Extraction and Reconstruction of Enterprise Models

Mario Sánchez(✉), Julio Cesar Reyes, and Jorge Villalobos

Systems and Computing Engineering Department, School of Engineering, Universidad de Los Andes, Bogotá, Colombia
{mar-san1,jc.reyes159,jvillalo}@uniandes.edu.co

Abstract. Enterprise Models for analysis, and especially for automated analysis, should have five characteristics: they have to be accurate representations of the reality; they have to be well structured; they have to be complete with respect to their intended usage; they have to be kept up-to-date; and the cost of their construction and maintenance has to be as low as possible. In this paper we present an approach for the semi-automatic construction of enterprise models which gathers and weaves information from multiple sources such as information systems, databases, files (system's logs, source code, configuration), and previously existing models. This approach is based on modeling and metamodeling techniques, and has been implemented in a tool called EM-AutoBuilder.

Keywords: Enterprise modeling · MDE · Automatic documentation · Model analysis

1 Introduction

Enterprise Modeling (EM), the discipline and practice of building and analyzing models representing one or many concerns of an enterprise, is progressively becoming mature and widespread. The value that can be gained from doing EM is directly proportional both to the quality of the models, and the quality of the available tools and methods to perform the analyses: if models are small, have low level of detail, have low fidelity, or are unstructured (e.g., text documents), it is difficult to perform insightful analyses; on the other hand, if analyses are simple and naive, there is no point in building advanced and detailed models. A clear example of this are simulation-based analyses, which are very advanced but require high-quality models with information spanning several domains. In this paper we focus on the first concern (model quality) and make a proposal to address this problem and thus increment the value that can be gained from EM, and especially from model analysis.

The biggest issue affecting the quality of enterprise models is the elevated costs of construction and maintenance. Since building these models is typically a human-intensive task, compromises are made which go against quality. For example, the scope of the model may be limited, or its depth, or its completeness. Furthermore, the lack of widespread modeling tools usually results in the

J. Barjis and R. Pergl (Eds.): EOMAS 2014, LNBIP 191, pp. 3–20, 2014.
DOI: 10.1007/978-3-662-44860-1_1

usage of inadequate technologies (e.g., text processors, spreadsheets, unstructured diagrams) that produce models that are impossible to process. Finally, enterprises are ever changing, and thus enterprise models must be permanently maintained. However, the current situation makes it very expensive to make these permanent upgrades.

The hypothesis that we attempt to validate with this work is that a lot of the information that should be included in an enterprise model can be obtained and structured in a *largely automated* way. For example, it should be possible to gather information about the architecture of deployed information systems, combine it with information about the enterprise coming from structured documentation, and finally enrich it with real statistics about its behavior coming from systems' log registries. The result of this, would be a comprehensive enterprise model which can be easily kept updated. Automating steps in the process of collecting and structuring the models, should also remove potential sources of errors and inconsistencies. Ultimately, this all should lead to increasing the quality of enterprise models and the value that can be obtained from them.

To validate this hypothesis, we designed an approach and architecture for building enterprise models using information obtained from different sources. This approach was implemented in a tool called EM-AutoBuilder, which has already been tested in an internal case study, and is now starting to be applied in real scenarios.

The structure of the paper is as follows. Section 2 discusses in more detail what Enterprise Models are, the possible sources of information to build them, and the current state of art in automatic construction of enterprise models. Section 3 introduces a scenario to illustrate the solution. Then, the proposed approach and its implementation are presented and illustrated in Sect. 4. Section 5 concludes the paper and discusses the outlook for the presented work.

2 Automatic Construction of Enterprise Models

An Enterprise Model is a representation of elements of an enterprise that typically belong to different domains (e.g., business processes, business and regulatory environment, organizational structure, or information technology). The cost of building enterprise models varies depending on two factors: the required level of detail, and the scope that the model should cover (i.e., how much of the enterprise and how many domains should be represented in the model). Before making a commitment to build a model, these two variables should be analyzed and balanced against the cost of construction, and against the benefits that the model can eventually bring. To further complicate the matter, these benefits are not intrinsic to the model, but depend on how it is used. For instance, it can be used for (i) documentation or communication purposes; (ii) as a way to increase understanding of the enterprise; (iii) as the starting point for analyzing the current situation of the company; (iv) or to evaluate transformation projects.

Given the aforementioned potential uses of enterprise models, there are a number of desirable qualities that said models should posses. The first and most

important one is *accuracy*: a model that does not reflect the reality cannot answer truthfully any kind of answer, and thus is useless. On top of that, making decisions based on incorrect information can be worse than not making the decisions at all. A second, related quality is that a model has to be *up to date* (old information is just a particular kind of incorrect information, and just as risky as inaccurate information). Next, an enterprise model should to be *structured*: while models are frequently represented using unstructured means such as documents, diagrams, and spreadsheets, their real value can only be achieved if they are built around well defined structures and using representations that favor automatic processing. The fourth quality is *completeness*: a model should be complete with respect to its intended usage, and it should not lack information that it is expected to have. Finally, the *cost of building and maintaining* an enterprise model should be as low as possible; otherwise, it will probably be incomplete or will quickly cease to be up to date.

To build an enterprise model, it is critical to discover the potential sources of information. The typical sources include personnel of the company that deliver the information by means of interviews; manuals and documentations about processes, procedures, organizational structures, and responsibilities; and technical documentation about applications and technological elements. Other powerful sources of information are the Information Systems (IS) themselves, which are usually capable of providing structural and behavioral information. This can be achieved by direct observation of the IS technological components (interfaces, configuration files, source code, etc.), or by studying the relevant documentation and architectural documents. Furthermore, observing the storage systems and logging records of those IS provides valuable information to build models that are also behavioral, instead of purely structural.

Automatically building enterprise models is ideal, considering the discussed qualities. However, most sources of information are not suitable for this: only very well structured documentation (e.g., based in a Quality Management System Software), and the information systems themselves, can be automatically studied in order to build accurate, up to date, structured, and complete models, at relatively low costs.

In the past, some projects have made attempts at automatically building models with different levels of detail and focusing on particular domains. The work of Buschle et al. [1] reconstructs models of enterprise architecture based on network scanning and retrieving, in a graph-based structure, the main components of the application infrastructure of information systems. In this work they collected information using network analysis tools and vulnerability examination tools, and the results were application deployment models. Similarly, the work of Binz et al. [2] combined a manual and a semi-automatic approach for model construction. The automatic part was limited to network assets discovery (e.g., operative systems, DBMS, Application Servers), and the result was a graph representing the topology of enterprise systems.

Other works have tackled the problem from a source code perspective. Instead of obtaining the information from the systems already deployed, they have

collected the information from their source code. These works include MoDisco [3], the work of Schmerl et al. [4], and the work of Song et al. [5]. MoDisco is an extensible framework that is targeted to support software modernization. Moreover, as part of this, MoDisco is capable of analyzing artifacts such as source code, database structures, and configuration files, in order to create models representing existing systems. The work that we present in this paper borrows some architectural ideas from MoDisco, especially with respect to the extensibility capabilities.

On the other hand, the work of Schmerl et al. combines the analysis of source code with the analysis of low-level system events to obtain a representation that relates events that happen in specific use cases of the system, in a specific runtime scenario. Finally, the work of Song et al. [5] analyzes application API and calls, and uses this information to reconstruct the structure of the systems.

Our approach differs from the above mainly because they focus on a single domain, whereas we intend to create multi-dimensional enterprise models. We later show that our enterprise models are produced by combining static information obtained from documents, structural information coming from the observation of deployed information systems and/or their source code, and from the capture of behavioral information that is usually stored in such information system's logfiles.

3 An Illustrative Case Study

To illustrate the work presented in this paper, a case study is now introduced. This case study is a fictitious but realistic company called BPO Los Alpes[1], which offers outsourced services. In particular, we are going to focus on services for creating and managing donation campaigns. These include managing information, publicity, and press releases about the campaigns; gathering information about potential donors; contacting the donors; collecting payments; and tracking the success or failure of the campaigns.

The BPO bases its operation on the following internal information systems, which are illustrated in Fig. 1.

- **Donations System:** this is the system that handles donation campaigns and it is responsible both for managing the business logic as for storing the relevant information. This includes information about donation campaigns, potential and actual donors, donations, payments, and certificates. Information is stored in a relational database (DB DON).
- **CCA:** this is the system used by the call-center of the BPO to contact the potential donors and register information about the outcomes of the calls.

[1] We call the company fictitious because it does not really exist or offer any service. However, it is realistic because it was modeled after real companies that provide similar services, and because its information systems are completely build and operational. We use this, and other similar scenarios, to support research and initial prototypes, and also for educational purposes in the courses we teach.

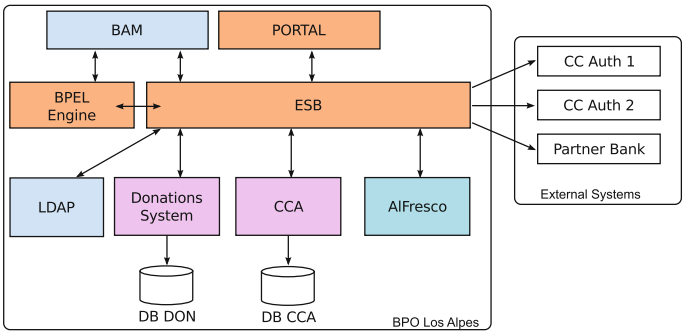

Fig. 1. Architecture of the case study

This system handles the contact-center agents, the lists of pending calls and their assignment, and an annotated registry of calls. Information is stored in a relational database (DB CCA).

- **AlFresco (ECM):** this is the enterprise content management system used to store the certificates for donors (which have legal validity for tax-exemption purposes), and to publish information about donation campaigns. The storage of its information is entirely handled by AlFresco.
- **LDAP:** this is the system that stores information about the users of the different applications and is capable of granting or denying usage privileges. All the applications use this LDAP system for authentication purposes.

In addition to these internal systems, the BPO depends on consuming services from payment transaction systems provided by allied banks and by credit card authorization systems. There are also three elements used for integration and coordination purposes: these are a BPEL engine, an Enterprise Service Bus (ESB), and a Portal. Finally, there is a BAM system to calculate and display indicators based on information obtained from the different applications, the bus and the BPEL engine.

In addition to this, there are several documents and artifacts describing the structure and operation of the BPO. Some of them are unstructured documents created with a common text editor, and thus are very difficult to process; others are diagrams and models created with specialized tools (i.e., ArchiMate and BPMN editors) and thus can be processed with relative ease.

4 Automatic Documentation: EM-AutoBuilder

The goal of this section is to present an approach to build enterprise models with the qualities that were discussed in Sect. 2. This approach has been implemented in a tool called EM-AutoBuilder, has been validated using the scenario presented in Sect. 3, and is currently starting to be used in a real scenario.

The approach was designed to address four critical requirements. The first one required models to be constructed in a way as automatic as possible. Therefore, the approach is targeted to collecting information from sources that can be automatically processed, i.e. information systems and well-structured documents and models. The second requirement was supporting heterogeneity. That is, it should be possible to gather information from various sources that have different structures, support different purposes, and are built around different technologies. These sources may also include documents, further increasing the complexity and heterogeneity. The next requirement is also closely related: the approach should be extensible, in order to be applicable to new sources of information. The fourth and final requirement regards the output of the approach: it should be a single, integrated artifact that can be processed or loaded into other tools that provide analysis capabilities.

Given these requirements, we designed the approach that is illustrated in Fig. 2. The core of this approach is a component that hosts several *configurable extractors* and processes their outputs. Extractors are independent components, and each one is capable of connecting to some specific kind of source to extract information. This information is returned to the core structured as a model. After each extractor has provided one or several models, the core has to process them to build an integrated one. However, the models may conform to different metamodels, and the core cannot know those metamodels before hand. Therefore, each extractor also has the capacity to provide the metamodels that it uses to build the models.

A final point in the strategy is the capacity of the core to weave the models, based on the weaving of the metamodels. It has been shown that completely automating the latter procedure is not feasible [6]. Therefore, we decided to let this step under the responsibility of the user, which has to specify the necessary relations between the metamodels. Finally, models are woven based on the information that the user provided, and a single tuple <metamodel, model> is produced.

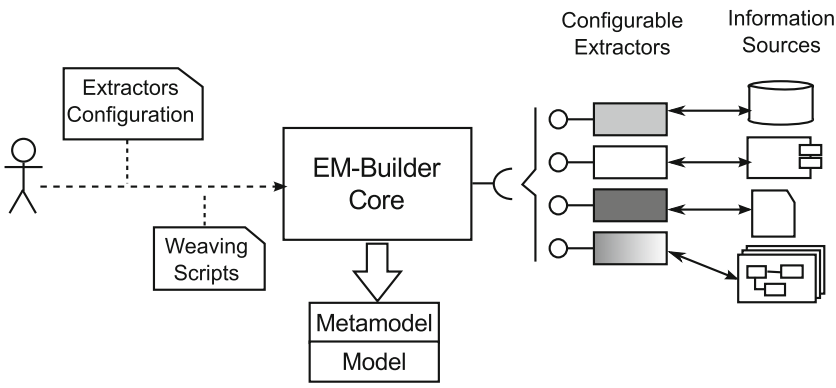

Fig. 2. Overview of EM-AutoBuilder architecture

The approach has been implemented in a tool called EM-AutoBuilder. This tool is based on Java and EMF[2] technologies because this makes it compatible with many tools for model processing, analysis, and visualization. Moreover, it should be possible to build similar tools using other technologies that match other tool environments. The following sections provide more details about each of the elements involved and about the responsibilities of each one.

4.1 Individual Extraction of Information

The first step in the automatic creation of enterprise models is to collect information from all the relevant data sources. However, these sources have a level of diversity that makes it impossible to have an universal component capable of querying them all. Even in the simplest cases, such as extracting information from relational databases, small technical differences in the way of managing the schema structure may prevent the same component for querying any RDMS.

In EM-AutoBuilder, the solution for this diversity problem was to build a framework and define an abstract component, the *extractor*, that is capable of querying systems to obtain and structure information. Concrete extractors are built using the framework, and they only share an API and some libraries to assemble and process models. The API that all concrete extractors implement has the following two main methods:

- `configure`: this method receives a set of Java properties with the information that the extractor needs to find its data source. For example, in the case of an extractor to collect information from a relational database that is accessed through JDBC, the properties include the name of the driver to use, the url to locate the DB, and the username and password to connect.
- `collectInformation`: this method uses the configuration information to obtain information and structure it in an EMF model. The output of this method is a tuple containing a model and a metamodel that are used subsequently by the EM-AutoBuild core.

It is worth noting that the metamodel that each extractor returns is sometimes calculated as part of the process of collecting information, i.e. it cannot be known a priori and it depends on the information obtained from the source. The reason for this has two parts. Firstly, the relation *instanceOf*, between *instances* and their *types* naturally exists within some domains. Secondly, both the types and the instances should appear in the same model, and the structure of the instances should conform to the restrictions imposed by their corresponding types. Unfortunately, this is not something typically supported by modeling frameworks, and thus it would require ad hoc solutions in each case. We experimented with some alternatives to manage these situations, but they required capabilities not supported by EMF, such as deep instantiation and potencies [7]. Ultimately, the chosen strategy was also adopted because it simplifies the weaving process.

[2] EMF - Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/.

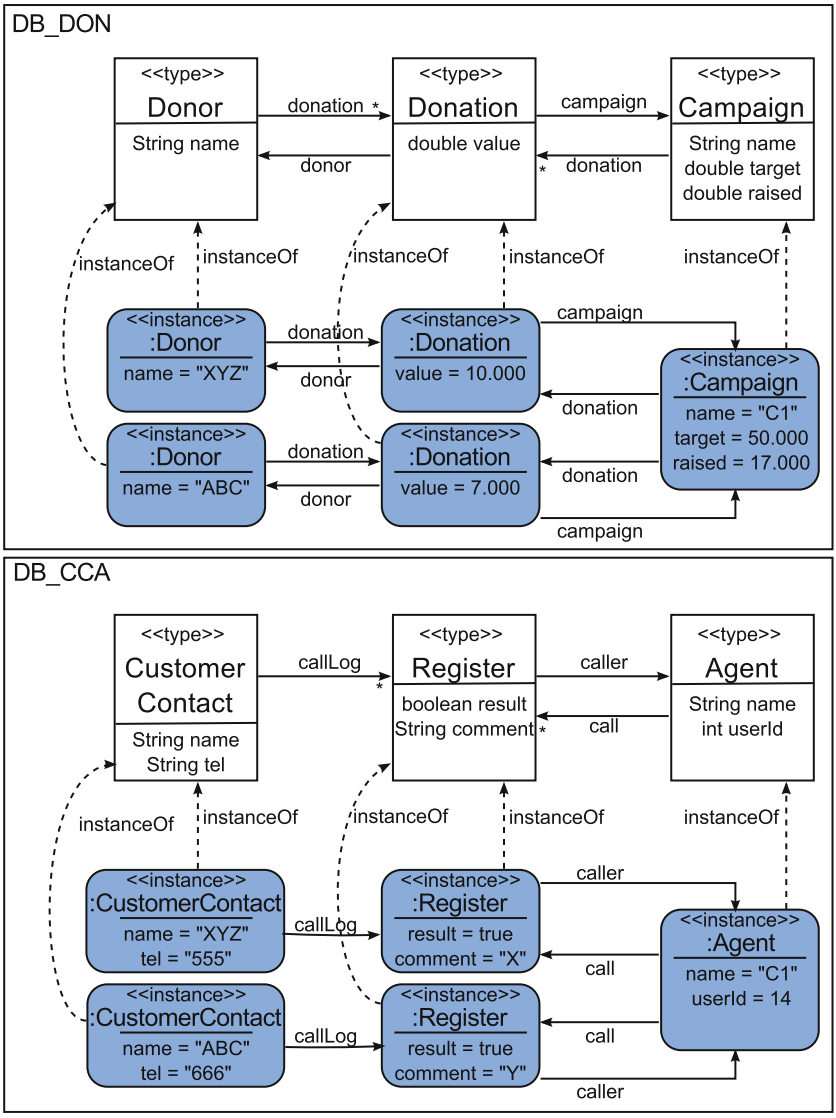

Fig. 3. Models obtained from the CCA and Donations System database

Figures 3 and 4 present a fragment of the results obtained from applying extractors to the BPO case study. In this case, the extractors utilized were capable of collecting information from relational databases, and they were applied to the Donations System database and to the CCA database. The results obtained were one model and one metamodel for each system.

Figure 3 presents the two models: on the top of the figure, there is the model obtained from analyzing the Donations System database; on the bottom of the

figure, there is the model obtained from analyzing the CCA database. Both models include two kinds of elements. Firstly, there are elements that have been represented as classes and are marked with the `<<type>>` stereotype. These elements appear in the model because the databases contained tables with those names, and because it was useful to include those concepts in the models. Secondly, there are elements represented as objects which received the stereotype `<<instance>>`. These elements appear in the model because there were corresponding registers in the tables. The relation `instanceOf` between an `<<instance>>` and a `<<type>>` is of *ontologic* nature [8], and it is a reflection of the relation between a registry in the database and the table that contains it. Furthermore, the elements marked as `<<type>>` also appear on the metamodels, thus making it possible to have *linguistic* instances of them in the models. Since the same extractor was used to collect data from both databases, the structure of both models is very similar.

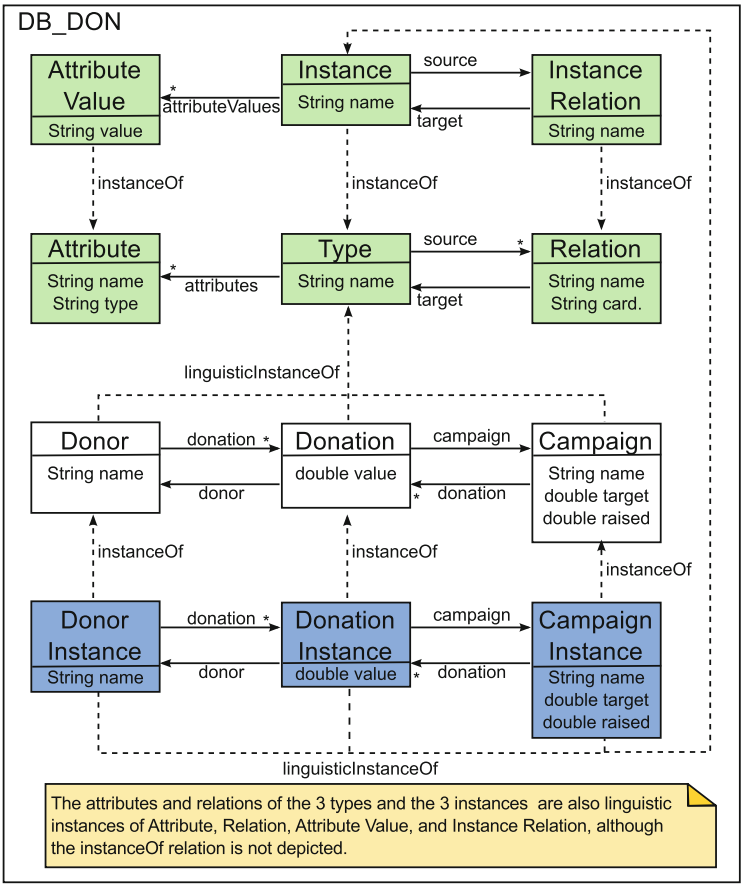

Fig. 4. Metamodel obtained from the Donations System database

Figure 4 depicts the metamodel obtained by the extractor when it analyzed the Donations System database. The model on the top of Fig. 3 conforms to this metamodel, which can be divided in three parts. The top part is composed by elements that are generic in nature and are usually found in generic meta-metamodels (e.g., MOF, Ecore): `Type`, `Attribute`, `Relation`, `Instance`, `AttributeValue`, and `InstanceRelation`. These elements appear in every metamodel created by an extractor, independently of the information source analyzed. In the models, the elements with stereotype `<<type>>` are instances of `Type`. The second part of the metamodel is composed by the elements `Donor`, `Donation`, and `Campaign`. These are all *linguistic instances* of `Type`, and they match the `<<types>>` that were previously described in the model. Finally, the third part of the metamodel is formed by `DonorInstance`, `DonationInstance`, and `CampaingInstance`. In the models previously described, the elements with stereotype `<<instance>>` are instances of the elements in this third part of the metamodel.

There are several reasons that explain the apparent complexity of this metamodel. First of all, relations of ontological and linguistic instantiation are put together instead of creating a separated meta-meta-model. This strategy was selected mainly because current, mainstream modeling tools only support two modeling levels (model and metamodel) and thus making it impossible to introduce a meta-meta-model. Furthermore, metamodel weaving (or type-level weaving), which will be discussed in the next section, is easier to perform when some elements are constant (in this case, the meta-meta-elements). Based only on this, one could wonder whether the rest of the metamodel is really necessary, since `<<types>>` are found in the models. The answer to this is twofold. On the one hand, the case of database analysis is special, because tables determine conceptual *types* which are important also in the model level; analyzing other types of information sources, such as BPEL specifications, does not produce any `<<types>>`, just `<<instances>>`. On the other hand, for many of the actions that we want to perform on the models, it is important to have a *domain metamodel* instead of a *generic* or *linguistic* one. On top of making analysis and queries much more powerful, it simplifies the specification of operations. This is analogous to defining a specific data model for a relational database instead of using a generic data model. Finally, an unexpected element in the metamodels are the attributes and relations found in the `Instances`. This is really duplicated information that can also be found in the types. The explanation for this lays again in the limitations of current modeling tools: they cannot support a mechanism similar to *potencies* [7] to force `<<type>>` to have certain attributes or relations. However, putting the attributes and relations in the `Instances` accomplishes the same goal.

The extractor used to collect this information works as follows. Firstly, it queries the database catalog to know the structure of the schema, including the tables, columns and keys. This information is stored in a temporal model, with a fixed structure. Afterwards, the extractor queries each table to obtain information about the registers, and stores that information in the temporal model.

Finally, the obtained model is transformed to obtain the definitive metamodel, which depends on the original structure of the database schema, and the definitive model.

In addition to the relational database extractor, the BPO case study required other extractors for a number of sources such as the BPEL engine, the BPEL process description files, the ESB Engine (for services, routes and transformations), the LDAP service, BPMN models (built with Bizagi Process Modeler[3]), and ArchiMate models (built with the Archi editor[4]), among others.

There is a further kind of extractors that stands apart to the ones that have been described: extractors that collect information about behavior. What makes these extractors special is that, instead of focusing on structural information, they focus on information about the behavior of elements in the architecture. For example, they can observe the execution logs of a BPEL engine and collect information about the frequency of execution of each process, the frequency of each path within the process, and the frequency of errors. This capacity adds a new dimension to enterprise modeling that is seldom available, and makes it possible to use the resulting models for purposes such as dynamic analysis and simulation.

4.2 Cross-Domain Types and Instance Weaving

Since extractors are completely independent, it is only logical that their outputs are completely independent as well. However, a great part of the value of an enterprise model lies in the identification and *reification* of relations between domains. Unless these relations are materialized, it is impossible to perform analysis involving multiple domains.

In order to obtain a single enterprise model, our approach uses two kinds of model-weaving techniques:

1. **Weaving of types:** this is the kind of weaving performed between the metamodels and also between elements marked with the `<<type>>` stereotype in the models.
2. **Weaving of instances:** this is the kind of weaving performed between elements marked with the `<<instance>>` stereotype.

Combined, these two types of weaving are capable of producing a unified tuple containing an enterprise model and its metamodel. The rest of this section presents in more detail these two kinds of weaving. In EM-AutoBuilder they are implemented on top of a model-weaving machine that is controlled using two different textual languages. These languages are enough to demonstrate the expressiveness and capacity of the machine, but we are in the process of developing a more usable interactive interface.

[3] Bizagi Process Modeler homepage: http://www.bizagi.com/index.php/en/products/bizagi-process-modeler.

[4] Archi homepage: http://archi.cetis.ac.uk/.

Type-Level Weaving. The first type of weaving that we are going to describe is the one that is done between elements marked as `<<type>>`. This process is performed at two levels: on the one hand, it is performed at the metamodel level, to obtain a single metamodel for the enterprise model; on the other hand, it is performed at the model level, to describe the same relations within the enterprise model. The process is performed at both levels following a single specification (i.e., it is not necessary to write one weaving script for each level), which has to be written by a modeler after inspecting the types produced by the extractors. Ideally, this task should be automated, but past experiences show that metamodel-weaving always requires human intervention [6]. Furthermore, this is a task that has to be done only when a new extractor is introduced.

The type-level weaving specification is written using a language based on Fusion [9]. Fusion is a simple language and a platform for weaving metamodels, which we developed for a previous project. Compared to other approaches, such as ATL, Fusion is extremely simple. However, it is powerful enough for its intended purpose, and it is more than enough for specifying type-level weavings. In fact, we only need to use two of the operators included in Fusion: `mergeEntities` and `newLink`. The script in listing 1.1, which is based on the example presented in Sect. 4.1, illustrates the language. This script is processed by the weaving engine that is part of the EM-AutoBuilder Core, and which is derived from the Fusion Weaver.

```
1 mergeEntities Donor {
2   entity1 = DB_DON.Donor
3   entity2 = DB_CCA.CustomerContact
4 }
5
6 newLink responsible {
7   source = DB_DON.Donation
8   target = DB_CCA.Agent
9   containment = false
10  minCard = 0;
11  maxCard = 1;
12 }
```

Listing 1.1. Example of type-level weaving

The example has two parts which depend on the metamodels called DB_DON and DB_CCA. The former refers to the metamodel produced by the extractor that explored the database of the Donations System. The latter refers to the metamodel produced by the extractor that explored the database of the CCA system. In the first part of the example, it is specified that the types `Donor` and `CustomerContact` must be merged into a single type, in the resulting metamodel, also called `Donor`. This kind of operation is used when the same concept appears in two different domains and must be merged. This results in the creation of a new type with a union of the attributes and relations of the original types. In this case, the new type is called `Donor`, has attributes `name` and `tel`, and has relations to `Donation` and `Register`.

The second part of the listing specifies a new relation called `responsible` between the type `Donation` and the type `Agent`. In this case, the relation is added because the modeler wants to make explicit the relation between a certain donation and the agent in the call center that got it. Finally, the types in the source metamodels that are not mentioned in the Fusion script, are simply copied into the combined metamodel.

While the above example is rather simple, it illustrates all the capacities required to create a unified metamodel. In the BPO case study we specified many other relations between elements from different domains. Some of these relations are represented in Table 1. In this table, the types in the columns and rows have been grouped by the domain that they represent, and an `X` has been included when the types have been related in the unified metamodel. For example, a new relation has been added between `BPMN/Process` and `BPEL/Process` to represents the fact that some BPEL processes are automated versions of BPMN processes. Similarly, the `X`s between `BPEL/Partner Link` and `Application/Service` and `External Application/Service` serve to relate BPEL processes and the applications that they rely on, by means of the processes' partner links and the services that they are bound to.

Instance-Level Weaving. The elements described in the previous section are only useful to weave elements marked with the `<<type>>` stereotype. Now we show how elements marked with the `<<instance>>` stereotype are woven in order to produce the expected unified model. In this point, it is important to note that the potential number of `instances` is much larger than the potential number of `types`. Therefore, it is important to have a largely automated process to produce the final woven model. Furthermore, it should be possible to easily run the instance-weaving scripts whenever the model has to be updated.

The language to describe the weaving process between `instances` is a reflection of the language used at the type level, and thus it has two weaving instructions. The first instruction, `mergeInstances`, is used to specify which instances have to be combined, following the merging of their types. The script in listing 1.2 specifies that `Donors` and `CustomerContact` should be combined into a `Donor` whenever the attribute `name` of the former matches the attribute `name` of the latter.

```
1 mergeInstances Donor where
2   (entity1.name == entity2.name)
```

Listing 1.2. Example of usage of instruction mergeInstances

The second instruction in the instance weaving language, `relateInstances`, is used to create relations between instances, based on the new relations created in the metamodel. The script in listing 1.3 continues the example and illustrates this instruction.

Table 1. Selected cross-domain relations in the BPO case study

	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	e13	e14	e15	e16	e17	e18
DB_DOM																		
e1: Donor				x						x	x							
e2: Donation						x				x	x							
e3: Campaign										x	x							
DB_CCA																		
e4: CustomerContact	x									x	x							
e5: Register										x	x							
e6: Agent		x								x	x		x	x				
BPEL																		
e7: Partner Link															x	x		
e8: Process											x				x	x	x	
e9: Invoke															x	x		
e10: Execution Log	x	x	x	x	x	x					x	x			x	x	x	
BPMN																		
e11: Process	x	x	x	x	x	x				x							x	
e12: Role										x				x				x
LDAP																		
e13: User						x				x								
e14: Org. Unit						x						x						
Internal Application																		
e15: Service							x	x	x	x								
External Application																		
e16: Service							x	x	x	x								
ArchiMate																		
e17: Business Service								x		x	x							
e18: Business Role												x						

The semantics of this script is as follows. Firstly, the model weaving engine has to lookup all the instances of the type `Donation`, which was the `source` type when the new relation was added. Then, for each `Donation` the engine has to find `Agents` using the expression in the second line of the script. This expression is written with the Cumbia Navigation Language [10], which is based mainly on the Object-Graph Navigation Language (OGNL)[5], the JSP Expression Language

```
1 relateInstances Donation.responsible ->
2   #this.donor.callLog.caller
```

Listing 1.3. Example of usage of instruction relateInstances

[5] OGNL Language Guide: http://www.opensymphony.com/ognl/html/LanguageGuide/index.html.

[11], and the MVEL[6]. The expression in the example specifies a path from each `Donation` to 0 or more `Agents`, following the relations `donor`, `callLog`, and `caller`.

Instance weaving instructions are executed in their order of appearance. Thus, when listing 1.3 is run, the `Donors` have already been merged with the `CustomerContact`, thus making `donor.callLog` a valid relation. Figure 5 depicts the resulting merged model.

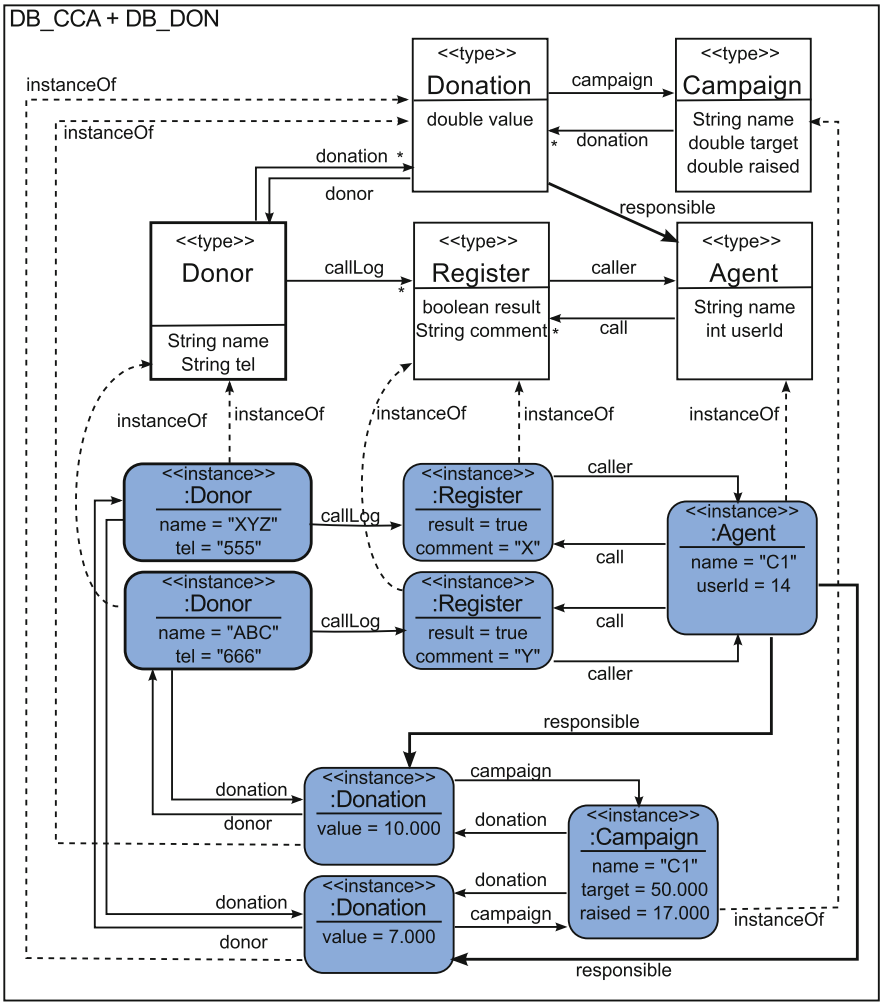

Fig. 5. Model obtained from merging the CCA and Donation models

[6] Language Guide for MVEL 2.0: http://mvel.codehaus.org/Language+Guide+for+2.0.

When all the instance-weaving instructions have been executed, the metamodel and the enterprise model are finally ready to be used.

4.3 Compatibility Concerns and Using the Models

The result of the weaving process is a model and the metamodel it conforms to. By themselves, these artifacts are not really valuable. However, with adequate tools a lot of value can be gained from them.

In consequence, a key requirement for EM-AutoBuilder was to produce models and metamodels that were compatible with other tools. In the end, the tool was given the capacity to export the artifacts to XMI and ecore. Thus, the enterprise models obtained are easily compatible with other tools based on EMF, and even with tools that are capable of using XMI models. Furthermore, the models can be converted from XMI into other representations whenever it is necessary.

Figure 6 presents a screenshot of a fragment of the model obtained during the case study. In the figure, the model was loaded into a tool for supporting visual analysis of models [12]. Within this tool, it is possible to configure how certain characteristics of the model and its elements should be mapped to visual attributes. For example, colors can be mapped to domains, sizes can be mapped to the number of relations, and forces (which the tool uses for calculating layouts) can be mapped to topological proximity. Other tools that can use these enterprise models include generators of documentation, quantitative and qualitative analysis tools (e.g., see [13]), and reporting tools.

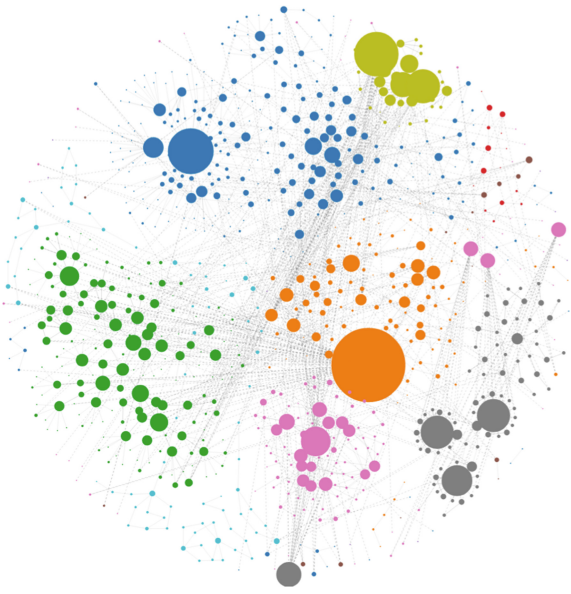

Fig. 6. Visualization of a fragment of the BPO model

5 Conclusion

In this paper we have studied the problematic of building enterprise models that are accurate, complete, structured, and up-to-date. For this purpose, we presented an approach to build and maintain enterprise models in a largely automated way, and we presented the tool that implemented the approach: EM-AutoBuilder. EM-AutoBuilder provides fixed and variable elements. The variable part is represented by a framework and an API for the implementation of extractors that connect to specific information sources. Among the fixed elements, there is a core that hosts the extractors, weaves the models and metamodels, and produces a single model and a single metamodel. This core is not completely automatic: it requires some input from a modeler in order to know how to put together the different pieces that should form the complete model.

In Sect. 2 we stated five desirable qualities for enterprise models. The presented approach, and the EM-AutoBuilder, are helpful for obtaining all of these qualities: the resulting models are *accurate*, because their base information comes from trustworthy sources; they are *well structured* because they conform to known metamodels; they are *complete* with respect to the information available as long as the right extractors are used; finally, the *cost* of building them may not be low if new extractors have to be developed from scratch, but the maintenance cost to keep them up to date is low compared to traditional EM methods.

The presented approach and its implementation have been tested so far with a case study, but they are starting to be used in project with a big company. Within this project, we are also building a reusable library of extractors. These will include extractors for source code, and for performing security vulnerability analysis, which will produce enterprise models *annotated* with security and risk information.

References

1. Buschle, M., Holm, H., Sommestad, T., Ekstedt, M., Shahzad, K.: A tool for automatic enterprise architecture modeling. In: CAiSE Forum. CEUR Workshop Proceedings, vol. 734, pp. 25–32. CEUR-WS.org. (2011)
2. Binz, T., Leymann, F., Nowak, A., Schumm, D.: Improving the manageability of enterprise topologies through segmentation, graph transformation, and analysis strategies. In: Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference (EDOC '12), pp. 61–70. IEEE Computer Society, Washington, DC (2012)
3. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '10), pp. 173–174. ACM, New York (2010)
4. Schmerl, B., Garlan, D., Yan, H.: Dynamically discovering architectures with DiscoTect. In: Proceedings of the 10th European Software Engineering Conference held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13), pp. 103–106. ACM, New York (2005)

5. Song, H., Huang, G., Xiong, Y., Chauvel, F., Sun, Y., Mei, H.: Inferring meta-models for runtime system data from the clients of management APIs. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part II. LNCS, vol. 6395, pp. 168–182. Springer, Heidelberg (2010)
6. Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: a generic model weaver. In: Proceedings of the 1ere Journée sur l'Ingénierie Dirigée par les Modèles (IDM05) (2005)
7. Atkinson, C., Kuhne, T.: Rearchitecting the UML infrastructure. ACM Trans. Model. Comput. Simul. **12**(4), 290–321 (2002)
8. Kuhne, T.: Matters of (meta-) modeling. Softw. Syst. Model. **5**(4), 369–385 (2006)
9. Florez, H., Sánchez, M., Villalobos, J.: EnAr-Fusion. A Tool for Metamodel Composition. Technical Report, Universidad de los Andes, ISIS-01-2012 (2012) http://backus1.uniandes.edu.co/~enar/dokuwiki/doku.php?id=fusion
10. Sánchez, M.: Executable Models for Extensible Workflow Engines. Ediciones Uniandes, Bogotá (2012)
11. Armstrong, E., Ball, J., Bodoff, S., Carson, D.B., Evans, I., Green, D., Haase, K., Jendrock, E.: JSP expression language. In: The J2EE 1.4 Tutorial, ch. 12: JavaServer Pages Technology, pp. 499–506. Sun Microsystems Inc., Santa Clara (2005)
12. Naranjo, D., Sánchez, M., Villalobos, J.: Connecting the dots: examining visualization techniques for enterprise architecture model analysis. In: Grabis, J., Kirikova, M., Zdravkovic, J., Stirna, J. (eds.) Short Paper Proceedings of the 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM 2013), vol. 1023, pp. 29–38. CEUR-WS (2013)
13. Ramos, A., Gomez, P., Sánchez, M., Villalobos, J.: Automated enterprise-level analysis of ArchiMate models. In: Bider, I., Gaaloul, K., Krogstie, J., Nurcan, S., Proper, H.A., Schmidt, R., Soffer, P. (eds.) BPMDS 2014 and EMMSAD 2014. LNBIP, vol. 175, pp. 439–453. Springer, Heidelberg (2014)