Applied Ontology 0 (2015) 1–0
IOS Press

# Requirements Engineering for the Design of Conceptual Modeling Languages

*A Goal- and Value-Oriented Approach*

Sybren de Kinderen [a,*], and Qin Ma [b]
[a] *University of Luxembourg, 6, rue Coudenhove-Kalergi, Luxembourg-Kirchberg, Luxembourg*
*E-mail: sybren.dekinderen@uni.lu*
[b] *Public Research Centre Henri Tudor, 29, rue Coudenhove-Kalergi, Luxembourg-Kirchberg, Luxembourg*
*E-mail: qin.ma@tudor.lu*

**Abstract.** Conceptual modeling languages are purposeful artifacts, hence their design should also start from the purpose that they serve. Such purposeful design addresses the requirements engineering concern of a language specification being aligned with the goals of its users. Thereby relevance of the language is ensured, instead of developing a language for language's sake. We posit that this addresses some known issues that are due to a misalignment between a language's specification and the goals of its intended users.

In this paper, we introduce vGREL, a goal- and value-oriented approach for purposeful language development. vGREL helps language engineers to start the design of conceptual modeling languages with requirements engineering exercises. To this end vGREL provides (1) a purpose driven requirements engineering process for language design; (2) a value profile for the Goal-oriented Requirements Language (GRL) to enable analysis and reasoning during the process and capture its results; and leverages (3) the software tool support of GRL for decision making during language design. To illustrate vGREL, we apply it to a case study on responsibility-based access rights management. Furthermore, we present reflections on vGREL from the language engineer involved in the case study.

Keywords: modeling language design, requirements engineering, goal modeling

## 1. Introduction

Conceptual modeling languages are used for a variety of purposes, such as information system design, analysis, and communication (Moody, 2005; Thalheim, 2011). As conceptual modeling languages are purposeful artifacts (Thalheim, 2011; Bjekovic et al., 2014; Bubenko Jr. et al., 2010; Rothenberg, 1989), naturally, an important measure of success of a conceptual modeling language is the degree to which it serves its purposes. As a consequence, we argue that the design of a conceptual modeling language should start from *requirements engineering* steps that clarify the purpose it will serve, and elicit the value/utility the language will provide to achieve the purpose. We posit that this addresses some known issues that are due to a misalignment between a language's specification and the goals of its intended users listed in (Malavolta et al., 2013), such as a language providing an excessive amount of uninteresting details, or even failing to provide the analysis capabilities of interest. The importance of requirements engineering for language design is furthermore underlined by the language development process proposed by (Frank, 2013). This process explicitly starts with requirements engineering activities to complement existing model-driven language engineering frameworks, such as the Eclipse Modeling Framework (Steinberg et al., 2008), which predominantly focus on technical language engineering considerations.

Such requirements engineering steps for modeling language design are also addressed by (Mernik et al., 2005) as the decision process for conceptual modeling languages. (Mernik et al., 2005) emphasize that decision making for conceptual modeling language design is difficult, and in their concluding remarks they call for computer-aided decision support.

[*]Corresponding author: Sybren de Kinderen, sybren.dekinderen@uni.lu.

1570-5838/15/$17.00 © 2015 – IOS Press and the authors. All rights reserved

Despite of its importance, currently there is only limited work in this direction. To the best of our knowledge, Frank (2013) is the only piece of work that explicitly targets the requirements engineering phase for the development of conceptual modeling languages. However, Frank (2013) only proposes textual guidelines, informed orientation, and coarse-grained processes for domain specific modeling language development. While this is a good first step, the informal nature of this work prevents it from providing detailed procedural guidance for requirements engineering as well as analysis and reasoning capabilities to facilitate decision making.

We also observe that languages are increasingly combined to fulfill situation-specific modeling needs. This is reflected in recent efforts to (1) extend the Enterprise Architecture language ArchiMate with security concerns (Feltus et al., 2012) and business model concerns (Meertens et al., 2012), so as to address the enterprise-wide impact of security aspects, respectively business model scenarios; (2) combine e$^3$value (a value modeling language) with process, strategy and IT application modeling, to provide conceptual modeling support for business-IT alignment across aforementioned perspectives (Pijpers et al., 2012); (3) define a transformation of e$^3$value to ArchiMate (van Buuren et al., 2005; de Kinderen et al., 2014a), to provide lightweight modeling support for analyzing the realization of a proposed value constellation. Of course this list is non-exhaustive, as each modeling effort calls for context-specific language needs. However, each of these efforts selects languages in an ad hoc manner. This indicates a lack of systematic requirements engineering support for modeling language design following a reuse strategy.

As a response in this paper we introduce an approach for value- and Goal-oriented Requirements Engineering for conceptual modeling Languages, or vGREL in short. Particularly, we introduce a requirements engineering process for designing conceptual modeling languages. The process translates a set of goals into a selection of languages, the integration of which achieves the goals. As a complement to the process, we specify a value profile for the Goal-oriented Requirements Language (GRL) (ITU-T, 2008) to enable analysis and reasoning during the process and capture its results. Moreover, we capitalize on the software tool support of GRL for decision making during language design.

Three aspects are key to our approach: (1) *intentionality*: who are the stakeholders of the modeling language, and what do they want to get out of the modeling exercise? Especially we deem it important to consider the motivations of stakeholders beyond the modeler him- or herself. For example: the Role based Access Control modeling language (RBAC, (Ferraiolo et al., 2001)) reduces the workload of IT administrators, because RBAC models allow for assigning access rights to roles instead of to individual employees; (2) *taking a value perspective on languages*: how is a language valuable to achieve stakeholder goals, both in terms of capabilities inherent to a language itself (e.g., an actor-role distinction for the RBAC language) and extra-language capabilities (e.g., industry uptake of RBAC)? (3) *reuse*: reuse of valuable language components and/or good practices from language design. If we know the purpose, can we reuse good practices from languages developed in the past?

In this paper we focus on the first two aspects: supporting language engineers in the design of conceptual modeling languages, particularly by (1) starting the design of languages from their intended purpose; (2) providing decision support during language design on the valuable capabilities that languages should provide to achieve their purpose. In addition, in line with the third aspect - reuse - we propose the use of two catalogs, a stakeholder perspective catalog and a language catalog. The first catalog stores good practices from language design exercises. The second catalog stores the profile of language components from a value perspective. We demonstrate the initial use of these two catalogs in our requirements engineering process. For future work, we foresee that these two catalogs enable reuse of good language engineering practices. Moreover, besides *requirements engineering* for modeling languages, vGREL can also potentially be used for "model archeology", i.e. as a scientific instrument to better *understand* the design decisions behind a language.

This paper is an extension of earlier work (de Kinderen and Proper, 2013; de Kinderen et al., 2014b). In this earlier work, a key idea is to treat languages as "building blocks" that can be decomposed and recomposed to provide the value required for purposes of the specific modeling exercise at hand. Moreover, we also hint that the selection of building blocks should be purposeful. In this paper, we further elaborate on the value perspective of languages by distinguishing different types of values, i.e., inherent and extra-language capabilities, and providing a structure to document them in terms of a language profile. In addition, we mature the intention-based language selection hinted in (de Kinderen and Proper, 2013; de Kinderen et al., 2014b) with a process which translates a purpose into a suitable selection of languages. The contribution of the selected languages to the achievement of the purpose is specified in a value profile for the goal modeling language GRL. Furthermore, we exploit the goal satisfaction evaluation capabilities of the jUCMNav tool for GRL so as to provide decision support for language selection to the language engineer. Finally, we use a substantive case study on responsibility based access rights management, involving feedback from a third party language engineer, to provide a practical evaluation of vGREL.

This paper is structured as follows. In Section 2, we introduce our vGREL approach for requirements engineering of conceptual modeling languages in terms of a value profile for GRL, a language engineering process, and software tool support. Thereafter, Section 3 applies the vGREL approach to a case study on responsibility based access rights management. Section 4 presents a reflection on vGREL by our case study partner. Moreover, it discusses related work. Section 5 concludes and presents future work.

## 2. The vGREL approach

The vGREL approach focuses on requirements engineering for conceptual modeling languages, less so on the more technical language design considerations such as how to perform language integration, model (de-) composition, or language federation. In more detail, we position our work according to the language development process proposed by (Frank, 2013), as depicted in Fig. 1. Our focus is on the first two steps belonging to the phase of requirements engineering, which mainly entails studying the stakeholder goals and propose a set of candidate languages whose integration fulfills these goals.

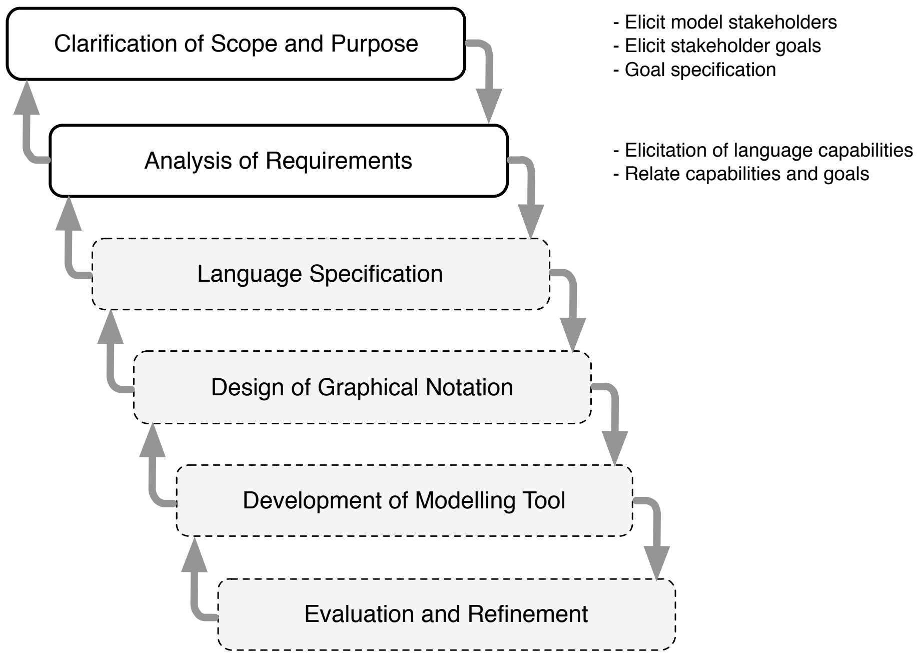
Clarification of Scope and Purpose
Analysis of Requirements
Language Specification
Design of Graphical Notation
Development of Modelling Tool
Evaluation and Refinement
- Elicit model stakeholders
- Elicit stakeholder goals
- Goal specification
- Elicitation of language capabilities
- Relate capabilities and goals

Fig. 1. Positioning the vGREL Approach in a Process for Conceptual Modeling Language Development (adapted from (Frank, 2013))

As stated in the introduction, we consider three aspects to be key for requirements engineering for language design: (1) intentionality, considering the goals of the modeling stakeholders, both stakeholders doing modeling, and those benefiting from it; (2) taking a value perspective on languages; (3) reuse of valuable language components and good practices from language design.

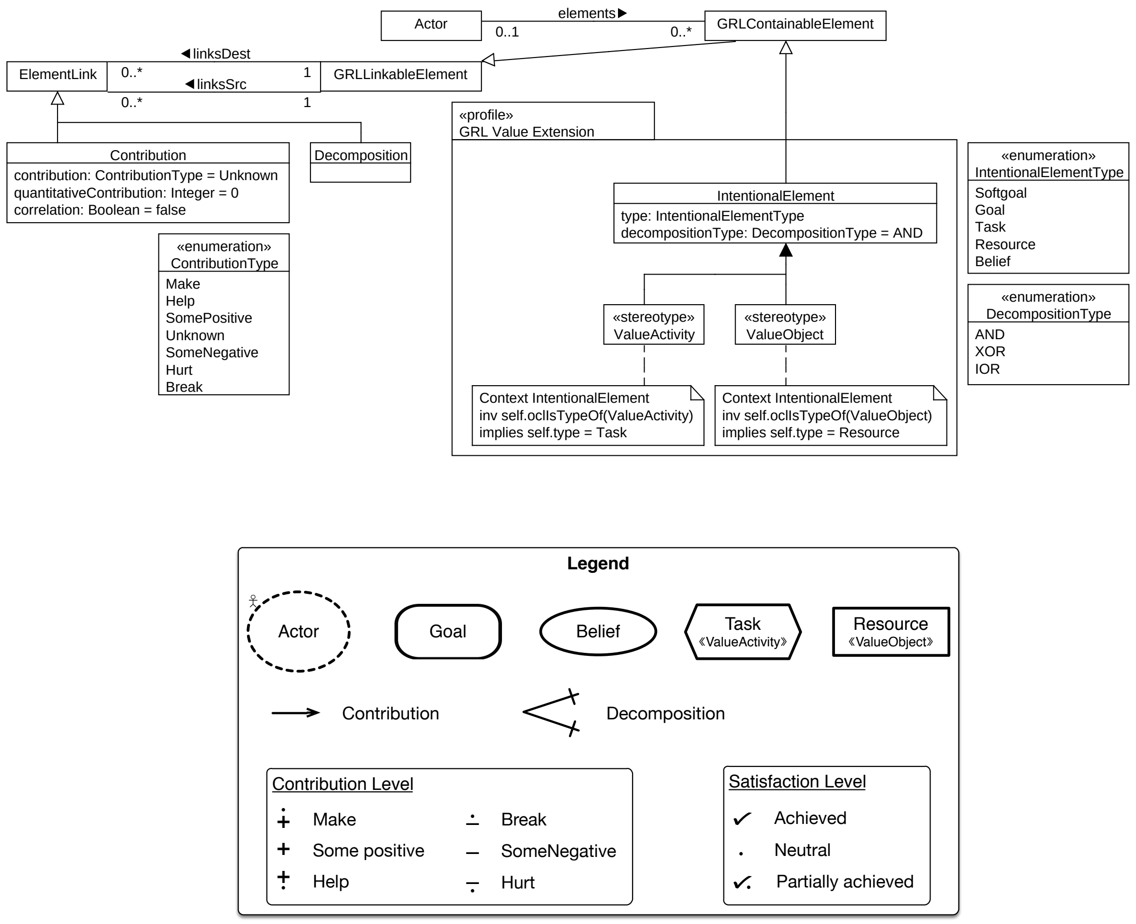

Fig. 2. The vGREL Metamodel (top) and its Concrete Syntax (bottom)

These aspects are addressed by the vGREL approach as follows. First, we introduce a requirements engineering process for conceptual languages, and a value profile for the Goal-oriented Requirements Language (GRL) to capture the result of the process. Second, we propose the use of two catalogs, a stakeholder perspective catalog and a language catalog. Third, we leverage existing tool support for GRL to facilitate decision making between alternative language engineering solutions.

## 2.1. A value profile for GRL

Fig. 2 shows the vGREL metamodel (top) and its concrete syntax (bottom). The vGREL metamodel is a UML profile of the metamodel of the Goal-oriented Requirements Language (GRL) (ITU-T, 2008). GRL enables the description and analysis of system goals from the perspective of various stakeholders, and the dependencies between them in terms of mutual contributions or conflicts. By profiling we enable the reuse of GRL features such as tool support, strategies for goal satisfaction evaluation, and the GRL concrete syntax (see the legend in Fig. 2).

GRL offers the following main concepts. In accordance with (Liu and Yu, 2004) *goals* are objectives that a stakeholder (modeled as *actors*) would like to achieve. To achieve goals, actors may employ *resources* (physical or informational entities) and perform *tasks*. Finally, GRL refers to a goal rationale as a *belief*.

Our vGREL profile extends GRL with two stereotypes: «ValueObject », applied to the GRL Resource concept, and «ValueActivity », applied to the GRL Task concept. These stereotypes are inspired by the e$^3$value concepts of the same name (Gordijn and Akkermans, 2003). A ValueObject is a Resource that is valuable for the achievement of a goal. A ValueActivity is a Task to produce ValueObjects.

Note that, in GRL, the concepts “Tasks” and “Resources” do not exists as independent metaclasses, but as enumerated attributes of a generic metaclass called “IntentionalElement”. To stereotype the right intentional elements (Resource, Task), we therefore introduce two OCL constraints (Object Management Group, 2014). See Fig. 2.

### 2.2. The vGREL process

The vGREL language introduced above captures the result from a requirements engineering exercise for conceptual modeling languages. In this section we introduce a process, depicted in Fig 3, to guide the language engineer to perform this requirements engineering exercise.

The vGREL process adopts a mixture of bottom up and top down requirements elicitation. In the top down sense, the language engineer elicits stakeholders, their top level goals, and the refinement of top-level goals into goals that are sufficiently detailed to match to a particular language. In Fig 3 this is done in Step 1, whereby the language engineer may receive support from the stakeholder perspective catalog. In the bottom up sense, the language engineer considers languages as a collection of valuable components. As we explain in more detail in Sect. 3.2, the language engineer thereby considers languages from the perspective of their inherent capabilities (i.e., what can be modeled with the language), and their extra-language capabilities (e.g., existence of tool support, industry uptake, and a concrete syntax). In Fig 3 this bottom-up elicitation is done in Step 2, whereby the language engineer follows a sequential procedure to examine one by one candidate languages for a possible solution by interacting with the language catalog.

The two threads meet in Step 3, where the valuable capabilities of candidate languages are related to goals to show how the former contribute to achieving the latter. Thereafter, an iterative step (Step 4 and 4’) gives room to the language engineer to consider alternative languages and complete the solution set. Finally, in Step 5 the solution set (goals, candidate languages, their capabilities and limitations, and contribution to goals) is stored in the stakeholder perspective catalog for future reuse.

Sect. 3 provides a more detailed walk-through of the vGREL process, and illustrates it with a concrete case study.

### 2.3. Tool support and goal satisfaction evaluation

We implement the value profile of GRL (the vGREL language) by customizing the jUCMNav tool [1], which is a tool for creating and analyzing GRL models. The tool provides stereotyping capabilities. This allows us to stereotype GRL tasks into vGREL value activities and GRL resources to vGREL value objects. More specifically, value activities are shown as tasks with the stereotype «ValueActivity ». Similarly, value objects are displayed as resources with the stereotype «ValueObject ».

Moreover, our vGREL approach leverages goal satisfaction evaluation strategies offered by jUCMNav to facilitate decision-making of language engineers during the vGREL process. An evaluation strategy specifies the initial satisfaction of the leaf nodes in a GRL model and follows a propagation procedure to evaluate the satisfaction of higher-level goals. Such an evaluation enables the language engineer to have a direct assessment of the fitness of the current solution set with respect to the achievement of the top-level goals. Also, jUCMNav highlights goals that are still to be addressed. We discuss this in more detail in Sect. 3.5.

## 3. Case study: Responsibility-based Access Rights Management

To explain the vGREL approach in further detail, we now apply it to a case study on Responsibility-based Access Rights Management, hereafter referred to as RARiMa.

[1] `http://www.ohloh.net/p/11712`

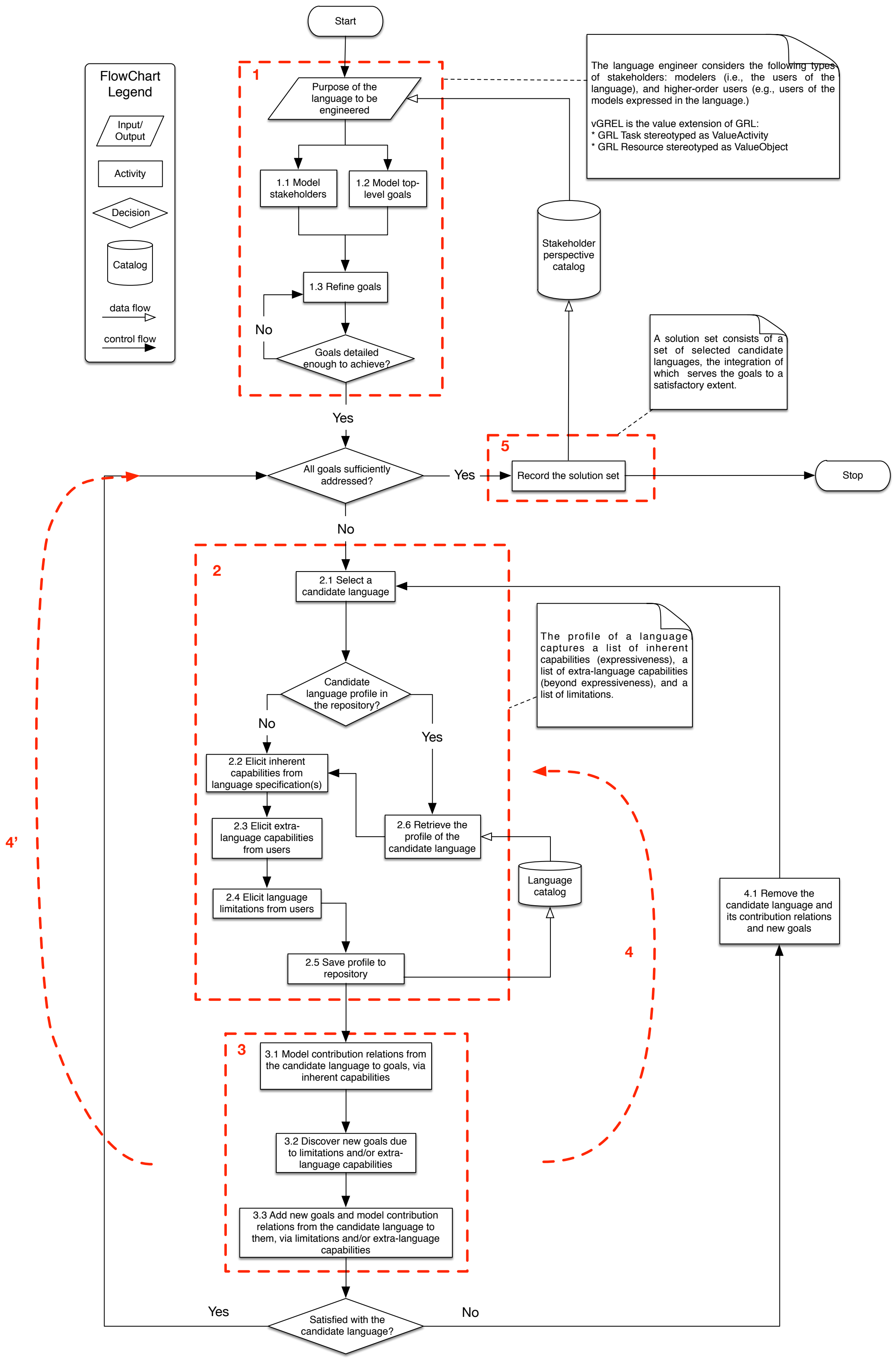
Start
FlowChart Legend
Input/ Output
Activity
Decision
Catalog
data flow
control flow
1
Purpose of the language to be engineered
1.1 Model stakeholders
1.2 Model top-level goals
1.3 Refine goals
No
Goals detailed enough to achieve?
Yes
The language engineer considers the following types of stakeholders: modelers (i.e., the users of the language), and higher-order users (e.g., users of the models expressed in the language.)
vGREL is the value extension of GRL:
* GRL Task stereotyped as ValueActivity
* GRL Resource stereotyped as ValueObject
Stakeholder perspective catalog
A solution set consists of a set of selected candidate languages, the integration of which serves the goals to a satisfactory extent.
5
All goals sufficiently addressed?
Yes
Record the solution set
Stop
No
2
2.1 Select a candidate language
The profile of a language captures a list of inherent capabilities (expressiveness), a list of extra-language capabilities (beyond expressiveness), and a list of limitations.
Candidate language profile in the repository?
No
Yes
2.2 Elicit inherent capabilities from language specification(s)
2.3 Elicit extra-language capabilities from users
2.4 Elicit language limitations from users
2.6 Retrieve the profile of the candidate language
Language catalog
2.5 Save profile to repository
4'
4
4.1 Remove the candidate language and its contribution relations and new goals
3
3.1 Model contribution relations from the candidate language to goals, via inherent capabilities
3.2 Discover new goals due to limitations and/or extra-language capabilities
3.3 Add new goals and model contribution relations from the candidate language to them, via limitations and/or extra-language capabilities
Yes
Satisfied with the candidate language?
No

Fig. 3. The FlowChart depicting the vGREL Requirements Engineering Process for Conceptual Modeling Language Development

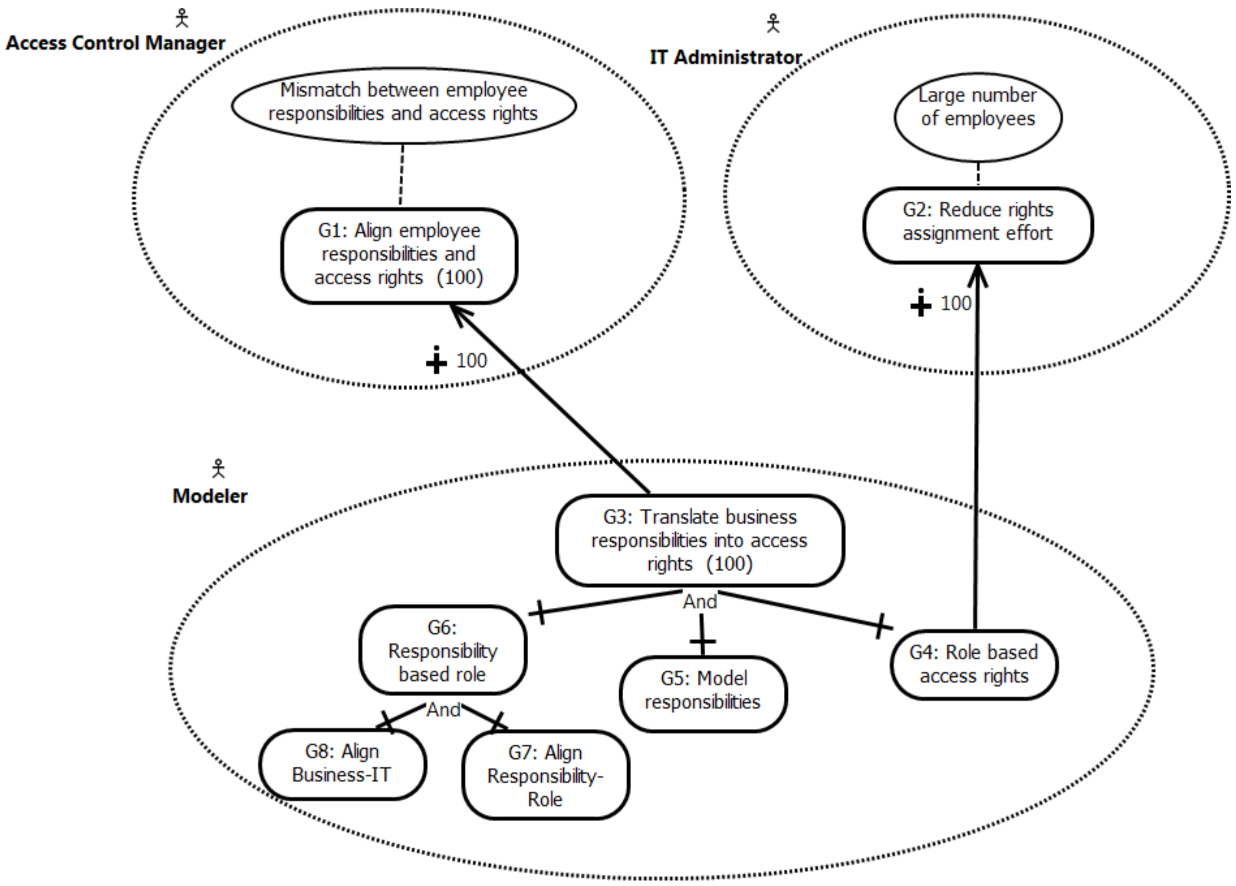
Access Control Manager
Mismatch between employee responsibilities and access rights
G1: Align employee responsibilities and access rights (100)
100
IT Administrator
Large number of employees
G2: Reduce rights assignment effort
100
Modeler
G3: Translate business responsibilities into access rights (100)
And
G6: Responsibility based role
G5: Model responsibilities
G4: Role based access rights
And
G8: Align Business-IT
G7: Align Responsibility-Role

Fig. 4. RARiMa: Stakeholders and Goals

## 3.1. *Case study setup*

The case study material is derived from two sources: (1) documentation, in terms of a PhD thesis that describes the language engineering exercise for RARiMa (Feltus, 2014); (2) Two semi structured interviews, of one hour each, with the language engineer of RARiMa. The first interview concerns elicitation. Taking the PhD thesis as a starting point, we asked about modeling motivations, and the reason for selecting languages. For example: using the interview, we elicited extra-language capabilities lacking from the thesis. The second interview concerns validation. We showed the resulting models to the language engineer to check their correctness. Furthermore, we explained the vGREL approach to elicit feedback on the potential usefulness of our approach for requirements engineering in the context of conceptual modeling.

## 3.2. *Step 1: Elicit stakeholders and goals*

Taking the purpose of the language as a starting point and by consulting the stakeholder perspective catalog for inspiration, the language engineer identifies the involved stakeholders (Step 1.1 in Fig. 3). Modelers are first-order stakeholders. They use the language directly to produce models. Moreover we deem it important to consider also *higher-order* stakeholders of the language. Instead of using the language directly, higher-order stakeholders use the models expressed in the language, and benefit from analysis results of the models. For example, (Stirna and Persson, 2012) points out problem owners as an important higher-order modeling stakeholder, in the sense that they see the usefulness of the modeling exercise and make an investment into it.

In addition, we model the top-level goals of the stakeholders (Step 1.2). By considering both first-order and higher-order stakeholders, we go beyond the perspective of modelers which is limited only to modeling itself. We emphasize that language engineering is more than just for the sake of modeling, but that we engineer languages as a means to an end. This is where GRL's goal orientation helps the language engineer, allowing him/her to ask *why?* questions about the different modeling stakeholders. In addition, GRL allows to identify conflicts between goals, helps to resolve these conflicts, and to consider the rationale (in terms of GRL beliefs) for goals.

Finally, after having identified stakeholders and their top-level goals, the language engineer refines the latter to a level of abstraction that is specific enough to find languages for (Step 1.3).

For the RARiMa case study we identify three main stakeholders in Step 1.1, modeled as actors: the modeler as a first order stakeholder, and the IT Administrator and Access Control Manager as second-order stakeholders (see Fig. 4). As identified by following Step 1.2, the Access Control Manager has one main goal: "G1: Align employee responsibilities and access rights", whereby responsibilities are those tasks that an employee actually carries out to do his or her job. This goal is motivated by the belief that there is a "Mismatch between employee responsibilities and access rights". As observed in (Feltus, 2014), business roles - the current concept that access rights are assigned to - are too coarse grained for access rights management. This results in employees having too many or too few access rights. Furthermore the IT Administrator wants to "G2: Reduce rights assignment effort" due to "Large number of employees".

Thereafter, in Step 1.3, we refine G1 and G2 into a set of sub-goals to be achieved. More specifically, to achieve the goal of the Access Control Manager (G1) the Modeler has the goal "G3: Translate business responsibilities into access rights". Note here that the "Make" relation between G3 and the goal of the access rights manager (G1) denotes that the achievement of G3 is sufficient for the achievement of G1. To achieve the goal of the IT Administrator (G2) the Modeler can exploit "G4: Role based access rights". This is because a role aggregates multiple employees, so that the number of assignments decreases. Meanwhile G4, together with "G5: Model responsibilities" and "G6: Responsibility based role definition" constitute a solution to achieve G3. Finally the achievement of G6 implies both "G7: Align Responsibility-Role" and "G8: Align Business-IT".

## *3.3. Step 2: Elicit language capabilities*

Starting from the specified goals the language engineer follows a sequential procedure to find candidate languages that possesses valuable capabilities to achieve the goals. Such candidate languages are discovered one at a time (Step 2.1), either from the language catalog (Step 2.5) or based on the experience of the language engineer.

If the candidate language is not yet in the catalog, the engineer analyzes its capabilities and limitations and models them as value objects (Step 2.2 – 2.4). We distinguish two types of capabilities: inherent capabilities and extra-language capabilities. Inherent capabilities refer to the expressiveness of a language itself, i.e. what can be modeled. They can typically be found in language specifications such as standard documents or reference books/papers. Extra-language capabilities refer to benefits beyond the expressiveness of a language such as existing tool support, industry uptake of a language, a user friendly concrete syntax, or otherwise. They can typically be derived from user experiences. These two types of capabilities are in line with the observations on factors for architectural language uptake reported in (Malavolta et al., 2013). In this survey, it is emphasized that about 50% of the surveyed language users deems extra-language capabilities important for selecting a language.

In addition, the language engineer should also consider the limitations of a language. The list of elicited capabilities and limitations together constitute a language profile. This language profile is stored in the language catalog to facilitate sharing of experience with a language (Step 2.6).

For the RARiMa case study, we initially select in Step 2.1 the Responsibility MetaModel (ReMMo), as defined in (Feltus, 2014), because the ideas behind ReMMo are nicely aligned with "G5: Model responsibilities" and "G7: Align Responsibility-Role". More specifically, we elicit in Step 2.2 the following two inherent capabilities of ReMMo: (1) expressing employee responsibilities, and (2) tracing roles used for access rights management at the application layer back to employee responsibilities. We model them as two corresponding ValueObjects and specify that ReMMo positively and sufficiently provides them. For

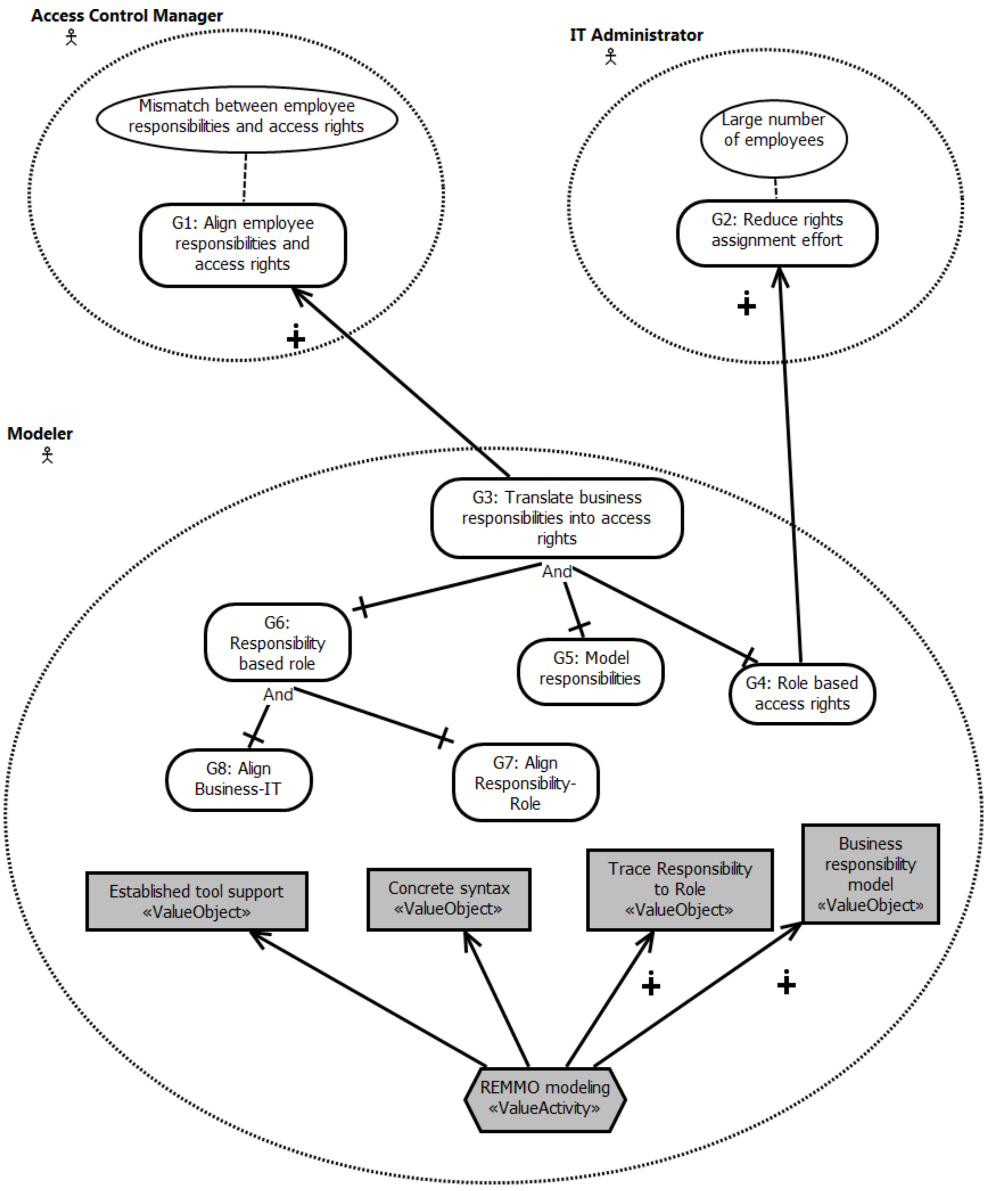
Access Control Manager
Mismatch between employee responsibilities and access rights
G1: Align employee responsibilities and access rights
IT Administrator
Large number of employees
G2: Reduce rights assignment effort
Modeler
G3: Translate business responsibilities into access rights
And
G6: Responsibility based role
G5: Model responsibilities
G4: Role based access rights
And
G8: Align Business-IT
G7: Align Responsibility-Role
Established tool support «ValueObject»
Concrete syntax «ValueObject»
Trace Responsibility to Role «ValueObject»
Business responsibility model «ValueObject»
REMMO modeling «ValueActivity»

Fig. 5. RARiMa: Elicited ReMMo Capabilities and Limitations

Step 2.3 and 2.4, from an interview with the designer of ReMMo two limitations emerged: (1) lack of concrete syntax, and (2) lack of tool support. We model the corresponding compensations as ValueObjects and specify that ReMMo fails to provide these ValueObjects by means of negative contribution links. In addition, no extra-language capabilities are identified. The above inherent capabilities and limitations constitute a language profile of ReMMo to be stored in the language catalog in Step 2.5. Note that this profile only reflects a partial characterization of ReMMo that is relevant for the use case at hand. In future use cases in which ReMMo plays a role, this profile can be retrieved and extended (Step 2.6). Fig. 5 shows the vGREL model at the end of applying Step 2 to the RARiMa case, whereby the additions are highlighted in grey.

### 3.4. Step 3: Elicit contribution to existing goals and discover new goals

In this step the language engineer examines the candidate language selected in Step 2. S/he does so by firstly specifying the contribution links from its inherent capabilities, modeled as value objects, to existing

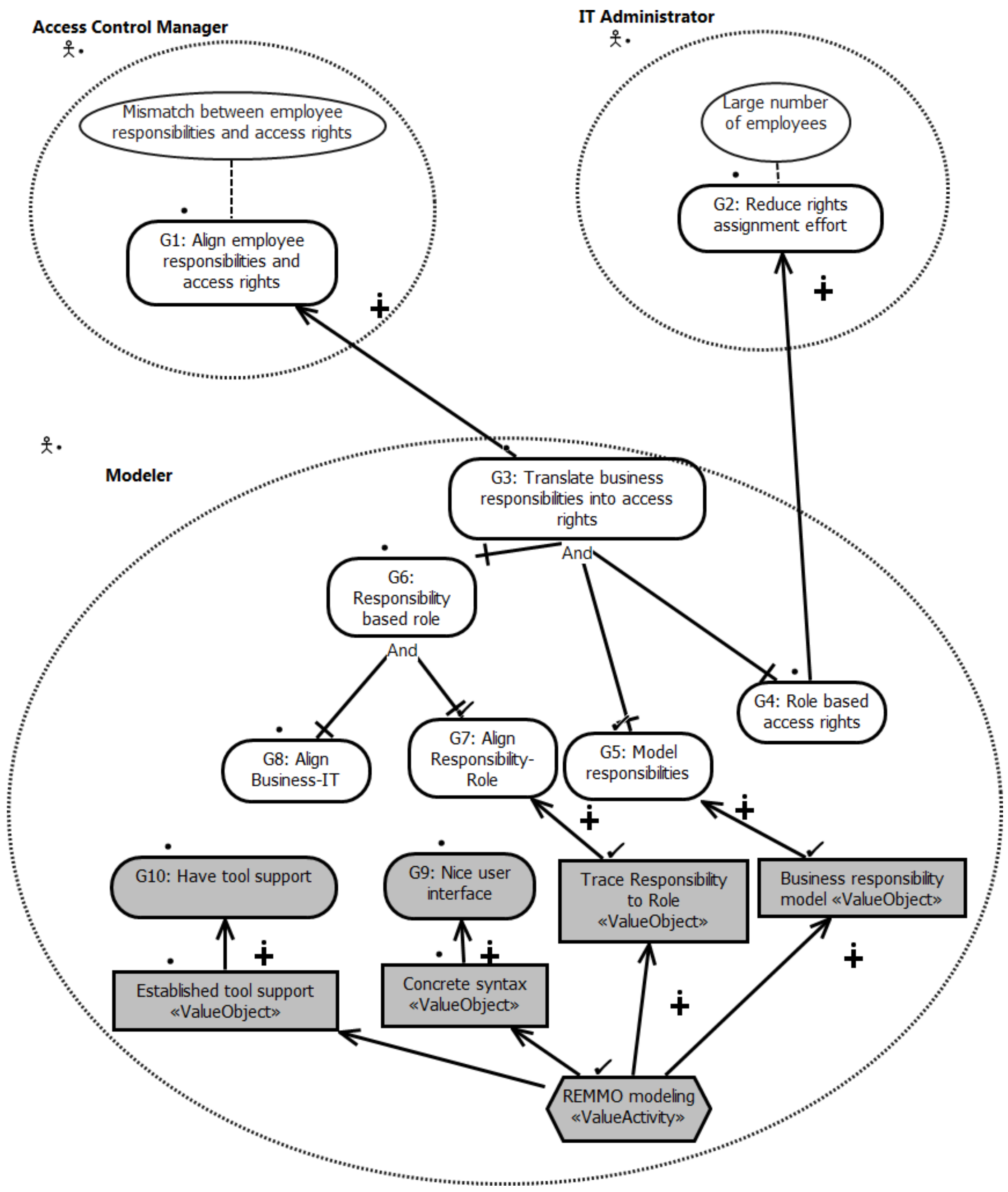
Access Control Manager
Mismatch between employee responsibilities and access rights
G1: Align employee responsibilities and access rights
IT Administrator
Large number of employees
G2: Reduce rights assignment effort
Modeler
G3: Translate business responsibilities into access rights
And
G6: Responsibility based role
And
G4: Role based access rights
G8: Align Business-IT
G7: Align Responsibility-Role
G5: Model responsibilities
G10: Have tool support
G9: Nice user interface
Trace Responsibility to Role «ValueObject»
Business responsibility model «ValueObject»
Established tool support «ValueObject»
Concrete syntax «ValueObject»
REMMO modeling «ValueActivity»

Fig. 6. RARiMa: Contributions and New Goals Introduced by ReMMo

goals (Step 3.1 in Fig. 3). Furthermore, to account for the extra-language capabilities and the limitations of the candidate language, the language engineer specifies new goals to achieve (Step 3.2). Finally, the language engineer specifies the contribution links between the limitations and extra-language capabilities to the new goals (Step 3.3). These contribution links cover both those from the current candidate language under examination, and those from the other languages included in the solution in earlier iterations.

For the RARiMa case, in Step 3.1 we capture the positive and sufficient contribution relations from the inherent capability "Business responsibility model" to the goal "G5: Model responsibilities" and from "Trace responsibility to role" to "G7: Align responsibility-role" in terms of "Make" contribution links. In Step 3.2, two new goals emerge as a result of ReMMO limitations: "G9: Nice user interface" as a response to lacking concrete syntax, and "G10: Have tool support" as a response to lacking tool support. Note that

G9 and G10 are not achieved in this step (for reference, see the vGREL legend of Fig. 2) because ReMMo fails to provide the two ValueObjects that can fulfill them (Step 3.3). After Step 3, the vGREL model of Fig. 5 evolves into Fig. 6.

## *3.5. Step 4: Iteration*

In this step the language engineer evaluates the satisfaction of the candidate languages by weighting the benefits against the limitations. This evaluation is supported by the goal satisfaction analyses offered by GRL which, from the positive/negative contributions, can identify the extent to which (upper-level) goals are satisfied.

As depicted by the guard "Satisfied with the candidate language?" in the vGREL process (Fig. 3), the language engineer has the option of following branch 4 to cancel the candidate language (Step 4.1), and selecting a new one by going through Step 2 and Step 3 again. Alternatively s/he can proceed to address other goals to be achieved, if any, by following branch 4'.

For the RARiMa case study, we are satisfied with ReMMo since it achieves G5 and G7 fully (as depicted by the ticks next to the goals in Fig. 6). As a consequence, after finishing Step 3, we follow the branch 4' to the guard "All goals sufficiently addressed?". The evaluation of goal satisfaction on the model resulting from step 3 (Fig 6) shows four leaf goals left to achieve: "G8: Align Business-IT", "G9: Nice user interface", "G10: Have tool support", and "G4: Role based access rights". In the following iterations, we will discover one by one other candidate languages for the unfulfilled goals.

Enterprise architecture models are a prominent instrument to capture business-IT alignment, as these models link business and technology layers (Lankhorst and et al., 2012, p.75). ArchiMate is a de facto standard for enterprise architecture modeling. Hence, we consider ArchiMate a candidate language for achieving G8. After repeating steps 2 and 3 for ArchiMate, we also find additional extra-language capabilities of ArchiMate that provide the following ValueObjects: "Concrete syntax", "Established tool support", and "Industry uptake". The first two ValueObjects contribute to achieving G9 and G10 respectively, while the last ValueObject contributes to achieving a new goal "G11: Foster uptake of language". This again emphasizes why it is important to look at a language in terms of both inherent capabilities *and* extra-language capabilities, as industry uptake and tool support can actually be important reasons for the uptake of the ArchiMate language.

After this iteration, only one leaf goal, namely G4, is left unfulfilled. We select in the following iteration the Role Based Access Control language (RBAC) as defined in (Ferraiolo et al., 2001) because it appears promising for the achievement of G4 (Role based access rights). Indeed, RBAC provides an inherent capability "Role based IT access right model" that fulfills G4. RBAC also manifests an extra-language capability "Industry uptake" that contributes to G11. RBAC fosters industry uptake due to the widespread use of RBAC in industry systems and tools: FreeBSD, Solaris, SAP R/3, Oracle DBMS, to name a few.

At the end of the iteration whereby RBAC is selected, all the goals are sufficiently achieved as can be seen in Fig. 7, hence the iteration stops here.

## *3.6. Step 5: Record solution set*

After all the goals are addressed to a satisfactory extent, the resulting vGREL is saved in a repository called "stakeholder perspective catalog". This catalog can be used for inspiration in similar future modeling situations.

For the RARiMa case, as can be seen in Fig. 7, the set of languages {ReMMO, ArchiMate, RBAC} constitutes a solution to achieving the two original goals of the higher order users: "G1: Align employee responsibilities and access rights" and "G2: Reduce rights assignment effort". This solution (selected languages, their capabilities and limitations, and the contribution to goals) is stored in the stakeholder catalog for reuse.

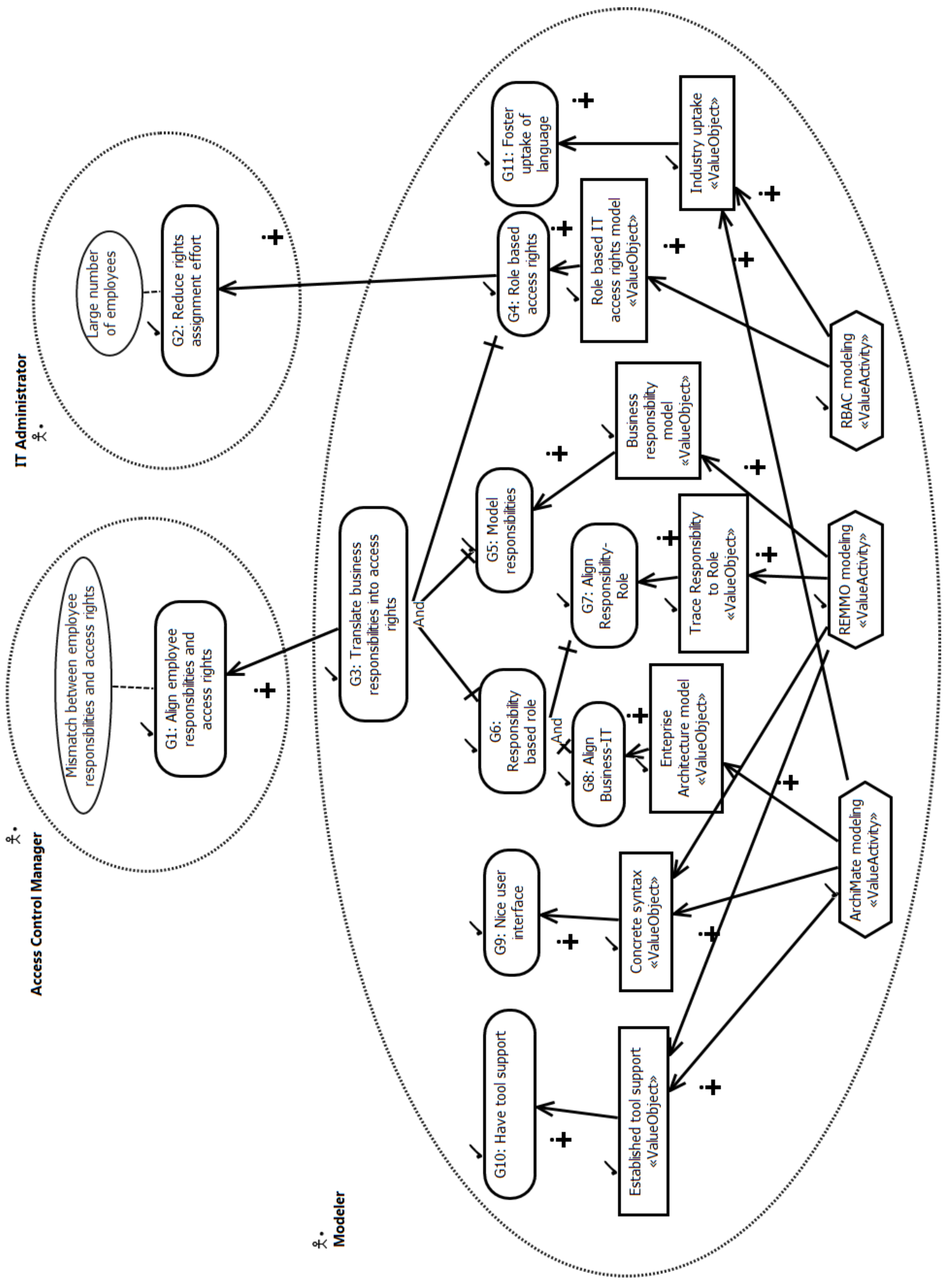
Access Control Manager
IT Administrator
Modeler
Mismatch between employee responsibilities and access rights
G1: Align employee responsibilities and access rights
Large number of employees
G2: Reduce rights assignment effort
G3: Translate business responsibilities into access rights
And
G4: Role based access rights
G5: Model responsibilities
G6: Responsibility based role
And
G7: Align Responsibility-Role
G8: Align Business-IT
G9: Nice user interface
G10: Have tool support
G11: Foster uptake of language
Role based IT access rights model «ValueObject»
Industry uptake «ValueObject»
Business responsibility model «ValueObject»
Trace Responsibility to Role «ValueObject»
Enteprise Architecture model «ValueObject»
Concrete syntax «ValueObject»
Established tool support «ValueObject»
RBAC modeling «ValueActivity»
REMMO modeling «ValueActivity»
ArchiMate modeling «ValueActivity»

Fig. 7. RARiMa: Complete Stakeholder Perspective Catalog

## 4. Discussion and related work

### *4.1. Reflection from the language engineer*

As stated in Sect. 3.1, we validated our approach with the RARiMa language engineer in terms of (1) the extent to which the vGREL models reflect the reasoning, (2) the extent to which vGREL could be useful for engineering languages.

Regarding the extent to which the vGREL models reflect the reasoning of the language engineer, it was confirmed that the vGREL models reflect the modeling considerations of the language engineer. In particular, the language engineer confirmed that the motivations of higher-order users, such as the access control manager, were important drivers for developing the responsibility based access control management language.

Furthermore it was indeed deemed important to capture both inherent and extra-language capabilities. For example, the language engineer reflected that, indeed, extra-language capabilities such as "industry uptake" and "concrete syntax" were important factors in selecting ArchiMate and RBAC.

Regarding the extent to which vGREL could be useful for engineering languages, the language engineer recognizes the potential of vGREL for showing alternative solutions to a language engineering problem. To illustrate this, the language engineer related to us a particular access control use case. In this case, the engineer was required to use a proprietary architectural language, while the preference was to use ArchiMate. By applying vGREL, and in particular its capability to evaluate goal satisfaction (see Sect. 2.3), the language engineer could have demonstrated several alternatives for modeling the use case, and the benefits for selecting ArchiMate (the preferred option).

Moreover the language engineer confirmed potential for the reuse of good language engineering practices as stored in the two catalogs.

### *4.2. Related work*

In software engineering, modeling languages can be designed by employing feature diagrams (Mernik et al., 2005). In feature diagrams, one designs a language by specifying an abstract feature (for example "browsing") into more detailed ones (for example "get" or "post" features for "browsing") until one derives specific language concepts. Logical operators ((X)OR, AND) are used for feature specification. However in feature diagrams everything is a "feature", tailored to software concerns such as (multiple) inheritance.

(Strembeck and Zdun, 2009; Zdun, 2010) provide processes for developing modeling languages. While these processes seem useful for actually creating a language, they ignore the step emphasized in this paper: the requirements engineering phase for the design of conceptual modeling languages. Furthermore (Spinellis, 2001) provides design patterns for DSL (Domain Specific Language) design. Examples include piggybacking DSLs, whereby the capabilities from one DSL form the basis for hosting another DSL, or the restriction and extension of existing DSLs. However these DSL development patterns remain generic: what the different actors and their concerns are, and how languages are valuable, is not addressed.

Generally speaking, as also emphasized in (Strembeck and Zdun, 2009), in software engineering the role of "soft" factors for designing modeling languages, such as whom you are modeling for, and languages as collections of valuable components, is under-researched.

In Situational Method Engineering (SME), (Chiniforooshan Esfahani et al., 2010; Chiniforooshan Esfahani and Yu, 2010) propose an approach for Goal based SME, based on the goal modeling technique i*. However, Chiniforooshan Esfahani et al. (2010); Chiniforooshan Esfahani and Yu (2010) lack (1) an explicit conception of languages as sets of valuable components. (2) an explicit consideration of different actors, that have different motivations/interests in valuable capabilities of modeling languages. Furthermore the SME approaches by (Ågerfalk and Fitzgerald, 2006; Rossi et al., 2004) record method rationale in terms of goals so as to, amongst others, better understand situation specific method adaptations, and to foster communication between method engineers and method users. However, this recording of method rationale is descriptive while we primarily aim at supporting decision making during language design. Finally, as implied by name, aforementioned SME approaches are intended for *methods* while we aim specifically for *language* engineering. As a consequence, language specific dimensions such as the specification of syntax and semantics are not considered. This consideration is especially relevant for later iterations of our work, whereby we intend to complement our requirements engineering process with actual language design and implementation steps, thus leading to a complete language design process.

Finally (Bubenko Jr. et al., 2010) consider the modeling purpose as a central tenet for the development and selection of enterprise modeling techniques. However (Bubenko Jr. et al., 2010) focus on (1) a generic set of enterprise modeling purposes: “develop the business”, “develop information system”, and “develop vision and strategies”. Thus they forgo situation specific goals and values of modeling languages; (2) focuses on a specific technique, called Enterprise Knowledge Development (EKD), hence they do not extrapolate to the selection of language fragments from multiple different languages; (3) does not consider the value of modeling languages.

## 5. Conclusion and future work

In this paper, we introduced the vGREL approach for value- and goal-oriented requirements engineering of conceptual modeling languages. We showed how our approach can help a language engineer in perceiving of language design as a requirements engineering exercise in its own right, by considering languages as valuable components that together satisfy some purpose. To this end, we introduced (1) a process for the requirements engineering of conceptual modeling languages, (2) a value profile for GRL to capture the result of the language engineering process, (3) software tool support for language selection.

Finally, we applied vGREL to a case study on responsibility based access rights management, and gathered feedback from our case study partner. The feedback is encouraging, particularly in the sense that the language engineer sees potential for using vGREL as a language selection mechanism.

For further research, we first and foremost foresee further practical validation of vGREL. Currently, we are involved in an effort from the standardization body The Open Group[2] to integrate BPMN (a process modeling language) and ArchiMate. With vGREL we can elaborate and test different scenarios for integrating these two languages. Also, this language integration effort provides an opportunity for extending the prioritization mechanism of jUCMNav (the software tool for creating and reasoning with GRL models) with different decision making strategies. Particularly we consider using the heuristics inherent to non compensatory decision making strategies to alleviate the effort required for using vGREL. An example of such a non compensatory decision making strategy is conjunctive decision making (Elrod et al., 2004), whereby an alternative is discarded if it fails to meet a minimal threshold score for one particular attribute. The reduced decision making effort is of particular importance given that we want to keep vGREL lightweight, so that the costs of doing requirements engineering for conceptual modeling languages do not outweigh the benefits of doing so.

In addition, an important next step will be to combine requirement engineering for conceptual modeling languages with language design and implementation techniques. As such, we will arrive at a *complete* language engineering process. One typical consideration for language design and implementation, especially when following a language reuse strategy, is to integrate selected candidate languages. Such an integration may impose extra constraints and requirements on language development. However, technical considerations such as the identification of integration points and the selection of integration methods are more pertinent to the more detailed specification phase of the language engineering, i.e., the actual language specification (see also Fig. 1). In this specification phase, more technical considerations are tackled, such as defining the syntax and the semantics of a language. Therefore these considerations are out of scope for this paper, which focuses on requirements engineering considerations and their relevance in language engineering. Nevertheless, we do recognize that working on language specification may influence require-

[2]`http://www.opengroup.org`

ments engineering as well (as indicated by the feedback loop from “Language Specification” to “Analysis of requirements” in Fig. 1). Therefore, as part of future work, we will further explore the interaction between language specification and requirements engineering.

Finally we plan to extend on the notion of language reuse, i.e., to (1) define clearly criteria for elementary language fragments that can be stored in the two vGREL catalogs; (2) to define strategies for intelligent mining of the catalogs. The latter becomes especially relevant as the number of language fragments, and especially language use situations, increases.

*Acknowledgments. The authors would like to thank Christoph Feltus for his valuable case study input and the provided feedback on case study material. Furthermore, the authors thank Sepideh Ghanavati for her guidance on using the jUCMNav tool. This work has been sponsored by the Fonds National de la Recherche Luxembourg (*`www.fnr.lu`*), via the PEARL programme.*

## References

Ågerfalk, P. and Fitzgerald, B. (2006). Exploring the concept of method rationale: A conceptual tool. In *Advanced Topics in Database Research*, volume 5, pages 63–78. IGI Global.

Bjekovic, M., Proper, H. A., and Sottet, J.-S. (2014). Embracing pragmatics. In Yu, E., Dobbie, G., Jarke, M., and Purao, S., editors, *Conceptual Modeling*, volume 8824 of *Lecture Notes in Computer Science*, pages 431–444. Springer.

Bubenko Jr., J., Persson, A., and Stirna, J. (2010). An intentional perspective on enterprise modeling. In Nurcan, S., Salinesi, C., Souveyet, C., and Ralyté, J., editors, *Intentional Perspectives on Information Systems Engineering*, pages 215–237. Springer.

Chiniforooshan Esfahani, H. and Yu, E. (2010). A repository of agile method fragments. *New Modeling Concepts for Today's Software Processes*, pages 163–174.

Chiniforooshan Esfahani, H., Yu, E., and Cabot, J. (2010). Situational evaluation of method fragments: an evidence-based goal-oriented approach. In *Advanced Information Systems Engineering*, pages 424–438. Springer.

de Kinderen, S., Gaaloul, K., and Proper, H. A. (2014a). Bridging value modelling to archimate via transaction modelling. *Software & Systems Modeling*, 13(3):1043–1057.

de Kinderen, S., Ma, Q., and Proper, H. A. (2014b). Model bundling: Towards a value-based componential approach for language engineering. In *Proceedings of the 8th international workshop on Value Modeling and Business Ontology (VMBO 2014)*.

de Kinderen, S. and Proper, H. A. (2013). E3rome: A value-based approach for method bundling. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1469–1471, New York, NY, USA. ACM.

Elrod, T., Johnson, R. D., and White, J. (2004). A new integrated model of noncompensatory and compensatory decision strategies. *Organizational Behavior and Human Decision Processes*, 95(1):1–19.

Feltus, C. (2014). *Aligning Access Rights to Governance Needs with the Responsibility MetaModel (ReMMo) in the Frame of Enterprise Architecture*. PhD thesis, University of Namur.

Feltus, C., Dubois, E., Proper, E., Band, I., and Petit, M. (2012). Enhancing the ArchiMate standard with a responsibility modeling language for access rights management. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, SIN '12, pages 12–19, New York, NY, USA. ACM.

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., and Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274.

Frank, U. (2013). Domain-specific modeling languages: Requirements analysis and design guidelines. In *Domain Engineering*, pages 133–157. Springer.

Gordijn, J. and Akkermans, H. (2003). Value based requirements engineering: Exploring innovative e-commerce ideas. *Requirements Engineering Journal*, 8(2):114–134.

ITU-T (2008). User requirements notation (URN)–language definition. Recommendation Z.151 (11/08). URL: http://www.itu.int/rec/T-REC-Z.151/en. Last accessed on 31-07-2014.

Lankhorst, M. and et al. (2012). *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, 3rd edition.

Liu, L. and Yu, E. (2004). Designing information systems in social context: a goal and scenario modelling approach. *Information systems*, 29(2):187–203.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., and Tang, A. (2013). What industry needs from architectural languages: A survey. *Software Engineering, IEEE Transactions on*, 39(6):869–891.

Meertens, L. O., Iacob, M. E., Nieuwenhuis, L. J. M., van Sinderen, M. J., Jonkers, H., and Quartel, D. (2012). Mapping the Business Model Canvas to ArchiMate. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1694–1701, New York, NY, USA. ACM.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Moody, D. L. (2005). Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, 55(3):243 – 276.

Object Management Group (2014). Object Constraint Language v2.4.

Pijpers, V., de Leenheer, P., Gordijn, J., and Akkermans, H. (2012). Using conceptual models to explore business-ICT alignment in networked value constellations. *Requirements Engineering*, 17(3):203–226.

Rossi, M., Ramesh, B., Lyytinen, K., and Tolvanen, J. (2004). Managing evolutionary method engineering by method rationale. *Journal of the Association for Information Systems*, 5(9):356–391.

Rothenberg, J. (1989). The nature of modeling. In *Artificial intelligence, simulation & modeling*, pages 75–92. John Wiley & Sons, Inc.

Spinellis, D. (2001). Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 56(1):91–99.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional.

Stirna, J. and Persson, A. (2012). Purpose driven competency planning for enterprise modeling projects. In Ralyté, J., J., Franch, X., Brinkkemper, S., and Wrycza, S., editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 662–677. Springer.

Strembeck, M. and Zdun, U. (2009). An approach for the systematic development of domain-specific languages. *Software: Practice and Experience*, 39(15):1253–1292.

Thalheim, B. (2011). The theory of conceptual models, the theory of conceptual modelling and foundations of conceptual modelling. In Embley, D. W. and Thalheim, B., editors, *Handbook of Conceptual Modeling*, pages 543–577. Springer.

van Buuren, R., Gordijn, J., and Janssen, W. (2005). Business case modelling for e-services. In *18 th Bled eConference eIntegration in Action*. AIS.

Zdun, U. (2010). A DSL toolkit for deferring architectural decisions in DSL-based software design. *Information and Software Technology*, 52(7):733–748.