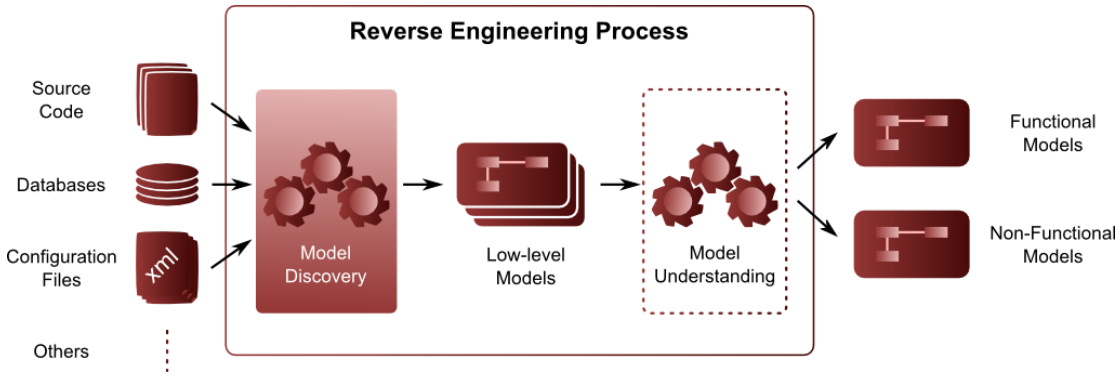

Figure 1. Overview of Model Driven Reverse Engineering

The software components in charge of obtaining models from software artefacts are called *model discoverers*. Due to the specificities of each kind of software artefact, these components are generally developed for a given type of artefacts. As a consequence, a set of different model discoverers is usually needed to cope with the heterogeneity of the artefacts composing a software system.

In a MDRE process, models are discovered from the concerned software artefacts and are later used in following phases to perform the actual migration. Since a big number of different software artefacts are normally treated, a considerable effort to analyse and classify each group of artefacts is required to later define and/or reuse the corresponding model discoverers.

To facilitate this task of classifying the software artefacts composing the system, we propose the definition of a taxonomy of legacy artefacts. This taxonomy is particularly relevant according to three main perspectives:

- For software architects, as it can allow them to have a more accurate vision of the current state of their legacy system and thus facilitating its global (technical) understanding.

- For modernization engineers (i.e., model discoverer users), as it provides some good support in the process of identifying or selecting the most appropriate model discoverers in their particular scenario.
- For Model Discovery experts (i.e., model discoverer creators), as it offers detailed information about the legacy artefacts to guide the model discoverer design and implementation.

3 Taxonomy of legacy artefacts

This section describes the proposed taxonomy of legacy artefacts, and also provides additional information useful for the model discovery phase (e.g., how the model discovery phase is potentially influenced accordingly). The taxonomy is reflecting our current vision and understanding of the situation, but is not exhaustive or complete. Thus, it could be extended and/or refined in the future according to newly identified characteristics or properties.

The taxonomy is based on several dimensions from which software artefacts can be considered. These dimensions allow evaluating the actual "type" of a legacy artefact and thus provide by default a better understanding of the elements composing the legacy system to be migrated. They can be generally organized into two main groups:

- The "technical" ones (i.e., the technical space, the origin, the organization, the nature, the opacity and the environment);
- The "non-technical" ones (i.e., the purpose, the consumer, the size and the architectural layer).

One of the main reusable outcomes of this work is a classification template that can be adapted to/filled in the different concrete scenarios. Depending on how the considered legacy artefacts are going to be positioned within the obtained template (cf. Section 0), the way the model discovery will actually be performed afterwards can be more or less strongly impacted (e.g., identifying which software artefacts will require more effort or which ones could be overlooked).

The next subsections will describe each one of the different dimensions and will mention their respective potential impact on the model discovery phase.

3.1 The Technological Context: Technical Spaces

The concept of Technical Space (TS) is presented in [4] and provides guidance with regards to the use of a specific technological context. It can be defined as a working context with a set of associated concepts, body of knowledge, tools, required skills and capabilities. The artefacts composing a software system are usually part of several different TS. For instance, source code conforming to a grammar (whose TS is called *grammarware*), an XML document conforming to an XSD (whose TS is called *xmlware*), etc.

A TS is defined based on a couple of basic concepts which are related to each other by means of a conformance relationship (e.g., program/grammar in *grammarware* or document/schema in *xmlware*). TSs are often organized in a 3-level architecture as in the OMG MDA [5]. **Error! Reference source not found.** shows some examples of TSs broadly used in software engineering.

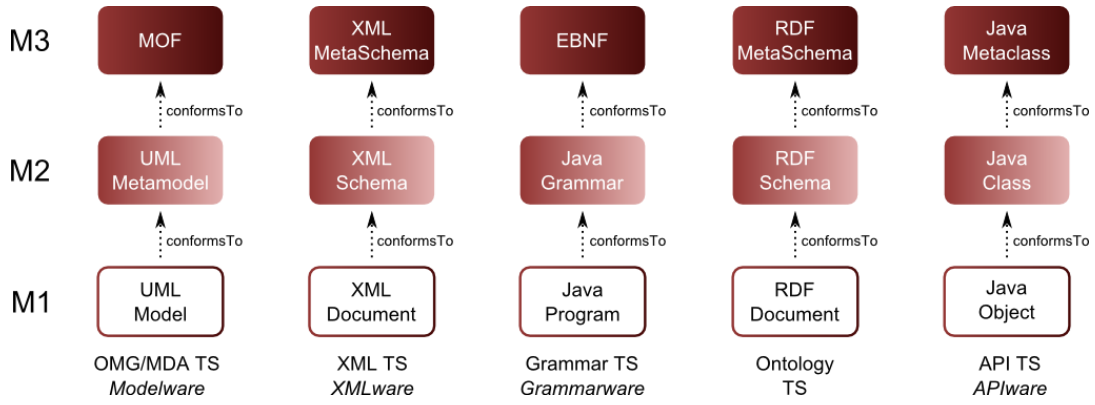

Figure 2. Some common TSs within the three level organization.

TSs are not isolated and (bidirectional or one-way) bridges can actually be defined between two different TSs, thus allowing them to interoperate and interchange data. This allows switching from one TS to another in order to benefit from more efficient capabilities provided in other TSs. For instance, a bridge between *grammarware* towards the MDE TS, called *modelware*, would allow using the principles and techniques provided in the modelling world. In fact, Model Discovery can be seen as the action of bridging *modelware* with other TSs, as models are created out of artefacts coming from these other different TSs.

Thus, classifying the set of software artefacts according to the TS to which they belong is a clear way to identify which discoverers are needed. The required effort directly depends on the existence or not of the corresponding discoverer(s). If new discoverers have to be implemented, the complexity of the concerned TSs will have a strong impact on the design and development effort of the model discovery phase.

3.2 The Origin: Manual vs. Generated

Many artefacts in a software system have been produced manually by developers. However, and more especially in the context of MDE, artefacts can also be derived automatically (at least partially) from other artefacts. This comes typically with the use of transformations that convert, merge and/or filter information from one or several artefacts and generate new artefacts from them.

According to this dimension, the artefacts can be classified into three main categories:

- **Manual**: artefacts are completely produced manually (by developers for instance) with no automated intervention;
- **Partially generated**: artefacts, or at least their skeleton or superstructure, are produced automatically but with the intent of being modified manually afterwards;
    - **Free modifications**: manual modifications have been performed globally within a whole artefact;
    - **Restricted modifications**: manual modifications have been performed only in some pre-defined parts of the artefacts (e.g., by the template's producer when dealing with code generation);
- **Fully generated**: artefacts are completely produced automatically with no intent of being modified manually.

This dimension allows identifying which artefacts should be taken into consideration during the software migration process. Note that the migration can involve either the generation or reuse of the artefact. Software artefacts created manually are usually considered to be

migrated whereas those that are fully generated may be ignored, as they can potentially be regenerated afterwards. Software artefacts which are partially generated require a deeper analysis, for instance to identify which parts (containing the actual knowledge or human inputs) should be concerned by the discovery.

3.3 The Purpose: Code vs. Documentation

Artefacts can be classified according to the purpose they serve in a software project. This purpose might not in all cases be unambiguous, since artefacts can sometimes serve different purposes at the same time (e.g., abstracted functionality and documentation).

According to this dimension, the artefacts can be classified into one or several of these four main categories:

- **Code**: artefacts composing the core of the legacy system and actually implementing their actual capabilities (e.g., Java source code, SQL database schema, etc.);
- **Configuration**: artefacts that are part of the legacy system sources but more dedicated to the systems' parameterization and customization (e.g., using pre-defined frameworks);
- **Specification**: artefacts defined and used to specify legacy systems, notably prior to their implementation (e.g., architecture or design models);
- **Documentation**: artefacts written during or after the implementation of the legacy systems and intended to be provided to future users and/or developers (e.g., user manual).

A typical software migration process requires dealing with code and configuration files. However, in some cases, it can also be required to analyse existing specifications or documentation in order to be able to extract useful information for the modernization process, e.g., some design models of the legacy system. Depending on the format of the artefact content (e.g., natural language, cf. also subsection 3.5), it may be difficult to extract automatically relevant information.

3.4 The Consumer: System vs. End User

When dealing with different artefacts of a legacy software, several entities actually consuming these artefacts are usually also concerned. This "consumer" dimension can be sometimes linked to the "purpose" of the artefacts (cf. Section 3.3). Generally, many artefacts specify the system to be developed. Some describe and allow configuring the infrastructure the system operates in. Others are produced for the end-user in order to enable him/her to use the system. Developers can also produce artefacts that are primarily intended to them (e.g., bug reports, IDE configurations).

The corresponding artefacts can be classified into four main categories according to their so-called consumer:

- **System**: artefacts intended to be consumed directly by the legacy system itself (e.g., the core source code of an application);
- **Infrastructure**: artefacts implemented to be provided to the legacy system's underlying infrastructure (materialized by APIs for instance) so that they can run properly (e.g., configuration files);
- **Developer**: artefacts specified to be provided to the developers so that they can understand better and/or implement the legacy system (e.g., specifications models, developer documentation, bug tickets, etc.);

- **End user**: artefacts implemented to be provided to the users so that they can actually use the legacy system in practice (e.g., base examples/demos).

While this dimension clearly allows for a better understanding of the artefacts composing the software system, we currently do not see a direct relation between the different categories of this dimension and how the model discovery phase would be performed.

3.5 The Organization: Structured vs. Unstructured

Some software artefacts conform to organized architectures with explicitly specified behaviours and well-defined data schemes (e.g., source code, databases, XML-based frameworks, etc.). Some others are less rigorous, have less restrictions or let more freedom in terms of internal structure, data, etc. (e.g., documentation).

According to an organizational perspective, three main types of artefacts can coexist in a same system:

- **Structured**: artefacts having completely structured data schemes (e.g. databases, XML documents conforming to XML schemas);
- **Unstructured**: artefacts without any restriction in the structure (e.g. textual files);
- **Semi-structured**: every artefact in between, i.e., containing structured parts and more open ones (e.g., Excel files not using explicitly a well-defined template).

This dimension is related with the TS one. The organization of a software artefact directly depends on the fact that its TS defines (or not) a formalism to specify the structure. For instance, source code files belonging to the *grammarware* TS, where a grammar defines the structure of the language, will be classified as structured artefacts.

This distinction in terms of structure has a strong impact on Model Discovery: the more an artefact is structured (i.e., being structured better than semi-structured), the easier it will be to analyse it using an automated process. On the opposite, an unstructured artefact will be much more difficult to “parse” without human intervention during the process.

3.6 The Nature: Dynamic vs. Static

Independently from their complexity, some software artefacts can evolve in time while others do not. The corresponding changes can be related to the software system, its actual deployments or handled data. Also, some artefacts can represent more dynamic information (e.g., data or execution trace) or more static one (e.g., source code).

During the actual Model Discovery, artefacts will most of the time have to be “frozen” at some point. This is principally due to technical/implementation reasons, as it is still challenging today to efficiently deal with “infinite” inputs at a precise moment in time. Nevertheless, it is important to take these two aspects into consideration:

- Is the legacy artefact itself potentially evolving during the Model Discovery or after?
- Is the represented information “static” or “dynamic”?

Behind these two questions, there is the general problem of evolution management which can affect Model Discovery. Dealing with evolving artefacts means that the Model Discovery process really has to support it via discovery iteration, version control, conflict resolution, etc. This can have a significant impact on the processes to be defined.

3.7 The Size: Small vs. Large

Some systems are naturally simple whereas some others are (sometimes much) more elaborated. This difference can come from several aspects of the concerned system such as the addressed domain itself, the number and variety of the provided features, the size of the system's (internal) structure, the technical architecture the system is actually deployed on, the volume of the processed data, etc.

Units to evaluate the size should be adapted to the artefacts which are being analysed, for instance, according to:

- **Lines of code** (for textual files or source code);
- **Tuples** (for databases);
- **Model elements** (for models);
- **Nodes** (for xml documents);
- **Bytes** (for binaries);
- Etc.

The size of the legacy artefacts is one of the key aspects to consider in Model Discovery. It largely influences the way the corresponding model discoverers will have to be built or selected, according to the potential scalability and coverage issues implied by their size. Within this context, the choice of the most appropriate infrastructure and techniques for performing Model Discovery, e.g., in term of performance or completeness, is essential.

3.8 The Opacity: White box vs. Black box

Some systems freely expose their architecture, sources and data while others hide (parts of) them. An extreme case is when all the system's internal information is made available, automatically or on demand, to the environment via interfaces such as export features, APIs or data streams. The other extreme is where there is no direct way to access the structure and data of a completely closed system. In the general case, many systems offer an in-between situation.

The corresponding artefacts can be classified into three main categories according to this “opacity” dimension:

- **Black box**: “closed” artefacts (e.g., binary files, even if some de-compilers may exist);
- **White box**: all the (useful) artefacts structure and data are available (e.g., source code, databases);
- **Grey box**: every artefacts in-between (e.g., system’s data only accessible via limited interfaces or APIs).

This dimension is also important when dealing with Model Discovery. Notably, it is crucial to identify as soon as possible if the available legacy artefacts allow a sufficient access for the legacy system to be reverse engineered as expected. Otherwise, alternative solutions may have to be found, including having to collect the needed information via other means (e.g., from other artefacts or even human interaction).

3.9 The Architectural Layer: Conceptual Content

Artefacts can represent concepts at different conceptual levels of an application, and thus be part of different architectural layers. Typically, systems are structured in layers (that are built on top of each other, depend on each other, etc.) in order to separate concerns and so to better manage complexity. Typical layers are “data”, “communication”, “logic” or

"presentation". The architectural layer has influence on both the technical concepts and technologies used in the respective layers. There might be (high level) artefacts that represent various architectural layers at the same time.

According to this dimension, the artefacts can be classified into four main well-known categories:

- **Data**, including those software artefacts in charge of managing the data model of the system;
- **Communication**, which consider those software artefacts dealing with external interfaces (e.g., web services, sockets, etc.);
- **Logic**, which includes those software artefacts implementing the business logic of the system;
- **Presentation**, including those software artefacts defining the user interface of the system.

Depending on the targeted modernization scenario, all or part of the artefacts composing the previously enumerated categories are concerned by the Model Discovery phase. Moreover, they can normally be considered separately when discovering models (e.g., the user interface may be considered in an isolated way). However, it is not trivial to identify complexity levels without analysing the actual artefacts composing each category. For instance, the more complex the data layer is (e.g., the database schema is rich on tables and associations) the harder the discoverer process can be.

3.10 The Environment: Supporting Tooling

Depending on their types, legacy artefacts can be developed, run, processed, deployed and maintained onto different environments. When talking about such environments or supporting tooling, all tools/software/components used during the life cycle of the legacy artefacts are included.

Among possible candidates for characterizing this dimension, the following entities (but not only) could be considered:

- **Development environments**;
- **Compilers**;
- **Virtual machines**;
- **Execution platforms**;
- Etc.

This dimension is particularly important for the actual performance of the model discovery phase because it usually determines the technological requirements to access to the artefacts composing the software system. Thus, it really influences both the design and implementation of the corresponding model discoverers.

4 Classification Framework

This section presents a simple concrete framework that needs to be applied to the legacy artefacts for classifying them considering the proposed taxonomy. This framework is based on a single template that allows describing the treated legacy artefacts according to the different dimensions composing the taxonomy. Thanks to the collection of these data, useful information can be obtained on the legacy system itself, independently from the targeted objectives. More particularly, in an ARTIST/modernization context, the potential impact on model discovery can be studied, as well as the support currently proposed/missing by/from the available MDRE solutions.

The proposed template is a two dimensional table which is shown in Table 1. The table uses the TS dimension as a base (for columns) while considering the other dimensions as rows allowing for a fine-grained classification of the legacy artefacts. As shown by the template, several columns can be used to identify concrete technological subspaces (e.g., Java or C# technologies in the *grammarware* TS). One table must be filled in for each TS involved in the migration process, thus resulting in several different tables for a given legacy system.

Table 1. Template for the taxonomy

	Use Case/Scenario: Technical Space:		
	Sub-category 1	Sub-category 2	Sub-category 3
Origin			
Purpose			
Consumer			
Organization			
Nature			
Size			
Opacity			
Architectural Layer(s)			
Environment			

In the context of Model Discovery, using TSs as the main dimension provides a first classification which helps to identify the (family of) model discoverers required to deal with the software artefacts. Once the TS has been identified, the other dimensions enables a better characterization of the software system at hand and may allow developers to measure the effort required to perform the discovery process.

- The Origin (i.e., Manual vs. Generated). Generated elements can usually be ignored during the model discovery process thus reducing the time and effort required in the migration.
- The Purpose (i.e., Code vs. Documentation). This dimension is normally related to the TS (e.g., code normally is included in the *grammarware* TS while the configuration files usually use the *xmlware* TS). This way, the purpose allows for a fine-grained classification of the software artefact and helps developers to identify the correct discoverer required in the migration.

- The Consumer (e.g., System vs. End User). The classification provided by this dimension rarely affects the performance of the discovery process. Instead, it is useful to better classify and understand the artefacts composing the system.
- The Organization (i.e., Structured vs. Unstructured). Similarly to the purpose dimension, this critical one is also related to the TS used to define the software artefact and helps developers to better classify them.
- The Nature (i.e., Dynamic vs. Static). The nature of a software artefacts may help to detect those elements which could require a continuous model discovery process (i.e., a dynamic software artefact can require being discovered each time it changes) or, at least, the need for applying several times the process.
- The Size (e.g., Small vs. Large). The size of a software artefact helps developers to identify which ones could require more effort to be discovered.
- The Opacity (e.g., White box vs. Black box). The detection of black-box elements can hamper the model discovery phase because they can be difficult to be analysed. On the other hand, it is expected to mainly deal with white-box elements, which can easily be accessed to discover models.
- The Architectural Layer (e.g., Conceptual Content). Similarly as what happens with the consumer dimension, this one rarely affects the performance of the process.
- The Environment (e.g., Supporting Tooling). The technological context (i.e., the set of tools required to implement/execute the system to be migrated) can help developer to identify which discoverers are required to perform the migration.

The different model discoverers developed in the context of the ARTIST project will be described within the D8.2.x deliverable series.

5 Case studies

In this section the presented taxonomy is illustrated by using the case studies involved in the ARTIST project. The migration scenario involved in each case study is first shortly described and, for the sake of conciseness, a summary of the set of templates describing the system is then shown (the complete set of templates can be found as appendixes at the end of this document). A tentative set of needed model discoverers is also described for each case study. Note that the final list of required discoverers may change during the development of the ARTIST project.

5.1 DEWS CCUI (ATOS)

This case study involves the migration of DEWS system into a Hybrid Cloud (e.g., coexistence of public and private Clouds) to enable Cloud Bursting techniques to support peaks of seismic activity. DEWS in the Cloud takes advantage of some Cloud features, notably on the higher efficient usage of computational resources and their cost reduction, and also on their flexible maintenance and architecture topology. The main DEWS components that will be migrated are:

a) the DEWS console (CCUI), a desktop application (Eclipse RCP) whose architecture is not Cloud compliant
b) DEWS services, such as ILC, IDC, SPC, SS[1], which will be deployed in the Cloud, according to Cloud Bursting pattern
c) DEWS data sources of DEWS CCUI and DEWS Services, which rely on diverse persistence technologies, ranging from RDBMS to XML.

The whole system is composed of: (1) source code files (i.e., Java and Python), (2) data files (mainly XML), (3) resource files (i.e., pictures) and (4) several APIs. However, according to the purpose of the migration process, only three types of artefacts are involved: (1) source code and libraries (all of them developed in Java), (2) data schemas defined in SQL and XML schemas, and (3) configuration files in XML or property files. Table 2 shows a summary of the resulting taxonomy according to the expertise of the model discovery developers of the project.

Table 2. Summary of the taxonomy for DEWS CCUI case study

Dimension	General overview of the TSs
Origin	Source code mostly manual, only data-management source code is generated
Purpose	Mainly application code and data management
Consumer	End-users and/or System
Organization	Mostly structured
Nature	Static
Size	Medium size application
Opacity	GNU-GPL
Architectural Layer	Presentation, Business logic, Data
Environment	Mainly Eclipse Indigo, JDK 6, Linux

[1] See D12.1 for additional descriptions

In the context of ARTIST project, while for the first three types there is already existing support in the ARTIST Model Discovery toolbox (e.g., in the MoDisco framework [3]), it is required to develop a specific model discoverer for SQL. The effort foreseen for developing this required discoverer may be low since there are currently some approaches [6] performing a similar model discovery step from SQL sources.

5.2 LoB (Spikes)

This case study involves the migration of *SpikesTogether*, one of Spikes' main products. SpikesTogether is a toolkit/framework built on top of Microsoft SharePoint facilitating rapid application development. It has a strong focus on flexible and collaborative workflows. The toolkit mainly consists of a custom workflow engine and means to design and manage workflows, all tightly integrated with Microsoft SharePoint. The aim is to go from an on-premise single context instance to a multi-tenant SaaS solution by migrating both the frontend (management) and the backend (engine) part of the toolkit. The toolkit also incorporates a Designer part but it will not be considered. The main reasons for the migration are the adoption for cloud technologies together with a strong market pull.

The whole system is composed of: (1) source code files (i.e., C#. PowerShell, JavaScript, HTML, ASP Markup, XAML), (2) data files (i.e., MSSQL and DBML), (3) XML files (i.e., configuration, SharePoint definitions, service descriptions and schemas), (4) plain text files (i.e., Word, PDF, HTML, docs and security) and (5) resource files (i.e., pictures, Visio, batch).

According to the purpose of the migration process, there are two main artefacts involved for which model discoverers are needed: (1) source code (i.e., C# and ASP.NET mark-up files) and (2) configuration, Microsoft SharePoint and Service description files in XML. Table 3 shows a summary of the resulting taxonomy according to the expertise of the model discovery developers of the project.

Table 3. Summary of the taxonomy for LoB case study

Dimension	General overview of the TSs
Origin	GPL manual, DSLs generated
Purpose	Mainly application code and configuration
Consumer	Developer, System and End-User
Organization	Mostly structured
Nature	Static
Size	Medium for GPL, Small for DSLs
Opacity	White-box
Architectural Layer	Presentation, Business logic, Data
Environment	Microsoft Visual Studio, SQL Server, Visio

In the context of the ARTIST project, the former type of artefacts will require the implementation of specific discoverers dealing with these programming languages (cf. D8.2.1 – "Components for Model Discovery"), while the latter type of artefacts (i.e., XML-based files) is already included in existing discoverers provided by the existing toolbox.

5.3 eGov (ENG)

The eGov scenario proposed by ENG involves the migration of some components of the Public System for Cooperation (SPCoop), which lets its users to have a unified view (independent from the provisioning channel) of all services of Italian public administrations both central and local. It defines a shared model for on-line interaction that saves the specificity of service provider and, at the same time, assures uniform interaction and identification of both service provider and service consumer (more in general of all actors participating in a interaction). In particular, the selected components are the SPCoop Domain Gateway and the Services for Interoperability, Cooperation and Access Control (SICA).

The whole system is composed of: (1) source code files (i.e., Java, OWL, WSDL), (2) XML files (i.e., configuration and schemas), (3) plain text files (i.e., Word and HTML) and (4) resource files (i.e., pictures, EA diagrams and property files).

According to the purpose of the migration process, the main artefacts which are going to be involved are: (1) source code files involving Java source files, (2) service definition including OWL and WSDL files, (3) configuration files in XML, (4) data definition in XML schemas and (5) property files. Table 4 shows a summary of the resulting taxonomy according to the expertise of the model discovery developers of the project.

Table 4. Summary of the taxonomy for eGov case study

Dimension	General overview of the TSs
Origin	Partial generative approach
Purpose	Mainly application code
Consumer	End-users, System, Developer
Organization	Mostly structured
Nature	Static
Size	Large (OWL) the rest mostly small
Opacity	White-box except for Java
Architectural Layer	Presentation, Business logic, Data
Environment	Mainly Eclipse Indigo, JDK 6 + Protegé

In the context of the ARTIST project, it will be required to provide model discoverers for OWL and WSDL files while the discovery of model from Java, XML, XML schemas and property files is already covered by the existing toolbox.

5.4 NewsAsset (ATC)

This case study involves the migration of News as an Asset (*NewsAsset*), an end-to end multimedia cross-channel solution for evolving News Agencies, Broadcasters and Publishers. In the scope of ARTIST, *NewsAsset Agency Edition*, a modular, configurable, all-in-one multimedia solution marketed by ATC will be migrated. It supports the whole Life-cycle of news item from planning, through creating, gathering and selecting editing, to producing, distributing and archiving. An All-in-One system covers all internal, user-generated, web-accrued or wired multimedia assets, all types of workflow and all news asset management

activities, via a friendly, industry-standard compliant interface. Workflow is flexible and configurable.

ATC is interested in using a combination of public and private cloud services. A private cloud will be utilized to support its highly distributed development application and may leverage public commercial cloud services to ensure that customer service remains satisfactory during times of peak use. The following *NewsAsset* layers are considered: a) Event Management and Editorial planning layer, b) Content Creation, Ingestion and Aggregation Layer, c) Adaptation for multiple channels layer and d) Distribution Channels.

The whole system is composed of: (1) source code files (i.e., C#, ASP.NET, Javascript, HTML, CSS and SQL), (2) APIs (i.e., external component libraries and ActiveX mainly), (3) XML files (i.e., configuration and service descriptions).

According to the purpose of the migration process, the main artefacts which are going to be involved are: (1) source code involving C#, ASP.NET and JavaScript files, (2) website source files including HTML and CSS files, (3) data from Oracle database and (4) configuration files in XML. Table 5 shows a summary of the resulting taxonomy according to the expertise of the model discovery developers of the project.

Table 5. Summary of the taxonomy for News Asset case study

Dimension	General overview of the TSs
Origin	Manual (few parts are generated)
Purpose	Large variation
Consumer	Mainly system (and end-user in some cases)
Organization	Structured
Nature	Static
Size	Large (application code), medium for the rest
Opacity	White-box except for some APIs/libraries
Architectural Layer	Presentation/Communication, Business logic, Data
Environment	Microsoft Visual Studio (.NET v3.5), Oracle RDBMS

In the context of ARTIST project, given that there are no model discoverers for the great majority of software artefacts involved in the migration (except for XML files and the data, which can be tackled by components from the toolbox), the model discovery phase for this case study will require a considerable implementation effort. In particular, discoverers for C#, ASP.NET, JavaScript, HTML and CSS languages have to be developed. On the other hand, model discoverers for XML and SQL (database definition and context) could be reused.

6 Discussion & Potential improvements

The proposed version of the taxonomy has allowed us to characterize with more precision each case study involved in the ARTIST project. Its application to the project's use cases has also provided valuable feedback to discuss and identify potential improvements on it. Generally, it is also important to note that each dimension will be practically validated as the project evolves, thus allowing us to assess how they actually affect the implementation effort of the model discoverer.

According to the collected information up to now, the most valuable dimensions which allow better characterizing the software system and its artefacts are:

1. **Technical spaces**. This dimension becomes the main axis of the taxonomy, thus allowing performing the first classification (according to the proposed template) to easily identify the required (family of) model discoverer(s).
2. **Origin**. This dimension allows easily filtering out the elements that are automatically generated and so that are not directly concerned by the migration process.
3. **Purpose**. The classification according to the purpose of the software artefact allows for a finer-grained categorization, which usually helps developers to identify which elements must be migrated (and which must be not).
4. **Size**. The size of the software artefacts serves as a good indicator of the resources needed to perform the migration, whether it is done automatically or it is applied manually.
5. **Architectural layer**. The separation of artefacts according to the architectural layer, when possible, provides valuable information to identify the main parts of an application (i.e., the ones that will be fundamental to address during the modernization scenario).
6. **Environment.** This dimension allows developers to understand how the application was developed and is executed, and also impacts the model discoverer design decisions and building process.

Other dimensions were useful to classify the software system, but they do not really seem to impact the migration process in terms of improving the model discovery phase. These dimensions are:

1. **Organization**. Although knowing whether an artefacts is structured (or not) is a valuable information to locate the proper model discoverer, this information is already strongly linked to the Technical Space dimension.
2. **Nature**. This dimension does not really influence the selection of the model discoverers nor the measurement of the required effort for developing new ones.
3. **Opacity**. The possibility to access the source describing an artefact is crucial for applying the model discovery phase. Obviously, if dealing with black-box artefacts the model discovery is most of the times simply not applicable (and alternative options, e.g., more manual, have to be envisioned).
4. **Consumer**. The final consumer of artefacts does not currently appear to be practically useful when applying the model discovery phase. However, it can bring some value from a purely informative/understanding perspective on the legacy system.

One important element to face when migrating legacy software is the level of reuse in terms of software artefacts to be migrated. This means that some parts of the legacy code may not be required to be migrated but adapted (or wrapped) in the new system. At the moment,

the reuse level of an artefact has not been considered by the current version of the taxonomy. However, we foresee the study of possible extensions within the ARTIST project during the development of the case studies.

7 Conclusion

The discovery of models from the legacy software system is crucial in a model-driven migration process, which relies on such models to succeed. Dealing with the number of different software artefacts composing a system, and studying how models can be obtained from them, is not an easy task. In this document, we presented a taxonomy which helps developers analysing the nature of each artefact of the system, thus providing an overview of the system to be migrated and facilitating the identification of the required model discoverers. The taxonomy may also help developers to have an idea about the effort needed to perform the discovery (e.g., the previous existence of model discoverers for a concrete artefact can decrease the global effort needed for the full process). The taxonomy is accompanied with a classification framework to facilitate its application, which has already been put into practice with four industrial case studies taken from the project.

Thus, within the context of ARTIST, we have currently applied the taxonomy to characterize the software systems provided by each case study. This has allowed us to identify which model discoverers will be required, and also to have a first idea about the effort which is needed to perform the model discovery phase.

We plan to polish and refine the taxonomy definition (e.g., considering the proposed dimensions) along with the classification framework as the project evolves. These findings are planned to be discussed in future deliverables around the second year of the project (e.g., D8.3 – "Methodology and techniques for model understanding" or D12.3.1 – "Deployed use cases"). For instance, we foresee possible improvements related to the level of reuse of software artefacts as commented in the previous section. We also plan to study more deeply how each taxonomy dimension affects the actual implementation of the model discovery phase once the various case studies are deployed and tested.

8 References

[1] B. Selic, "MDA Manifestations," *UPGRADE, the European Journal for the Informatics Professional,* vol. 9, no. 2, pp. 7-11, 2008.

[2] J. Bézivin, M. Barbero and F. Jouault, "On the Applicability Scope of Model Driven Engineering," in *Workshop on Model-based Methodologies for Pervasive and Embedded Software*, 2007.

[3] H. Bruneliere, J. Cabot, F. Jouault and F. Madiot, "MoDisco: a Generic and Extensible Framework for Model Driven Reverse Engineering," in *Conference on Automated Software Engineering*, 2010.

[4] I. Kurtev, J. Bézivin and M. Aksit, "Technological Spaces: An Initial Appraisal," in *Industrial track in CoopIS, DOA'2002 Federated Conferences*, 2002.

[5] OMG, "MDA," [Online]. Available: http://www.omg.org/mda/. [Accessed 6 9 2013].

[6] J. L. Cánovas Izquierdo and J. García Molina, "Extracting Models from Source Code in Software Modernization," *Software and Systems Modeling,* vol. Model Evolution Special Issue, 2012.

APPENDIX A: Templates for DEWS CCUI case study

	Use Case/Scenario: *DEWS (ATOS)* Technical Space: *Source code (GPL)*	
	Sub-category 1: Java	**Sub-category 2: Python**
Origin	Mostly manual, only some data management code generated (1 project)	Manual
Purpose	Application code (32 projects), Data management code (2 projects)	Application code (4 projects)
Consumer	Most projects targeting end-users (19 projects) and system (10 projects)	System (4 projects)
Organization	Structured code, according to Eclipse RCP rules	Structured code (Python rules)
Nature	Only static: source code of the RCP application	Only static
Size	"Medium" application: 34 projects, 1127 Java classes	"Small" application: 4 projects, 64 python files
Opacity	White-box: open source under GNU-GPL	White-box: open source under GNU-GPL
Architectural Layer(s)	Mainly presentation layer (21 projects), Business logic (7 projects), data management (6 projects)	Business logic (4 projects), data management (4 projects)
Environment	Development: Eclipse Indigo Running: Java Virtual Machine (Java 6)	Python (version ?), Linux

	Use Case/Scenario: *DEWS (ATOS)* Technical Space: *XML And other data formats*	
	Sub-category 1: XML and variants:	**Sub-category 2 Other Data Formats**
Origin	Semi-automatically generated (manual edited)	Mostly generated
Purpose	Configuration (1 projects), Project Configuration (23 projects), Project building (23 projects), configuration launch (1 projects), Geographic Data (1 projects), Data management/exchange (1 projects), Data schema (1 projects)	Internationalization (1 projects), Project building (22 projects), Configuration (2 projects), Geographic Data (2 projects), Data management/exchange (4 projects), fonts (1 project)
Consumer	System (34 projects)	System (4 projects)
Organization	Structured (XML and variants: XSD, EXSD, OVF, SQL, scripts)	Structured (scripts), semi-structured (properties, map, csv, cfg, poi, dat, list), binary(tty, grd, dbf, shd, shx)
Nature	Static	Mostly Static, some dynamic configuration data.
Size	"Medium" application: 34 projects: 114 files	"Medium" application: 34 projects: 681 files
Opacity	White-box: open source under GNU-GPL Some confidential data	White-box: open access. Some confidential data

Architectural Layer(s)	Mainly presentation layer (21 projects), Business logic (7 projects), data management (6 projects)	Mainly presentation layer (23 projects), Business logic (4 projects), data management (5 projects)
Environment	Development: Eclipse Indigo Running: Java Virtual Machine (Java 6)	Python (version ?), Linux

	Use Case/Scenario: *DEWS (ATOS)* Technical Space: *Plain Graphics*
	Sub-category 1: PNG/GIF/JPG/BMP
Origin	Generated
Purpose	Application Images/Icons (14 projects), data (1 project)
Consumer	End-user (14 projects), System (1 projects)
Organization	Structured (PNG, GIF, JPG, BMP)
Nature	Static
Size	"Medium" application: 15 projects: 225 images
Opacity	White-box: open source under GNU-GPL
Architectural Layer(s)	Mainly presentation layer (14 projects), data management (1 projects)
Environment	Unknown, images given.

	Use Case/Scenario: *DEWS (ATOS)* Technical Space: *API*
	Sub-category 1: API
Origin	Generated
Purpose	Application code (7 frameworks), data management (5 frameworks)
Consumer	System (12 frameworks)
Organization	Structured
Nature	Static

Size	Small frameworks (9 frameworks), medium (2 frameworks), large (2 frameworks): total number of libraries (Jars): 174
Opacity	White-box: FLOSS
Architectural Layer(s)	Data (5 frameworks), Business logic (5 frameworks), Presentation (2 frameworks)
Environment	Java

APPENDIX B: Templates for LoB case study

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Source code (GPL)*		
	Sub-category 1: C#	**Sub-category 2 PowerShell**	**Sub-category 3: Javascript**
Origin	Mostly manual, some code-behind files are automatically generated. Few projects are partially generated	Manual	Manual
Purpose	Application code	Configuration	Application Code
Consumer	System	System	System
Organization	Structured code	Structured scripts	Structured code
Nature	Static	Static	Static
Size	"Medium": 708 source files	"Small": 15 script files	"Small": 8 scripts
Opacity	White-box	White-box	White-box
Architectural Layer(s)	Business Logic		Presentation
Environment	Development: Microsoft Visual Studio Running: .NET Framework (v.4)	Development: PowerShell ISE Running: Powershell	Development: Microsoft Visual Studio Running: Browser

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Source code (DSL) 1/2*		
	Sub-category 1: HTML/CSS	**Sub-category 2 ASP Markup**	**Sub-category 3: XAML**
Origin	Partially generated	Partially generated	Partially generated
Purpose	Application code	Application code	Application code
Consumer	System	System	System
Organization	Structured code	Structured code	Structured code
Nature	Static	Static	Static
Size	"Small": 7 html files, 2 css files	"Medium": 56 server pages (aspx), 59 user controls (ascx), 1 handler (ashx), 1 web service (asmx)	"Small": 4 files
Opacity	White-box	White-box	White-box

Architectural Layer(s)	Presentation	Presentation	Presentation
Environment	Development: Microsoft Visual Studio Running: Browser	Development: Microsoft Visual Studio Running: Browser	Development: Microsoft Visual Studio Running: .NET Framework

	Use Case/Scenario: *DEWS (ATOS)* Technical Space: *XML And other data formats*	
	Sub-category 1: MSSQL	**Sub-category 2 DBML**
Origin	Manual	Mostly manual
Purpose	Configuration	Application code
Consumer	System	System
Organization	Structured code	Structured model
Nature	Static	Static
Size	"Small": 10 files	"Small": 1 database model
Opacity	White-box	White-box
Architectural Layer(s)	Data	Data
Environment	Development: SQL Server Management Studio Running: MSSQL	Development: Microsoft Visual Studio Running: .NET Framework

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *XML*		
	Sub-category 1: Config/Project files	**Sub-category 2 Definitions**	**Sub-category 3: Services**
Origin	Partially generated	Partially generated	Partially generated
Purpose	Configuration	Code	Code
Consumer	System and End-User	System	System
Organization	Structured code	Structured code	Structured code
Nature	Static	Static	Static

Size	"Medium": 19 config files, 41 project files, 6 settings files, 6 user options files, 3 solution files	"Medium": 151 XML files, 7 datasource files, 37 feature files, 21 package files, 91 item data files, 9 webparts, 16 resource files	"Small": 2 WCF Service (svc) and 3 Service Descriptions (wsdl)
Opacity	White-box	White-box	White-box
Architectural Layer(s)		Data and Business Logic	Presentation
Environment	Development: Microsoft Visual Studio	Development: Microsoft Visual Studio	Development: Microsoft Visual Studio Running: .NET Framework

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Models*
	Sub-category 1: XML Schema
Origin	Manual
Purpose	Specification
Consumer	Developer, system
Organization	Structured
Nature	Static
Size	"Small": 4 schemas
Opacity	White-box
Architectural Layer(s)	Business logic
Environment	Development: Microsoft Visual Studio

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Plain text*		
	Sub-category 1: Word, PDF	**Sub-category 2 HTML**	**Sub-category 3: Security**
Origin	Word files are created manually, PDF files are generated	Generated	Generated
Purpose	Documentation	Documentation	Configuration
Consumer	Developer and Business user	Developer	Developer, End-user, System

Organization	Unstructured	Structured	Structured
Nature	Static	Static	Static
Size	"Small": 10 word files, 10 PDF files	"Medium": 1060 HTM files + 96 PNG images	"Small": 1 license, 1 security certificate, 29 strong name keys
Opacity	White-box	White-box	Black-box
Architectural Layer(s)	Presentation	Presentation	
Environment	Opened: Microsoft Word / Adobe Reader	Opened: Web Browser	Development: LicenseKeyGenerator PuttyGen

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Plain graphics*	
	Sub-category 1: PNG/JPG/GIF/ICO	**Sub-category 2 Visio Diagrams**
Origin	Manual	Manual
Purpose	Application images/icons	Documentation (User Interface designs)
Consumer	System	Developer
Organization	Structured	Unstructured
Nature	Static	Static
Size	"Medium": 114 png, 6 gif, 5 jpg, 2 ico	"Small": 16 Visio diagrams
Opacity	White-box	White-box
Architectural Layer(s)	Presentation	Presentation
Environment		Opened: Microsoft Visio

	Use Case/Scenario: *LoB (Spikes)* Technical Space: *Miscellaneous*	
	Sub-category 1: Visio	**Sub-category 2 Batch**
Origin	Manual	Manual

Purpose	Application code	Configuration
Consumer	System	System
Organization	Structured	Structured
Nature	Static	Static
Size	"Small": 3 stencils, 2 templates	"Small": 2 batch files
Opacity	White-box	White-box
Architectural Layer(s)	Presentation	
Environment	Development: Microsoft Visio	

APPENDIX C: Templates for eGov case study

	Use Case/Scenario: *eGov (ENG)* **Technical Space: *Source code (GPL)***
	Sub-category 1: **Java**
Origin	Partially generated: some classes have been produced manually and other ones automatically (then manual modifications have been performed on some of them)
Purpose	Application Code
Consumer	System
Organization	Structured code according to the J2EE rules
Nature	Static
Size	"Small": 250 source files
Opacity	Black box
Architectural Layer(s)	Business Logic
Environment	Development: Eclipse Indico Running: Java Virtual Machine (Java 7) Environments: Apache Tomcat Apache CXF services framework Exist XML DB Libraries Required: OWL API Pellet OWL API XML DB API

	Use Case/Scenario: *eGov (ENG)* **Technical Space: *Source code (DSL)***	
	Sub-category 1: **OWL**	**Sub-category 2** **WSDL**
Origin	Manual	Partially generated
Purpose	Application code	Application code
Consumer	System End-user	System
Organization	Structured code according to the OWL 2 Web Ontology Language specification	Structured code
Nature	Dynamic	Static

Size	"Large": ~ 19780 owl files (one for each accredited public administration)	"Small": 2 service description files
Opacity	White-box	White-box
Architectural Layer(s)	Data	Business Logic
Environment	Development: Protégé ontology editor Running: Pellet OWL Reasoner	Development: Eclipse Web Tools Platform WSDL Editor

	Use Case/Scenario: *eGov (ENG)* Technical Space: *Source code (DSL)*	
	Sub-category 1: OWL	**Sub-category 2 WSDL**
Origin	Manual	Partially generated
Purpose	Application code	Application code
Consumer	System End-user	System
Organization	Structured code according to the OWL 2 Web Ontology Language specification	Structured code
Nature	Dynamic	Static
Size	"Large": ~ 19780 owl files (one for each accredited public administration)	"Small": 2 service description files
Opacity	White-box	White-box
Architectural Layer(s)	Data	Business Logic
Environment	Development: Protégé ontology editor Running: Pellet OWL Reasoner	Development: Eclipse Web Tools Platform WSDL Editor

	Use Case/Scenario: *eGov (ENG)* Technical Space: *XML*	
	Sub-category 1: Config/Project files	**Sub-category 2 XML Schema**
Origin	Partially generated	Partially generated
Purpose	Configuration	Specification
Consumer	Infrastructure	System

Organization	Structured code	Structured code
Nature	Static	Static
Size	"Small": 2 config files (web.xml and bean.xml deployment descriptor files)	"Small": 2 xml schema (serialized data)
Opacity	White-box	White-box
Architectural Layer(s)		Data
Environment	Development: Eclipse Web Tools Platform XML Editor Running: Apache Tomcat	Development: Eclipse Web Tools Platform XML Editor

	Use Case/Scenario: *eGov (ENG)* Technical Space: *Plain Text*	
	Sub-category 1: Word	**Sub-category 2 HTML**
Origin	Manual	Fully Generated
Purpose	Documentation	Documentation
Consumer	Developer End-user	Developer
Organization	Unstructured	Structured
Nature	Static	Static
Size	"Small": 2 word files	"Small": 450 HTM files
Opacity	White-box	White-box
Architectural Layer(s)	Presentation	Presentation
Environment	Viewer: Microsoft Word	Viewer: Web Browser

	Use Case/Scenario: *eGov (ENG)* Technical Space: *Plain Graphics*	
	Sub-category 1: JPG	**Sub-category 2 Enterprise Architect diagrams**
Origin	Manual	Manual

Purpose	Documentation	Documentation
Consumer	Developer	Developer
Organization	Structured	Unstructured
Nature	Static	Static
Size	"Small": 2 jpeg files	"Small": 2 Enterprise Architect diagrams
Opacity	White-box	White-box
Architectural Layer(s)	Presentation	Presentation
Environment		Tool: Enterprise Architect

	Use Case/Scenario: *eGov (ENG)* **Technical Space: *Miscellaneous***
	Sub-category 1: **Properties files**
Origin	Manual
Purpose	Configuration
Consumer	System
Organization	Semi-structured
Nature	Static
Size	"Small": 2 properties files (.properties)
Opacity	White-box
Architectural Layer(s)	Business Logic
Environment	Development: Text editor

APPENDIX D: Templates for NewsAsset case study

	Use Case/Scenario: NewsAsset (ATC) **Technical Space: *Source code (GPL)***		
	Sub-category 1: C#	**Sub-category 2: ASP.NET (C#)**	**Sub-category 3: Javascript**
Origin	Manual	Manual	Manual
Purpose	Application code (5 projects)	Application code	Application Code
Consumer	System	System	System
Organization	Structured code	Structured code	Structured code
Nature	Static	Static	Static
Size	"Large": 500 source files, 230,000 lines	"Small": 48 source files, 27,000 lines	"Medium", 243 files, 11,000 lines
Opacity	White-box	White-box	White-box
Architectural Layer(s)	Business Logic (2 projects), Presentation (1 project), Data (1 project), Communication (1 project)	Presentation	Presentation
Environment	Development: Microsoft Visual Studio Running: .NET Framework (v.3.5)	Development: Microsoft Visual Studio Running: MS IIS, .NET Framework (v.2)	Development: Microsoft Visual Studio Running: Browser

	Use Case/Scenario: NewsAsset (ATC) **Technical Space: *DSL***	
	Sub-category 1: HTML/CSS	**Sub-category 1: Oracle**
Origin	Mostly Manual, some Generated	Manual
Purpose	Application code	Data Management, Configuration : DB schmema, 239 tables, 15 triggers, 5 functions, 9 stored procedures
Consumer	System	System
Organization	Structured code	Structured model
Nature	Static	Static
Size	"Small": 34 html files, 1 css file	Medium
Opacity	White-box	White-Box

Architectural Layer(s)	Presentation	Data
Environment	Development: Microsoft Visual Studio Running: Browser	Oracle RDMBS 11g

	Use Case/Scenario: NewsAsset (ATC) **Technical Space:** *APIs/Libraries*		
	Sub-category 1: External .NET Component Libraries	**Sub-category 2 Active X**	**Sub-category 3: External Libraries**
Origin		Mostly Manual	
Purpose	Code	Code	Code
Consumer	System	System	System
Organization	Structured	Structured	Structured
Nature	Static	Static	Static
Size	Small	Small	Small
Opacity	Black Box	White-Box	Black Box
Architectural Layer(s)	Presentation	Presentation	Presentation, Communication
Environment	Development, Compiler: Microsoft Visual Studio Running: .NET Framework (v.3.5)	Development: Borland Delphi Compiler: Microsoft Visual Studio Running: MS Windows	Development: Unknown Compiler: Microsoft Visual Studio Running: MS Windows

	Use Case/Scenario: NewsAsset (ATC) **Technical Space:** *XML*	
	Sub-category 1: Config/Project files	**Sub-category 2 Services**
Origin	Genarated, Manually edited	Manual
Purpose	Configuration	Code
Consumer	System and End-User	System
Organization	Structured code	Structured code
Nature	Static	Static

Size	"Small": 4 config files	"Small": 1 WCF Service (svc) and 1 Service Description (wsdl)
Opacity	White-box	White-box
Architectural Layer(s)		Communication
Environment	Development: Microsoft Visual Studio	Development: Microsoft Visual Studio Running: .NET Framework (v.3.5)