Requirements Eng
DOI 10.1007/s00766-015-0235-1

ORIGINAL ARTICLE

Modelling mobile app requirements for semantic traceability

Robert Andrei Buchmann[1,2] · **Dimitris Karagiannis**[2]

Received: 19 May 2014 / Accepted: 14 July 2015

Abstract The paper presents a modelling method aimed to support the definition and elicitation of requirements for mobile apps through an approach that enables semantic traceability for the requirements representation. Business process-centricity is employed in order to capture requirements in a knowledge structure that retains procedural knowledge from stakeholders and can be traversed by semantic queries in order to trace domain-specific contextual information for the modelled requirements. Consequently, instead of having requirements represented as natural language items that are documented by diagrammatic models, the communication channels are switched: semantically interlinked conceptual models become the requirements representation, while free text can be used for requirements annotations/metadata. Thus, the method establishes a knowledge externalization channel between business stakeholders and app developers, also tackling the Twin Peaks bridging challenge (between requirements and early designs). The method is presented using its modelling procedure as a guiding thread, with each step illustrated by case-based samples of the modelling language and auxiliary functionality. The design work is encompassed by an existing metamodelling framework and introduces a taxonomy for modelling relations, since the metamodel is the key enabler for the goal of semantic traceability. The research was driven by the ComVantage EU research project, concerned with mobile app support for collaborative business process execution. Therefore, the project provides context for the illustrating examples; however, generalization possibilities beyond the project scope will also be discussed, with respect to both motivation and outcome.

Keywords Mobile app requirements · Mobile interaction · Linked Data · Requirements modelling · Semantic traceability

✉ Robert Andrei Buchmann
rbuchmann@dke.univie.ac.at;
robert.buchmann@econ.ubbcluj.ro

Dimitris Karagiannis
dk@dke.univie.ac.at

[1] Faculty of Economic Sciences and Business Administration, University Babes-Bolyai of Cluj Napoca, Str. Teodor Mihali 58-60, 400591 Cluj Napoca, Romania

[2] Faculty of Computer Science, Knowledge Engineering Research Group, University of Vienna, Waehringerstr. 29, 1090 Vienna, Austria

1 Introduction

The goal of this work is to establish a knowledge-oriented, process-centric method for modelling mobile app requirements in tight and traceable relation to (a) the business environment where the apps will be used, as well as to (b) the early design decisions that must be negotiated with stakeholders across the *Twin Peaks gap* [41] between requirements and designs. The work was motivated on one hand by drawbacks identified in common practice and tool support for requirements representation, and on the other hand by opportunities and challenges identified during the development of the ComVantage EU research project [17]. We highlight in the remainder of this introduction the multiple facets of the established goal:

Knowledge-orientation. The commonly used tools for requirements engineering can be seen as bug trackers or task management tools that are repurposed or generalized as "requirements trackers" (e.g. FusionForge [21]). Their support goes mainly to requirements management, and less

Published online: 28 July 2015

to requirements representation, although this is the content communicated during elicitation [29]. Requirements are often captured in a centralized repository of weakly structured natural languages statements, typically derived from itemized survey or interview answers. General purpose diagrams (UML [45]) may support such descriptions (see IBM's tooling [26]), but they are still employed as weakly integrated artefacts, as auxiliary graphical documentation, rather than structured and integrated representations of the acquired knowledge, capable of supporting a comprehensive analysis of the requirements' context. Thus, we highlight this aspect with the subgoal of *bringing requirements gathering closer to structured knowledge acquisition*.

Process-centricity. For business stakeholders, requirements are "wishes" describing (mostly in domain-specific terms) what they have to do and what capabilities they need in order to do it. The developers discuss in terms of user interface controls, queries, design features, etc. The border between the two viewpoints is a fuzzy ground that needs to be structured during requirements elicitation. A key commonality identified on this border is the notion of *control flow*, which is present both on the business side (emerging from the practice of business process management) and on the IT side (inherent to algorithmic thinking). This can be leveraged to set up a communication strategy whose content is built around business process modelling.

Conceptual modelling languages can educate their users to communicate in a structured way, with a limited set of concepts, based on a common understanding of their semantics. We align to the position of J. Mylopoulos who emphasizes the broad value of conceptual models ("for purposes of understanding and communication" [39]), beyond the scope recently advocated by the software engineering community (works like [2] go as far as stating that *conceptual modelling is programming*, subordinating the entire field to model-driven code generation, in a context closely related to our work—*interaction modelling*). Therefore, we have to stress that, for example, business process modelling has proven successful as a knowledge management facilitator, regardless of any goals related to workflow automation. Thus, we highlight a second characteristic of the proposed approach: *reliance on existing stakeholder competence regarding business process modelling*.

Technological and domain specificity. These aspects originate in the context where the work was developed: the ComVantage EU research project [17], which proposes a Linked Data-driven mobile information system architecture to support collaborative business networks for specific application areas (customized production and remote maintenance). A subtask of the project deals with designing and implementing a method for modelling collaborative supply chains, integrated with their motivators (i.e. products, services, KPIs) and their IT requirements. The work at hand presents a fragment of this method, the one concerned with requirements representation. This fragment is reusable across business domains since it can be detached from the project's *domain specificity* (manifested in model types and constructs that can be linked to the ones discussed in this paper, but not mandatorily). On the other hand, the *technological specificity* (apps and Linked Data) has a tighter embedding in the modelling semantics (manifested in object properties and notation) and thus will limit reuse. Therefore, we can highlight a third characteristic of the proposed approach: *reliance on metamodelling as means of tailoring a modelling approach to some* (*evolving*) *degree of specificity*.

Semantic traceability. Our goal extends towards ensuring that the resulting requirements can be reused beyond the elicitation phase, in scenarios such as: human-based requirements validation, derivation of training materials from requirements, analysis of the requirements representation and, if certain prerequisites are met, model-aware code selection or process-aware requirements management systems. We designate the term *semantic traceability* for the ability to retrieve semantically related information both from upstream (business motivators and context) and downstream (early design decisions) of the development process, as well as from multiple levels of detail.

Towards this multi-faceted goal, we employ: (a) *metamodelling* to establish a procedure and a language for gradually building semantic links between conceptual models representing different facets or granularity levels from the requirements' enterprise context; (b) *Linked Data principles* to expose such semantic links for navigation, retrieval or rule-based processing; (c) a *taxonomy of modelling relations* that can become Linked Data, to the extent dictated by the level of traceability to be supported.

The paper is structured as follows: Sect. 2 states the problem and describes its background, including the research project context. Section 3 provides a high-level overview on the modelling method in the context of a metamodelling framework. Section 4 illustrates the modelling procedure with model examples for representing mobile app requirements. Section 5 glues the examples at metamodel level and introduces a taxonomy of modelling

relations that must be considered when aiming for traceability across models. Section 6 highlights some functionality built on top of the models. Section 7 discusses observations based on an empirical evaluation of a proof-of-concept implementation. Section 8 indicates related works briefly, followed by a more detailed analysis in Sect. 9, to better highlight the contribution. The paper ends with a conclusive SWOT analysis and states the final takeaway message.

2 Problem statement and background

2.1 Motivation and context

Both business process management and requirements engineering are fields that rely heavily on conceptual modelling; therefore, they converge here as application domains for our metamodelling concerns and framework. They further subsume the application domain of the ComVantage research project, where business-IT alignment must be achieved, considering business processes with domain-specific triggers (requested product customizations, sensor-triggered maintenance services) and some technological specificity (mobile apps with Linked Data).

From the projects' requirements elicitation workshops, several layers on which app requirements should be structured have been identified: (a) *the infrastructure layer*, where technological choices for front-end devices and back-end endpoints must be made; (b) *the data layer*, where data ownership and data requirements are defined; (c) *the domain layer*, described through domain concepts, tasks and required resource types; (d) *the user interaction layer*, where the required mobile IT support must be described for each role, down to a mock-up level of detail that can support the early design phase.

During elicitation workshops, we have noticed that, as the user stories are defined, stakeholders naturally tend to drill down requirements in a procedural fashion, in close relation to tasks from business processes they are familiar with, for example: *as a user* (*in role*) *R, in order to perform* (*the task*) *T, I want to be able to select an* (*item*) *X, then edit its* (*property*) *P*. This also assumes that user stories are allowed to be compounded on a manageable level, contrary to the practitioners' recommendation for their atomic decomposition [16] (hence losing the sense of flow between *selecting X* and *editing its property P*). If stakeholders are already familiar with their legacy system or their mobile devices, sometimes they add analogies in order to describe options that should be available at each business step.

Our challenge was to observe the above-mentioned communication style and to support it with conceptual modelling means, to help stakeholders externalize and structure their "wishes", while enabling app developers to refine and negotiate them iteratively, towards early mock-up proposals.

For business stakeholders, "domain specificity" manifests in the tasks to be performed, their motivators and resources types, the data entities to be handled. On the other hand, app developers think in terms of their own (mobile app) domain. Standards like UML abstract away from both the business view and the app design view, to generic concepts that software development processes and code generation frameworks are tailored to handle. However, it is often the case that none of the two parties are able or willing to discuss on that level of abstraction. While for business stakeholders this is quite often the case, the statement tends to become valid for mobile app developers as well:

With the recent advent of crowdsourcing-inspired deployment models ("app stores"), apps are being developed by members of a community, not necessarily by software companies. With a certain amount of training (not negligible, but with a tendency of being minimized by frameworks and reusable patterns) and a focus on front-end user experience (rather than architecture and back-end), app developers often short-circuit the maturation process of a software engineer and the life cycle for which UML was designed. This builds a case for a mobile app requirements engineering methodology that reflects the short circuit and still supports some level of agility in development. We add to this motivation the remark that UML is a modelling language, whereas this work proposes a modelling method focused on the semantic linking across different model types, to enable cross-model analysis and evaluation.

2.2 Research challenges and problem formulation

To summarize, the challenges tackled by this work are:

C1. How to support (with respect to requirements representation) externalization and communication of the above-mentioned classes of requirements over the business-IT gap and from the requirements analysis to the early design phase?

C2. How to enable semantic traceability and analysis of requirements, considering their semantics and directly or indirectly related entities (e.g. from the business context, from early designs)?

C3. Assuming that a modelling method is adopted as knowledge externalization approach, what types of modelling relations can be exposed as navigable semantic links?

We also formulate the problem in more grounded terms, inspired by pragmatic needs of the research project cases:

Assumptions. A company acting in a business collaboration network decides to deploy apps with a "bring-your-own-device" (BYOD) strategy [34] to support its collaborative processes. The stakeholders are familiar with their legacy systems; they know their business processes and working environments, and are able to describe them in business process models. The business partners have enforced access control policies on their (data) resources, but decide to relax them to the extent required for collaboration.

Open issues to be alleviated. The company must communicate requirements for supporting their business processes with front-end mobile IT and back-end Linked Data. The app developers must communicate back proposed refinements and mock-ups before initiating implementation. Requirements must be traceable (on a semantic basis) to their rationale (processes, process motivators), required resources and early design proposals. Several types of gaps must be identifiable (gaps in the requirements; gaps between required and available resources). When business processes change, impact on the mobile IT implementation should be minimized. Employees must be able to assimilate new/redesigned processes and learn how the apps support them. To enable collaboration, business partners need to know what data access means to relax or create. Finally, the company might have to justify that Linked Data and mobile technology are beneficial for their business processes.

3 An overview of the modelling method

The solution is provided in the form of a knowledge structure called *modelling method*, defined in [31] in terms of the following building blocks:

1. A *modelling procedure*: human-oriented procedural knowledge guiding the modeller to his/her goals;
2. A *modelling language*: structural and declarative knowledge defining the semantically connected modelling constructs, their notation and editable properties ("modelling attributes"), and their grouping in problem-specific "model types";
3. *Mechanisms and algorithms*: functionality built upon models, usually to automate some of the procedure steps or to derive new knowledge.

Thus, the work at hand relies on a metamodelling approach whose outcome must be both human-readable (to meet challenge *C1*) and machine-readable (to meet challenge *C2*), and a mapping (challenge *C3*) must exist between these to ensure semantic traceability. Unlike in more traditional requirements elicitation practice, diagrammatic models are not employed as artefacts attached to textual requirements—instead, they become the very representation of requirements, while natural language takes on the role of model annotations. Model queries can be employed to extract textual annotations filtered by the various types of relations that are present in models.

The modelling method proposed in this work is a requirements-focused partition of the broader (and more domain-specific) *ComVantage modelling method*. The broad method is hybrid, in the sense that it integrates parts of *existing modelling languages* (e.g. e3 value [24] for describing business models, elements of traditional control flow diagrams for describing processes) with constructs derived from *semi-formal diagrams suggested by best practices* (diagrams proposed by SCOR [57] for supply chain models, model types used in feature analysis [30] to describe feature variability) and some *newly designed model types* to integrate concepts specific to the ComVantage application areas (machine defect models, app orchestration models).

The full conceptual coverage of the broad method is only suggested here, and only its requirements-oriented fragment will be further described. Its main characteristics are: (a) it approaches *requirements elicitation as a modelling activity*, guiding users to decompose a business view on multiple layers of detail, down to mock-up designs; (b) it enables *knowledge externalization in human-readable form*, channelling communication between stakeholders and app designers; (c) it enables *knowledge externalization in machine-readable Linked Data structures*, thus enabling cross-model traceability for various purposes: model analysis (e.g. requirements gap analysis), interoperability (e.g. requirements-driven app deployment), derivation of auxiliary model-aware content or tools (interaction simulator, requirements manager, etc.).

The broad method is designed in an iterative cycle, as a series of project deliverables available at [17] (deliverables D311, D621, D721, D821, D312, D622, D722, D822). The fragment illustrated in this paper is implemented iteratively as a proof-of-concept modelling tool on the Open Model Initiative Laboratory portal [46].

4 The modelling procedure

4.1 Modelling procedure overview

The modelling procedure provides the backbone for the requirements elicitation process, which thus becomes a model refinement process, with typical scenarios such as: (a) the business stakeholder initiates the dialog with some rudimentary designs on a familiar level of granularity, and then the app developer enhances models and refines granularity; or (b) the app developer translates the stakeholder's user stories in initial models and presents them, integrating feedback or attracting the stakeholder to contribute in a collaborative modelling approach.

Table 1 maps the model types involved in the procedure on the requirements layers mentioned in Sect. 2.1. Section 4.2 will explain the application of these model types through examples from a maintenance scenario, and then Sect. 5 will provide an integrated overview (metamodel) on how they are linked together.

Figure 1 (designed with ADONIS CE [8]) depicts the modelling procedure as a business process (steps to be detailed in the next sections). Two high-level phases can be delimited (only the second being in the focus of this paper):

1. In the *business focus phase* (not depicted in Fig. 1), stakeholders describe their business view in terms of their business model and its domain-specific process motivators (defect models, product models, KPI models, etc.);
2. Then, in the *requirements focus phase,* the business view is decomposed down to operational processes and their required resources, of several types: *liable entities* (business entities for supply chain processes, human roles for operational processes), *tangible assets* (hardware) and *intangible assets* (apps, information, access means, locations, endpoints). The app-related assets are drilled down and refined by the app developer during negotiation loops with business stakeholders. For readability, loops are not visible in Fig. 1, but they can happen for every segment, and for the procedure as a whole (when requirements evolve based on feedback coming from evaluating initial design or implementation, as suggested by the Twin Peaks model [41]).

Once the models are stable, they are exported with the RDF [61] model serialization component, which further exposes them to queries from model-aware components (to be detailed in Sect. 6).

4.2 Example-based procedure details

The modelling procedure was designed to enable a collaborative modelling effort for two actors—a business-oriented modeller (BM) and an app modeller (AM), along the steps presented throughout this section:

Step 1. Business process modelling

Goals To capture the procedural knowledge of the stakeholders, on how their operational processes must be run. To represent user stories in a structured manner.

Relation to previous steps A business process has domain-specific motivators that are modelled in upper layers of the broad *ComVantage* method (not to be detailed by this paper). For example, one or more machine models are linked to a maintenance process describing how

Table 1 Model types for the requirements layers

App requirements layers	Modelling support
Domain layer	*Links to motivator model types*—to describe the business context and domain-specific process motivators: machine models, product models, etc. (not covered by this paper)
	Business process model type—to describe the domain-specific tasks that require app support
	Business entities model type—to describe the stakeholders involved in performing the activities (hence, the roles that require app support) and the ownership of required resources
	ER model type—to indicate the domain concepts
Interaction layer	*Interaction elements model type*—to describe interaction points and mock-ups
	Orchestration model type—to describe usage precedence for various resource types; for apps, it describes how they must be orchestrated along a business process, providing input to a deployment engine
	Interaction process model type—reused with different semantics, to describe user-app interaction flows (and possibly to evaluate interaction usability)
Data layer	*ER model type*—to describe the conceptual structure of the available or required data
	Information space model type—to describe information resources
Infrastructure layer	*App & capability pool model type*—to describe some technical requirements for the required apps (OS, device type)
	Location pool and *Information space model type*—to describe access means to information resources

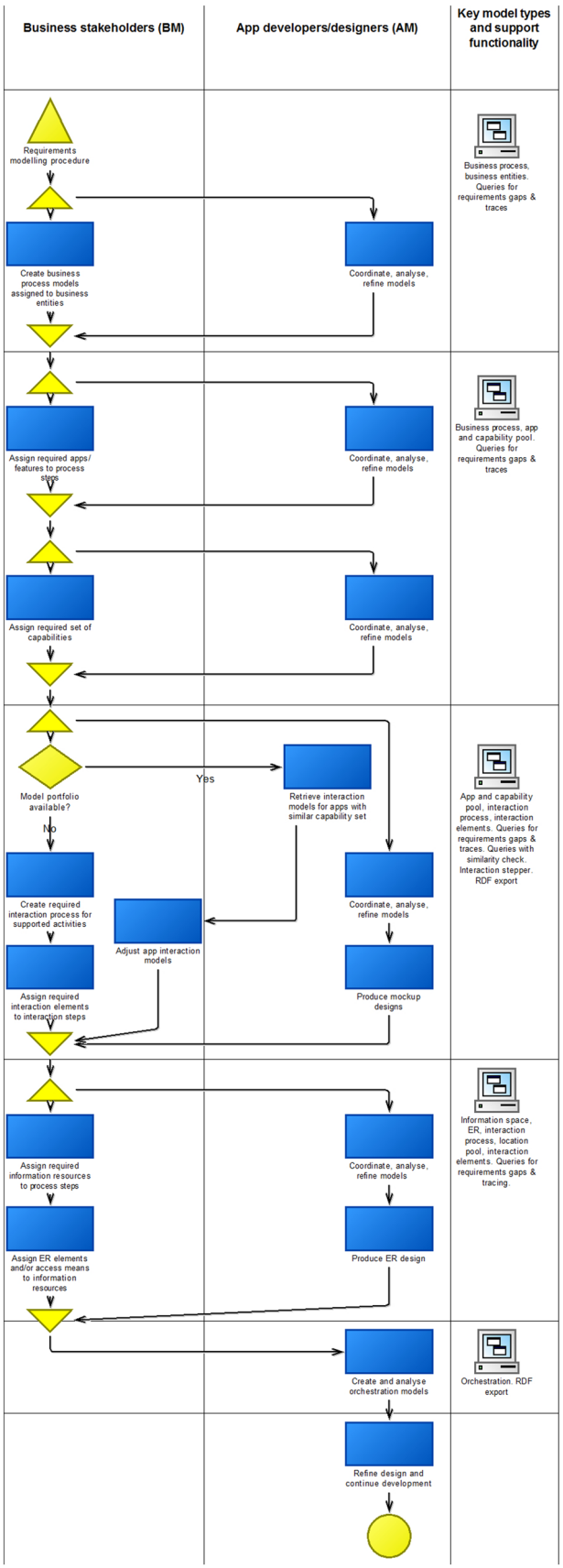


Fig. 1 Overview of the proposed modelling procedure

maintenance requests should be handled for each machine or defect type.

Details BM models an operational business process according to common business process modelling practices. As Fig. 2 (bottom-side) and the subsequent screenshots suggest, the proposed notation supports the core control flow semantics common in most flowcharting languages (activity, decision, parallel splits, swimlanes) but deviates from standard notations to gain two benefits: a) expressivity (e.g. a question mark suggests the concept of "decision" quicker than standard constructs inspired by their IT/logic background—see also [37] for discussions on notation morphology and understandability); b) the notation needs to be ornamented with visual cues acting as hyperlinks to related models. If, for various reasons (compliance, familiarity) a standard notation is preferred, the metamodelling platform employed for this work (ADOxx [9]) allows for defining alternative notations for the same language grammar.

Step 2. Role definition

Goals To capture knowledge about the liable entities/roles participating to the business context and their involvement in process execution.

Relation to previous steps Links are created from business process elements to roles from the business context. Links can be created from roles to the upper layers of the method (e.g. to the business model, supply network).

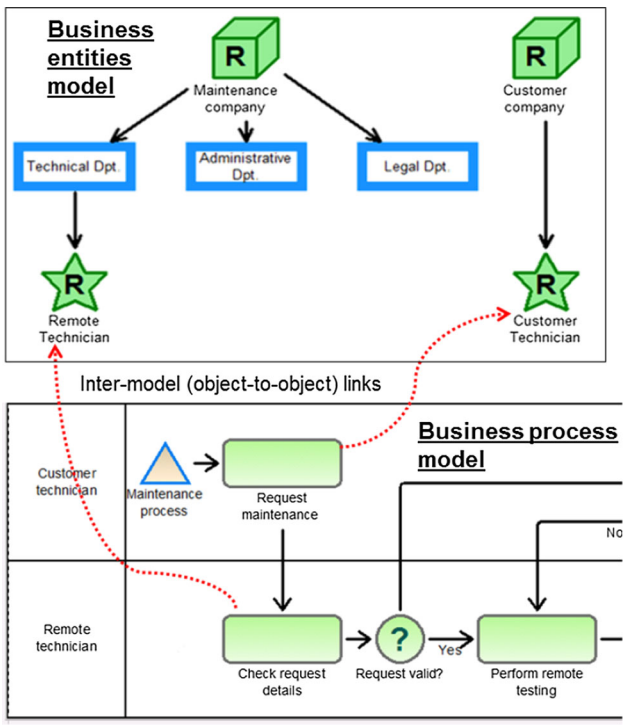


Fig. 2 Mapping business processes to business entities

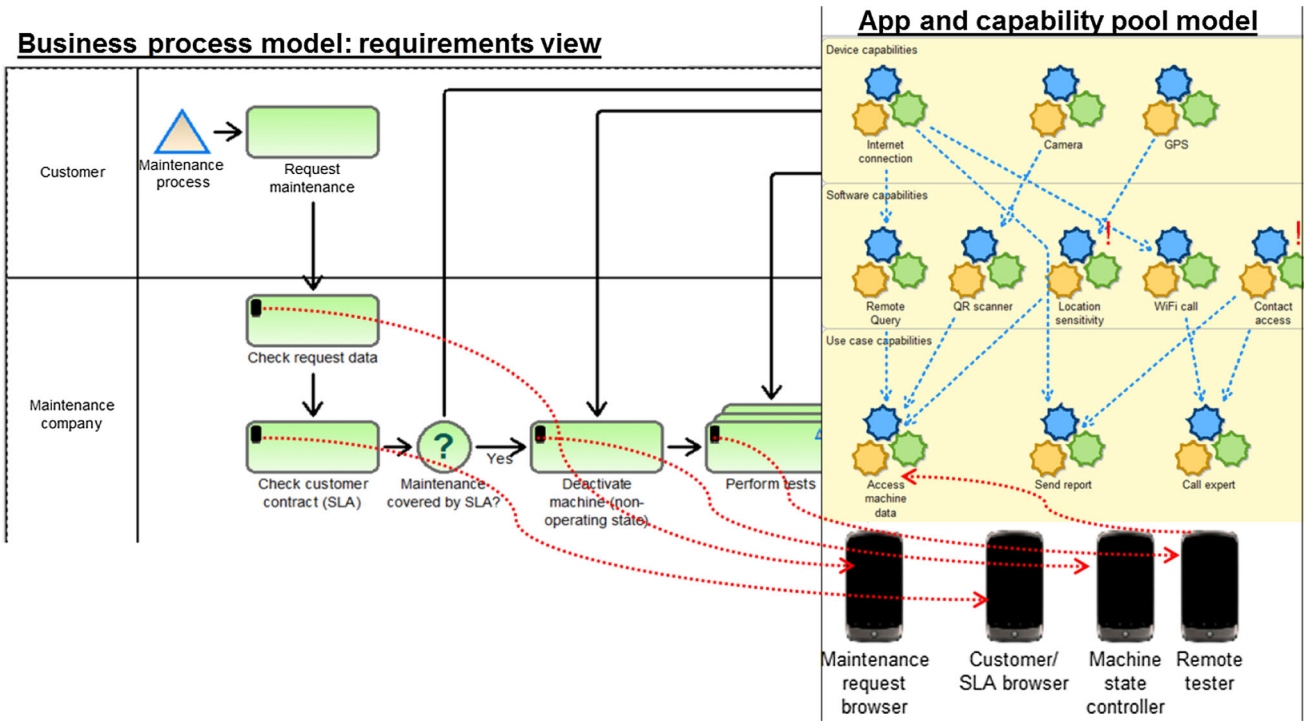


Fig. 3 Allocation of features and capabilities

Details BM models an organizational context (business entities model), indicating roles that are involved in performing the process (see topside of Fig. 2). The role assignment relation is also suggested *notationally* through swimlanes and *functionally* via navigable links from process elements to organizational elements. Their semantics extend beyond what is apparent in this simple example, including also location (where an entity is able to operate) and relations to the upper level's domain-specific model types.

Step 3. Initial requirements allocation

Goals To identify which of its tasks require the mobile IT support and what features are needed.

Relation to previous steps A "variant" link is created between the outcome of Step1 and a refined business process model (*requirements view*) created here. New role links may be identified and linked for this refined process.

Details Our experience has shown that Step 1 usually results in a process model with quite weak granularity (high-level business steps). It is, however, useful to allow Step 1 to be performed with minimal guidance, encouraging BM to describe the process in terms of *what needs to happen*, without thinking of any requirements. Stressing too early the focus on requirements triggers in some cases a "granularity imbalance"—a tendency to mix in the same process model very high-level tasks (e.g. *Perform tests*) with very low-level actions (e.g. *Press refresh button*). As a consequence, the modelling procedure aims to build a sense of decomposition effort, on different layers: Step 1 captures what needs to be done, whereas in Step 3 BM is asked to break down the granularity so that (s)he can assign only one app feature to each process activity.

We understand *feature* here in the sense of "unit of functionality" [3] rather than "quality of the system", since in subsequent steps the modeller must describe how the user will interact with the feature. We assimilate non-functional "qualities" with *capabilities* or capability properties (subject to Step4). Ideally, each feature will become a single-purpose app, so the modeller can be guided by the principle "one app/activity". As it will be later discussed, more complex support can be modelled in the form of an *orchestration* of single-purpose apps.

This step will result in a pool model collecting all the identified apps/features, and a new version of the business process model: *the requirements view*. In Fig. 3, one can notice the visual cues in the top-left corner of some process activities acting as hyperlinks to the features.

By imposing that BM should create two variants of his/her business process model will result in significantly better results than just leading the discussion from the outcome of Step 1. Firstly, it stimulates the modeller to think in different stages about *what needs to happen* and *what support is needed from apps*. Secondly, it forces him/her to go through an explicit step of granularity improvement and to identify clearly which business steps require

support and which do not (the latter can be kept with a coarser granularity, being out of the requirements scope). Finally, it makes him/her rethink the semantics of the "assigned role": while at Step 1, stakeholders tend to think in terms of *responsible roles* (who are accountable for the activity results), in Step 3 they will think more in terms of *performing roles* (who will actually use the apps). This might cause a significant process redesign (more than just breaking down activities to finer detail).

Step 4. Capability definition

Goals To acquire finer descriptions of the required features (including non-functional characteristics and device dependencies). To accumulate a reusable and extensible portfolio of app capabilities.

Relation to previous steps Capabilities are linked to the previously identified features.

Details The semantics of the app/feature concept enable the further collection of technical requirements, as a) *editable properties* prescribed for each app/feature (e.g. OS, device type, link where it can be downloaded) or b) links to an extensible *pool of capabilities*. The capabilities are classified in three categories: *use case capabilities* (e.g. "Access machine data"), *software capabilities* (e.g. "Location sensitivity") and *device capabilities* (e.g. "Internet connection"). Dependencies between use case and software capabilities and between software and device capabilities can also be modelled (right side of Fig. 3) by AM. Some capabilities may be further highlighted as being *privacy-relevant* (e.g. "Location sensitivity", "Contact database access"). The modellers are expected to collaboratively work on linking capabilities and collecting capability annotating properties which may capture various expected qualities (e.g. performance-related).

This step will benefit if the methodology has been employed in other projects and AM has a portfolio/repository of models accumulated from previous experience. Specifically, the pool model should be treated as a reusable, shared, *capability vocabulary* and, at the same time, as a catalogue of "tagged apps", accumulated from AM's experience. It is plausible that most device and software capabilities will be reused, whereas BM may introduce new use case capabilities or perform some linking on use case level (possibly suggesting some subfeatures). It must be noted that device and software capabilities can also be interpreted as *settings/states/configurations* that should be active or switchable for the app/feature to be able to run.

The ComVantage research project advocates the reuse of apps or app skeletons that have been previously designed/implemented and are described through their capabilities. An app orchestration framework [67] relies on this principle, by selecting and adapting app skeletons with certain properties. Model queries on the app and capability pool model can be employed to identify apps with a particular capability set, apps with similar capability sets or capabilities that have similar dependencies (e.g. using a set similarity metric like the Jaccard index).

Step 5. Interaction modelling

Goals To capture interaction requirements. To initiate early designs.

Relation to previous steps Tasks from the business process model are decomposed here in *interaction processes*. The features assigned in Step 3 are decomposed here in *interaction elements*.

Details Business stakeholders familiar with mobile technology or their legacy system are quite often inclined to present their requirements with a sense of flow, for example: *I need to check the machine sensors; if something looks wrong, I need to go to actuators, edit some of their values, then check the sensors again to see what effect the change had.*

BM will again describe such stories as "processes" (see Fig. 4)—but this time the semantics are different: (a) the tasks are not business activities, but *interaction steps* (select, input, read from screen, trigger a new feature, etc.); (b) the resources that must support each task are not roles/apps, but *points of interaction* (to be abbreviated as POIs throughout the remainder of the paper). POIs are described on a level of abstraction positioned between the abstract (inputs, outputs) and concrete (UI controls) layers of the Cameleon framework [14].

Relative to the business process model designed at Step 3, this interaction process will be linked as a *subprocess*. Each of its steps will be linked to an abstract POI that must support the step. There are, however, cases where BM is not capable of giving a sense of flow to the interaction and will only enumerate a list of operations that should be supported. These too, should be represented as POIs. In any case, BM is expected to provide an initial set of POIs that are relevant to the current feature/activity, and an initial mapping between interaction steps and POIs (as in Fig. 4).

For the POIs, we provide a controlled vocabulary that is guided by the following basic *modelling requirements*:

(a) The modeller should not be forced to think in terms of concrete UI control types for a particular interaction modality—for example, UI elements like checkboxes, radio buttons, drop down menus are abstracted as *lists* on which a *selection* interaction is performed;

(b) The modeller should not be limited to the input/output abstraction, but deeper specificity should be allowed. Therefore, both inputs and outputs are further typed to be closer to the concepts used in the natural language discourse of BM;

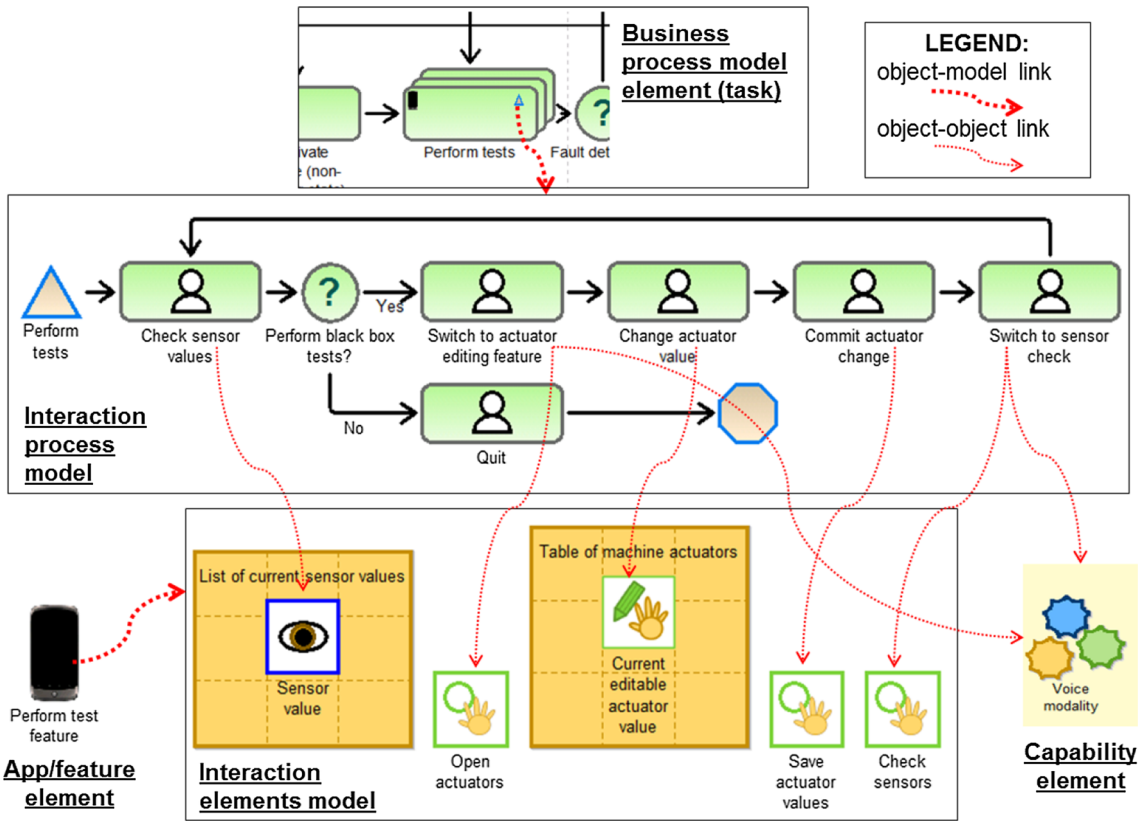


Fig. 4 Semantic links for an interaction process

(c) Emulated UI proposals should be derivable, hence models should retain some degree of UI structural decomposition;

(d) Models should retain domain-specific concepts that normally occur in the natural language requirements of the stakeholder. This requirement will be discussed in Step 7, where data requirements are bridged with interaction requirements.

In this section, details will be given to clarify the abstraction level of the POIs (positioned between the abstract and concrete layers of the Cameleon framework).

In this respect, we have defined a POI taxonomy (Table 2) with several levels of detail, so that BM can choose the detail level (s)he is comfortable with. AM is expected to refine the level of detail and consult BM for feedback.

The semantics of each POI type is not given strictly by the taxonomy, but also by the link type (*Use/Read* or *Change/Write*) between POIs and interaction steps.

1. An *output POI* is an element that is presented as read-only output to the user, without allowing any form of direct interaction. This does not mean that the POI cannot be changed, only that changes cannot be done by directly interacting with it. In this case, the link interpretation is:
   - *Use/Read* means that the user reads the POI content;
   - *Change/Write* means that the user affects somehow the POI or its content without directly interacting with it (e.g. causes a POI to appear/disappear, indirectly causes the update/refresh of a label value by providing input in some other POIs, etc.).

Typical output POIs can have the following structural types:

- *Simple*, meaning that a simple value is provided; the *Event* subtype is particularly distinctive, as it represents the output whose main goal is not content communication, but the notification of the user about a certain state of affairs;
- *Complex*, meaning that complex media content is provided, such as a non-interactive media stream or downloaded document/file. Under this category, we also include any content that the device must output through unconventional peripherals (e.g. content communicated through tactile feedback, 3D printing, etc.). A comprehensive subtaxonomy of these has not been designed, as common interaction metaphors are still relying on visualization and sound. This may change in the future, as unconventional interactivity gains popularity.

Table 2 POI taxonomy

POI interaction types	POI structural types	POI data types	POI format types
Output	Simple	Text	Location
Input			Date/Time
Interactive output			Phone
			URI
			Email
		Number	Integer
			Decimal
			Percentage
			Currency
		Boolean	
		Event	
	Complex	Image	
		Video	
		Sound	
		Document/File	
		Signal	
	Aggregator	Multiplier	List
			Chart
			Table
			Tree
		Component	Screen
			Permanent
			Generic
			Other app

2. An *input POI* is an element that accepts user interaction without communicating, by itself, any content to the user except for an indication of the POI role and existence (e.g. the name of a button/menu option/textbox, an on/off switch, or a status signal showing that the app accepts input). The link interpretation is:
   - *Use/Read* means that the user interacts with the POI without providing any content or affecting its existence, hence it should rather be used with the *Event* data type, meaning that some behaviour is triggered (e.g. pushing a button, activating a link, a menu option, a voice command, etc.). For other data types in most cases, one should use interactive output POIs (as probably they communicate some content to the user);
   - *Change/Write* means that the user affects somehow the POI itself (e.g. causes a POI to appear/disappear) or provides content to it (inputs a value, records media, inputs a drawing, uploads a file, scans an image, etc.). The data type should indicate what kind of input is expected.

   Typical input POIs can have the following structural types:
   - *Simple*, meaning that a simple value (or an event) is expected as input, possibly with some specific format type;
   - *Complex*, meaning that complex content is expected as input, for example: media recording, QR code scan, file upload or unconventional input signal (e.g. fingerprint, input from external controllers, etc.).

3. An *interactive output POI* is an output element that is presented as editable or interactive to the user. Examples can be a prefilled editable text field, an interactive map, a phone call or video call, a text with hyperlinks, an editable document, any piece of media that works as a trigger for other functionality. The link interpretation is:
   - *Use/Read* means that the user reads the POI content and possibly interacts without affecting the contents (e.g. reads an audio/video stream possibly skipping through it, reads a map possibly zooming/panning, reads a document possibly browsing/searching through it, uses a piece of media to trigger some other behaviour);
   - *Change/Write* means that the user affects somehow the POI itself or its content (e.g. causes a POI to appear/disappear, causes the contents to change indirectly, edits the presented value/file/stream, participates in a call, retrieves a new map, etc.).

   Typical interactive output POIs can have the following structural types:
   - *Simple*, meaning form fields that are prefilled and either editable in place or interactive; with the *Event* type, it can represent notifications that require the user attention and input before being dismissed;
   - *Complex*, meaning that complex content is editable or interactive (interactive map, audio/video call, interactive document, editable drawing).

4. *Aggregators* are used to create complex POIs in two ways:
   - *Components* group POIs with heterogeneous semantics. Their presentational style can be *screen* (allowing AM to propose a screen granularity for mock-ups), *permanent* (for permanently available components, such as a status bar), *generic* (grouping POIs that are related or must be reused as a

group) or *other app* (when they represent an existing app);

- *Multipliers* group POIs with homogeneous semantics, to indicate for example that a list or a table of similar POIs should be presented. As a presentational hint, it is also possible to indicate that the data series of such a multi-output must be presented as a chart, rather than value listing. For example, a menu would be a *List* of *input POIs* that should be *Used* to trigger another *List* (submenu) or *Component* (app, pop-up, screen). A data selection list would be a *List* of *interactive output POIs* that should be *Used* to access a "details" *Component*. A list of editable empty textboxes would be a *List* of *input POIs* that should be *Changed*.

It must be noted that the taxonomy is meant to be *modality-independent*, meaning that it depicts types of content ("what"), rather than interaction channels ("how"). This means that a POI of the type *Text* is not necessarily a text to be displayed/typed in, but some text that must be communicated between the user and the app: it can be dictated by voice, written by stylus, touch, or through an auxiliary peripheral; output text POI can be communicated on display, by voice (text reader) or even Morse-encoded in vibrations. Also, the *Complex* types do not represent ways of interacting, but kinds of content that must be transferred between the user and the app: *sound* does not refer to an audio channel, but rather to the availability of some audio content (e.g. the recording of the noise made by the maintained machine).

On the other hand, some object types are strongly bound to specific interaction channels (it is difficult to imagine audio/video content communicated to a human user by other means than sound/vision). Business stakeholders may have difficulties in abstracting away from familiar modalities (or are just not interested in alternatives), and consequently, they will blur the content–channel distinction by grounding requirements to an assumed modality. AM is responsible with identifying whether BM is content-focused or modality-focused and to assure that the interpretation is consistently kept throughout the elicitation process.

An explicit consideration on modality is represented by a dedicated modelling attribute applicable to all POIs—*Presentable*. This is used to distinguish between (a) POIs that must have a screen presence (hence occupying screen space), (b) POIs that must use alternate communication channels and (c) POIs whose modalities are left to the choice of the app developer. As limited display surface is a key issue in mobile app development, capturing this information early in requirements becomes essential input for evaluating design options.

Step 6. Interaction refinement

Goals To provide initial mock-up designs.

Relation to previous steps The previously created POI models are refined to emulate the appearance of a concrete UI.

Details Fig. 4 presented an example from the maintenance use cases of the ComVantage research project, where the following interaction process has been designed: it starts with getting a list of sensor values and then, in case something wrong is noticed, it switches to a testing component where actuator values are edited and saved, and then it switches back to check the effect on sensors. This maps on a *list* of (*output* POIs) sensor values, a (*input*) POI to open the actuator test component, a *list* of machine actuators with prefilled editable values (*interactive output* POIs), an (*input*) POI to save changes to the actuator values and an (*input*) POI to check the effect on sensors.

AM must refine the model by grouping the POIs in screens (*components*) and possibly adjusting the interaction process to accommodate the finished design. Several kinds of adjustments are expected:

- Model query/analysis will indicate gaps to be filled, for example if there are interaction steps with no POI assigned, or POIs that are not involved in any apparent way throughout the interaction. In Fig. 4, for example, AM might decide to add a *Quit* trigger, or might inquire why the interaction process does not contain a machine selection step. To this, BM may bring to light the assumption that the machine should be detected automatically by location, or that it should be identified in a previous step (by a QR code scan, or from the original customer request data).
- Some POIs are introduced only after AM creates a proposal of grouping them in screens. For example, in Fig. 4, it is plausible that BM does not explicitly require the POIs for switching to the actuator testing component and back to the sensor list. These become relevant once it was decided that the two subfeatures will not be displayed on the same screen at the same time. The interaction process has to be extended accordingly once the *screen-switching* elements are added.

Once AM has an initial grouping in screens and a flow that drives the user through the interaction, a UI emulation can be derived by switching the notation from *abstract* to *concrete*. This will give a more suggestive image based on some presentational attributes (colours, fonts, labels, positioning) and an alternate model notation mapped on the presentational types from Table 2.

The mock-up-style notation will also isolate the non-presentable POIs outside the screens. For example, in

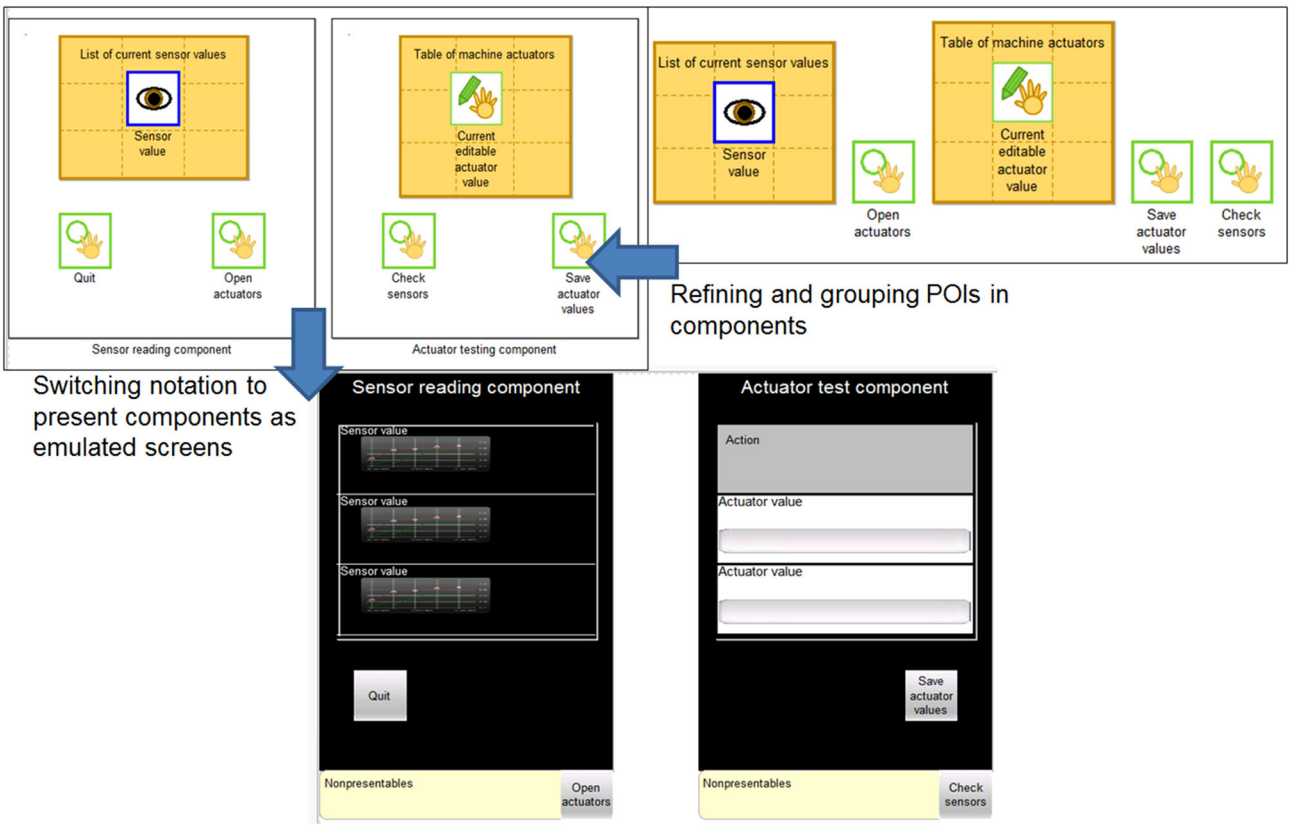


Fig. 5 Shifting between abstract and concrete notations of interaction elements

Fig. 5, AM and BM decided that the triggers that switch between the two screens can be performed by voice commands and should not occupy screen space.

The result is a UI emulation that can be further employed for a model-driven component called *interaction stepper* with the goal of validating requirements or training employees in understanding how the apps will support their workflow (to be discussed in Sect. 6).

The interaction refinement may also involve linking between interaction steps and capabilities (see Fig. 4), for purposes such as:

- to communicate that a certain interaction step (e.g. switching the screen) relies on a capability (e.g. voice modality, microphone); this enriches the possibilities for gap analysis, providing answers to queries such as: *Which requested capability is not used by any interaction process?*
- to communicate that a certain interaction step relies on a configuration setting; settings are assimilated to capabilities and not to POIs since they are not involved in normal interaction processes (instead, interactions rely on some existing configuration).

Step 7. Data requirements allocation

Goal To identify what the required data will be about. To initiate early designs for the domain concept model.

Relation to previous steps The previously identified interaction steps are linked here to elements of the information space.

Details BM is further involved in describing the data requirements, using the same process-centric approach. This again assures that no step of the flow is left without a statement on what is the input and output necessary to perform the activity, in a manner not unlike the one advocated (on a more superficial level) by the SIPOC modelling practices [54].

The data requirements can be allocated with superficial granularity on the outcome of Step 3 (the requirements view on the business process), or in more detail on the outcome of Step 5 (the interaction processes), depending on the knowledge granularity of BM regarding the data.

Just as with the apps, in both cases, there are two levels of detail on which data requirements can be described, with the AM being in the position of refining initial, intuitive BM proposals. On the more superficial level, the modeller may link *information resource* objects, while on the more granular level *data entities* from a simplified ER diagram can be captured. A simplified ER representation is abstract

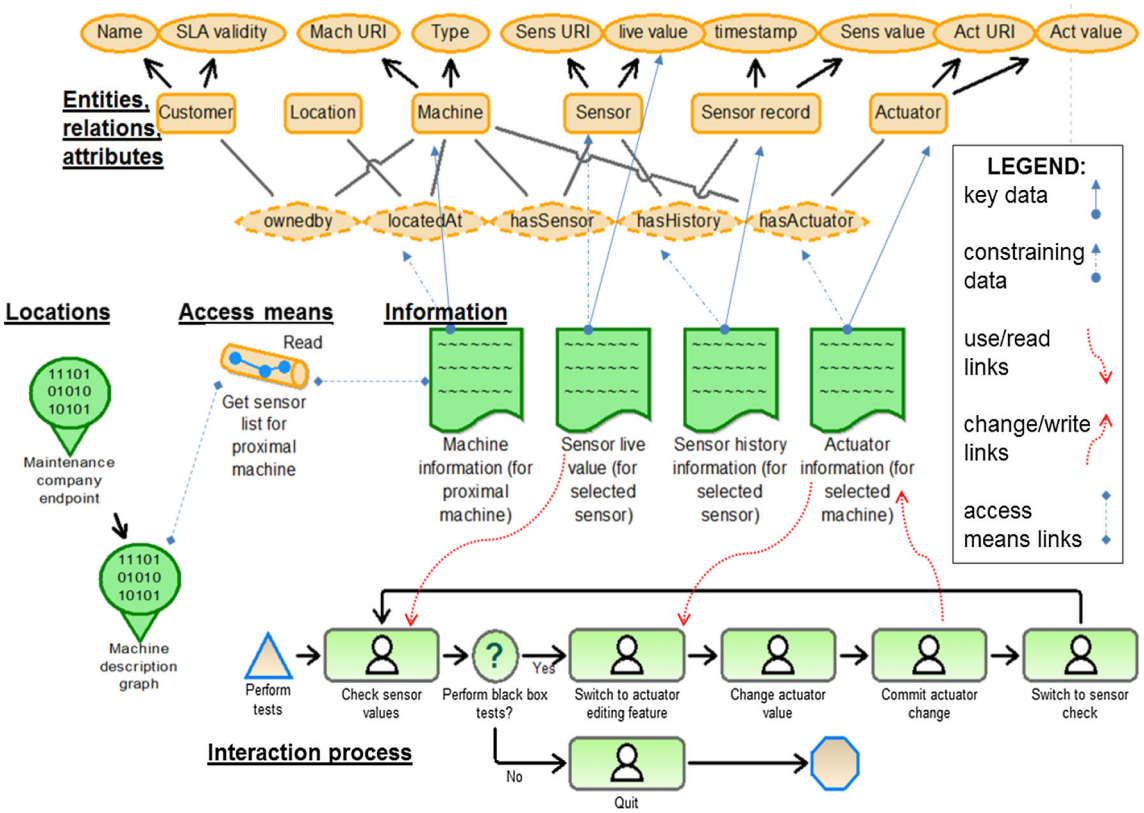


Fig. 6 Modelling the data requirements at interaction process level

enough to describe domain concepts regardless of how the data will be stored, successfully covering Linked Data, relational database schemata or XML. For BM, the elements of an ER diagram are the nouns, attributes and predicates used in describing *what* (*property*) *must be accessed about what* (*thing*). This will become an entry point for the later data design phase (it is not expected that BM will give a complete ER design, but rather a rudimentary pool of concepts, properties and relations). The reusability argument raised in Step 4 (with respect to an app portfolio) is more prominent here: it is plausible that mobile apps must be developed as front-end for already existing data. In such cases, it is even possible that an ER diagram is available upfront, and BM can perform mappings on it. If this is not the case, BM should try to define ER elements to intuitively express what information should be available about what "things".

Figure 6 illustrates such mappings on the interaction process resulted from Step 5.

The granularity of the *information resource* objects is not constrained in any way, and they can be created without any ER mappings, although mappings make the data requirements much more precise. For example, Fig. 6 contains a *sensor live value* resource mapped precisely on the *live value* attribute of the *Sensor* entity, whereas a *machine information* object is mapped more vaguely, on the *Machine* entity as a whole (it is, however, specified that the information should be constrained by the machine–location relation). BM should try a mapping as granular as possible, by translating in model form statements that are recurring in natural language requirements, such as *I need to see/change the* (*properties*) *p1, p2,… of* (*entity*) *E*. There is also a possibility of modelling first-hand constraints (e.g. machine *of current location*, value *of current sensor*) but not any deeper than this, since complete constraints would require dedicated constraint modelling, hence pushing the approach towards code generation, which is out of scope. We rely on natural language model annotations to complement any details not graspable through models.

Some interaction steps require data to be read or written. In order to describe how data are handled by an interaction step, modellers will rely on the same semantic links as in the case of the POIs (*Use/Read* means that data must be retrieved, *Change/Write* means that data must be changed). The fact that an interaction step is linked on one side to a POI and on the other side to an information resource gives to the interaction process the quality of a "traceability bridge" between front-end components and back-end data (see Fig. 7). With the proper granularity, the models will express what data need to be accessed through each POI.

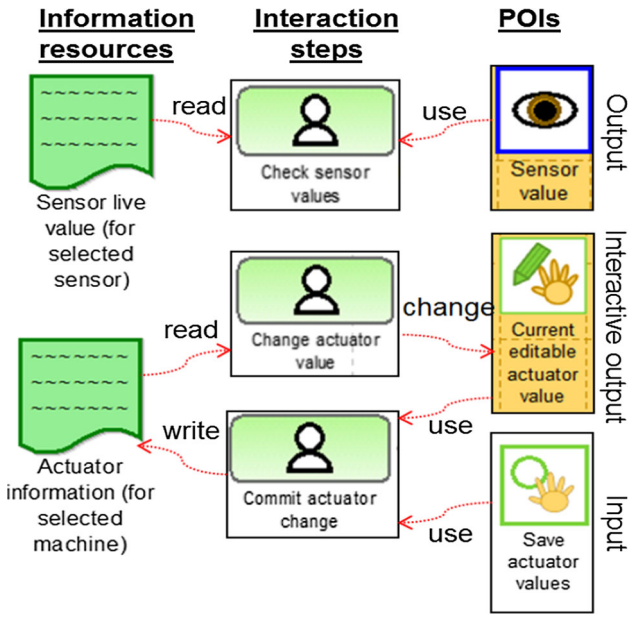


Fig. 7 Bridging data requirements and interaction

In addition, Fig. 6 also highlights the possibility of indicating, for an information resource what *access means* (queries) are needed/available for what *location* (URL of an endpoint or Linked Data graph). These are technical details that AM will add in order to define infrastructure-related requirements (what endpoints are needed/available, what queries do they accept—concerns particularly relevant when dealing with Linked Data and Web Services to insure that the available information space fulfils the data requirements).

Step 8. Interaction process extension

Goals To initiate designs for back-end connectivity.

Relation to previous steps The previously designed interaction process is extended with non-interactive steps where some back-end operation must take place.

Details In order to bridge the procedure towards the software design phase, the interaction processes can be drilled down by AM, who will separate the steps performed by the user (*interaction steps*) from those performed by the app (*functions*).

The outcome is an important entry to the app design process. Technical stakeholders from the business side can be involved in establishing common understanding of such an execution flow, as it is still less formal and rigorous than approaches oriented towards code generation (e.g. sequence diagrams). It is also a direct extension of the requirements representation and an enabler for a common understanding on access means requirements.

Some of the links created in Step 5 (to POIs) and Step 7 (to data entities) will have to be shifted from interaction steps to newly introduced app functions that will actually perform data queries and trigger changes in the user interface. For example, the retrieval of sensor values and the creation of the POI displaying them become the responsibility of the *Retrieve sensor values* function which must be inserted before the *Check sensor values* interaction step (Fig. 8).

Such adjustments will highlight aspects relevant both for the business stakeholders and the app developers:

- that some interaction steps will not be instantaneous, but conditioned by some back-end activity and possibly waiting times;
- that between interaction steps there will be some queries requiring access means and available endpoints;
- discrete event process simulation, usually employed for time/cost estimations over business process paths [8] can be repurposed for interaction processes with a different interpretation on time (e.g. processing or query effort) and cost (e.g. back-end data payloads, number of clicks/touches in the interaction steps).

Step 9. Orchestration modelling

Goals To identify the workflow of each role/entity participating in a business process.

Relation to previous steps: Orchestration models must rely on the outcome of Step 3 (process-role-feature mappings).

Details An essential goal of this work is to assure that requirements are usable beyond the elicitation phase. This is achieved by:

- enabling the requirements elicitation phase to flow into the early app design phase, allowing evaluations on rudimentary designs (analysing model-based mock-ups or identifying gaps in the semantic links between models);
- supporting a *semi-automated orchestration of apps driven by the business flow*, a key approach of the ComVantage research project introduced by [67] and based on the assumption that each feature is implemented in a single-purpose app.

An orchestration engine that deploys inter-connected apps takes input from an RDF serialization of models where the usage precedence of apps (as dictated by the business process) is described. For this goal, a diagrammatic *orchestration model* must be built (manually, or derived automatically from the business process model, as it will be highlighted in Sect. 6.2). In the current implementation, the modelling tool provides options for deriving an orchestration model for (a) an entire business process or (b) only for one role involved in a business process. Role-level orchestration is particularly important, as each role is

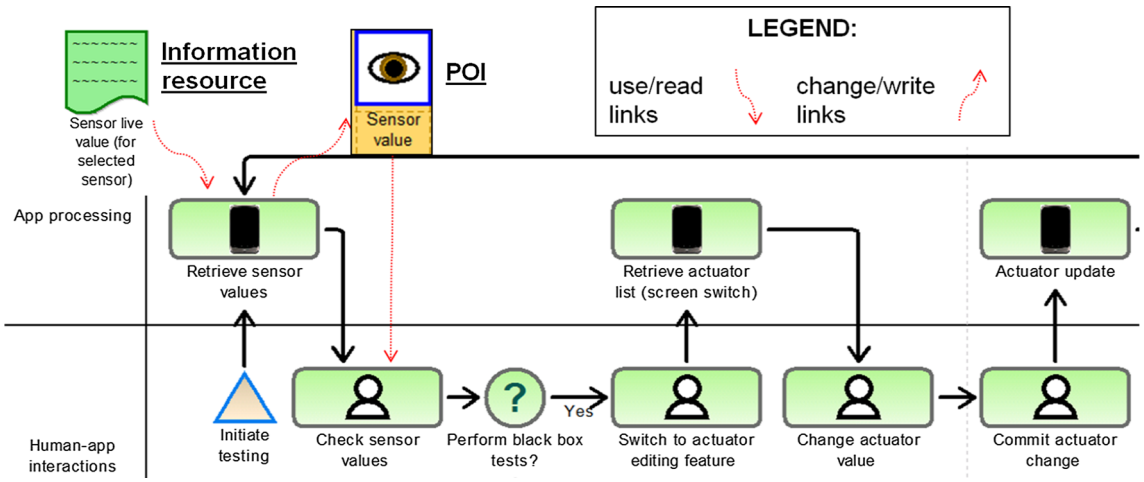


Fig. 8 Extending interaction process with app functionality

assumed to use a different device with different functionality.

Figures 9 and 10 showcase an example of a business process model (roles indicated by swimlanes) and the orchestration models that can be derived from it considering the allocated app requirements. As Fig. 10 suggests, the orchestration model loses the distinction between XOR and AND splits found in the original process model. This goes with the assumption that the splitting app (e.g. "History data analyser") is always treated as an OR, displaying to the user the choice list of paths to be followed and letting him/her decide on how to advance. Hence, we rather aim for a usage precedence recommendation derived from requirements, rather than a fully automated workflow. Since we discuss here about front-end execution based on human interaction, the notion of workflow automation is not as prescribing as in the case of Web service orchestration. However, investigations are still being carried out regarding the degree of prescription that is useful in a multi-app execution flow, considering also the control transfers between the apps serving each role (not in the scope of this paper).

5 The modelling language

5.1 Metamodel overview

Figure 11 brings all the previous examples together by providing an overview on the *metamodel*, on how it was partitioned in model types and what semantic links have been created to enable cross-model traceability. The metamodel groups the different visual constructs (concepts) in model types and prescribes relations of various kinds. The figure distinguishes only between *inter-model links* (commonly implemented as hyperlinks or cross-references) and *intra-model relations* (commonly implemented as visual connectors, object groupings, intra-model hyperlinks). Concept generalization is also highlighted (see the legend). These relations have been inspected in Sect. 4.2 with respect to various model fragments and how they enable the flow of the modelling procedure.

While a standard UML class diagram could be misused to present the content of Fig. 11, the semantics and usage are different—we do not deal here with OO classes and associations; instead, the intended usage is to track and query the evolution and different abstraction levels for the metamodel governing the different diagrammatic samples presented throughout Sect. 4.2, hence we rely on this custom language to manage the metamodel (it is created on the same framework as the method under scrutiny).

The metamodel is the "gluing" semantic core of this work, as it also captures modelling relations to be externalized as Linked Data, thus enabling the semantic traceability. The tracing itself is performed by semantic queries (SPARQL [65]) over a hypergraph derived from the interlinked models, as it will be explained in Sect. 6.1. Towards this goal, the next section will introduce a modelling relation taxonomy that must be considered when transferring model information to the hypergraph data model (RDF).

5.2 A modelling relation taxonomy

The key enablers for semantic traceability are modelling relations. They are present in diagrammatic modelling in a variety of forms: implied or explicit, for human interpretation or for machine processing, etc. With respect to the building blocks of a modelling method as defined by [31], we introduce here a classification of modelling relations.

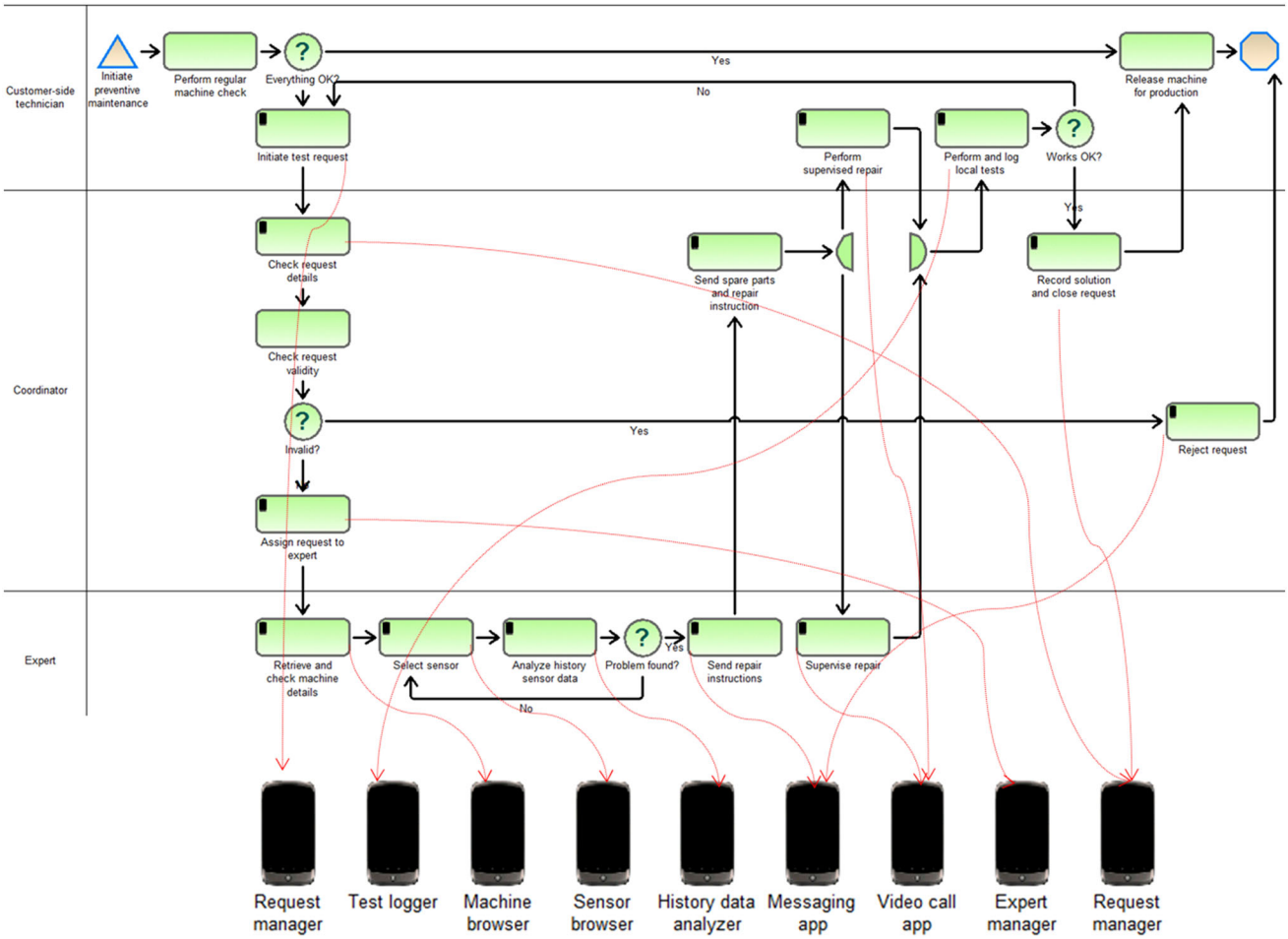


Fig. 9 Multi-role app-supported business process model as input for orchestration models

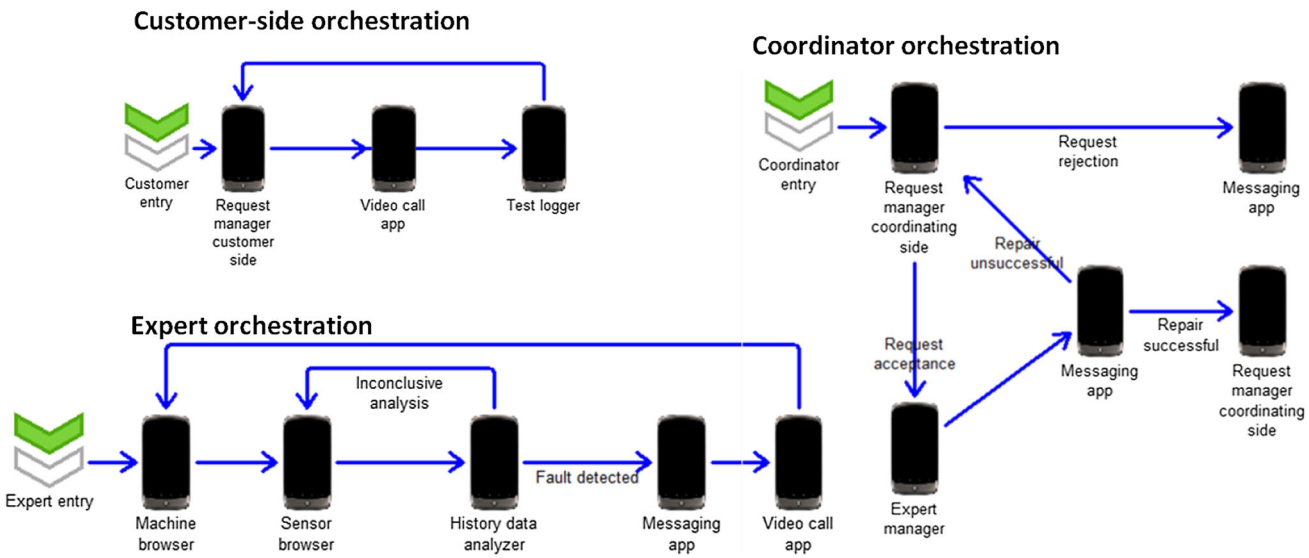


Fig. 10 Orchestration models derived from Fig. 9

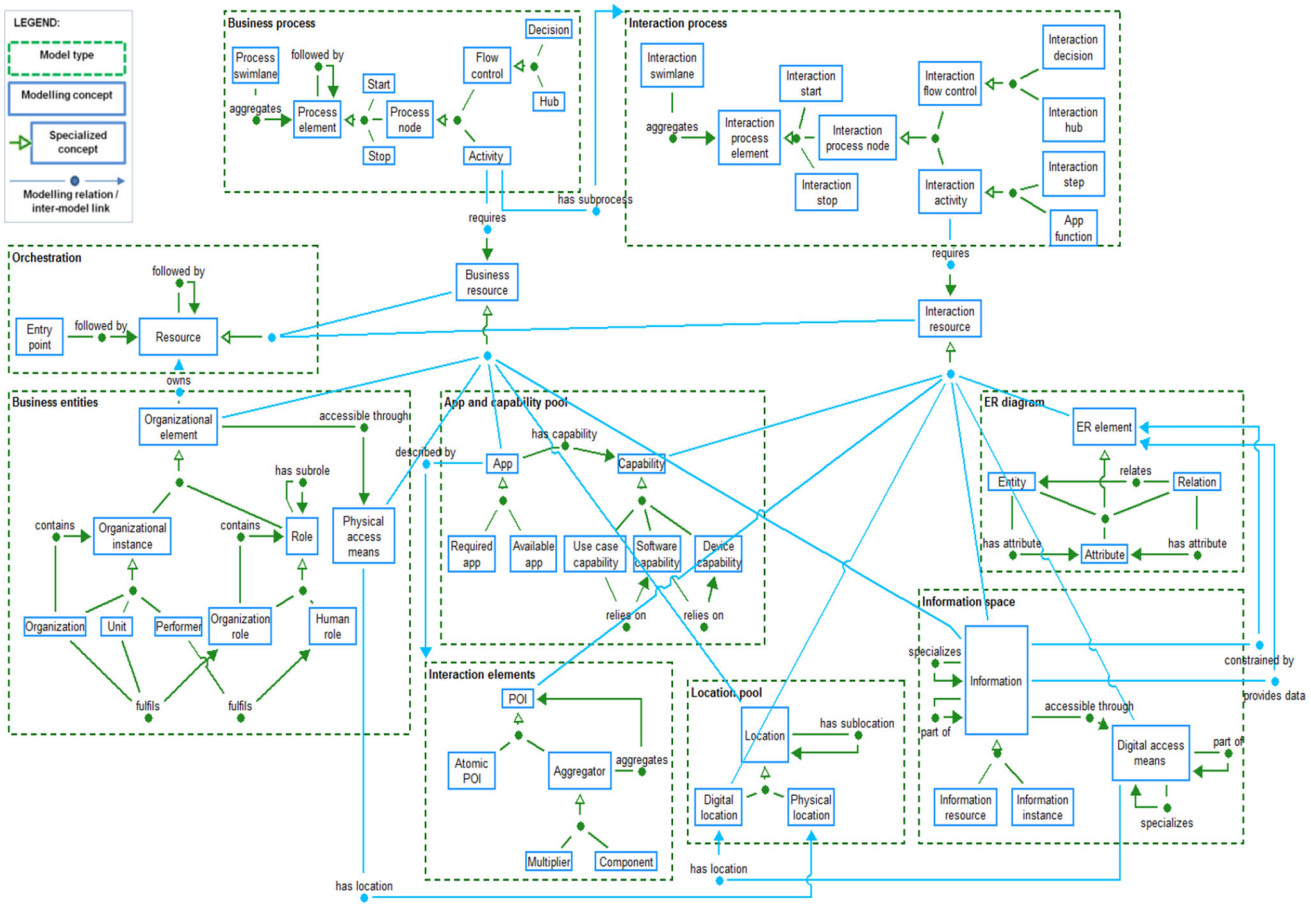


Fig. 11 Metamodel overview

Abstractly, a (binary) relation is defined as a set of ordered pairs, or through its indicator function:

$$R \subseteq X \times Y \text{ or } R : X \times Y \rightarrow \{0, 1\}$$

A conceptual *modelling relation*, however, must consider some pragmatic aspects:

First of all, X and Y are elements of the language alphabet, terminals representing concepts (from which modelling objects are instantiated). The alphabet is also the basis for the syntactical well-formedness rules. Some modelling tools also allow to capture the "semantics" of modelling objects in property sheets ("shape data" in MS Visio [35], "notebook attributes" in ADONIS [8]) where literal data are assigned to properties (hence Y becomes a data type).

More important for the work at hand is that modelling relations are not defined only by their participants and direction—we also have to consider their pragmatic goal of "communication and understanding" [39]. Hence, a defining characteristic of modelling relations is the level on which they fulfil this goal, the way they are perceived by the user. A modeller may grasp (understand) the existence of a relation from different aspects: some construct visible on the modelling canvas, by inspecting some off-canvas annotations, by following a hyperlink between related models, etc.

We map these levels on the modelling method building blocks established for the *modelling method* notion in [31] and then further drill down to a taxonomy comprising the following variants of modelling relations (see Table 3 for illustrative examples):

- *notationally induced* (*NotR*), covers relations perceived by the modeller from the way in which modelling objects look or are arranged on the modelling canvas;
- *syntactically induced* (*SynR*), covers relations induced by the well-formedness rules of the language (how modelling objects are allowed to connect on the canvas, their prescribed properties);
- *semantically induced* (*SemR*), covers relations implied by other relations or stated explicitly with non-modelling means (e.g. annotations);
- *functionally induced* (*FuncR*), covers relations perceived from the usage of some functional feature of the modelling tool (e.g. navigating a hyperlink).

Semantic traceability across models is achieved when relevant modelling relations of these types are made explicit and machine-readable in a format that enables cross-model queries. Therefore, modelling methodologists must set up mechanisms to produce traceable representations for these four relation variants. A shortcut recommended by this work is to perform a consolidation between these relation variants whenever they are meant to be perceived as the same abstract relation (R). Hence, for the reader of a model, the following must hold:

$$(NotR(x,y) \rightarrow R(x,y)) \wedge (SynR(x,y) \rightarrow R(x,y)) \wedge$$
$$\wedge (SemR(x,y) \rightarrow R(x,y)) \wedge (FuncR(x,y) \rightarrow R(x,y))$$

If this is the case, the variants should be consolidated and only one relation should be externalized for queries (otherwise, the methodologist must preserve intended distinctions). In the methodologist viewpoint (as well as the model creator's), the implications are reversed, since he/she needs to enable (or create) at least one of these variants for each conceptual relation that should be communicated:

$$(R(x,y) \rightarrow NotR(x,y)) \vee (R(x,y) \rightarrow SynR(x,y)) \vee$$
$$\vee (R(x,y) \rightarrow SemR(x,y)) \vee (R(x,y) \rightarrow FuncR(x,y))$$

Means of performing the consolidation are, however, platform-dependent. To facilitate implementation, we have opted for ADOxx [9], a metamodelling platform that provides built-in scripting means for deriving these relation variants one from another. A typical example is the following (the implications express the methodologist viewpoint):

1. a *syntactically induced relation* is created as a reference that is prescribed for some concepts of the metamodel; for example, from business process *Activities* to human *Roles* in organizational charts:

   $R(ac, ro) \rightarrow SynR(ac, ro)$

2. this can become *a notationally induced relation*, if the notation reflects the existence of the reference with some visual cue; for example, the referred *Role* is displayed under the *Activity* symbol:

   $SynR(ac, ro) \rightarrow NotR(ac, ro)$

3. it further becomes a *functionally induced relation*, if a hyperlink is created to navigate between models, using the visual cue as anchor;

   $NotR(ac, ro) \rightarrow FuncR(ac, ro)$

4. it is also a *semantically induced relation*, if such references are generated automatically from the existence of other relations; for example, when a process *Activity* is inside a *Swimlane*, and that *Swimlane* has a *Role* hyperlinked to it, then the Role can be assigned automatically to the *Activity*:

   $NotR(ac, sw) \wedge FuncR(sw, ro) \rightarrow SemR(ac, ro)$

   $SemR(ac, ro) \rightarrow SynR(ac, ro)$

It must be noted that this relation consolidation should not be taken for granted: hyperlinks can be generic, unconstrained by the language grammar; visual cues can be displayed without supporting any navigation; machine-readable relations can be created without having a hyperlink behaviour. From a tool developer viewpoint, the consolidation of relations might be a usability decision, but for the end-users it facilitates model understanding, since it enables a consolidated cognition of relations manifesting within and between models.

The format of choice for externalizing the consolidated relations for this work is RDF, a relation-centred representation that can be queried with graph-oriented means (the SPARQL language). The outcome is a hypergraph of model information to be further discussed in the next section.

6 Mechanisms and algorithms

Modelling functionality includes mechanisms and algorithms that support the modelling procedure and the refinement or validation of the modelled requirements. We highlight here the most important ones, as they have been implemented in a proof of concept available in the Open Model Initiative Laboratory (the modelling tool prototype is available on registration basis at [39]).

6.1 The RDF exporter

This is the serialization tool that exposes all models in a Linked Data format based on an RDF Schema representation of the metamodel from Fig. 11 and on the meta-metamodel of the ADOxx platform [9] (used for the modelling tool implementation). Implementation details for this component are available at [17] (deliverables D311, D312), the proof of concept and its usage guidelines are available at [46]. Its aim is to enable model processing outside the modelling tool (the other features described in this section also make use of it). Its RDF vocabulary evolves in synchronicity with the metamodel evolution, following several basic principles: (a) each model type prescribed by the metamodel becomes a named graph (in the TriG [7] and TriX [15] syntaxes); (b) each modelling concept becomes an RDF class; (c) each modelling relation becomes an RDF class if it does have its own editable

Table 3 Proposed relation taxonomy for modelling languages

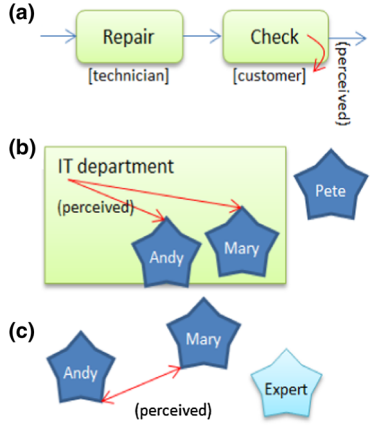
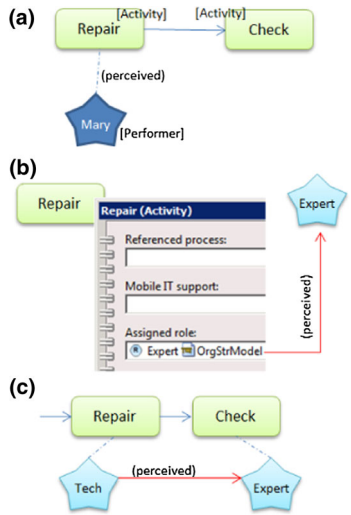
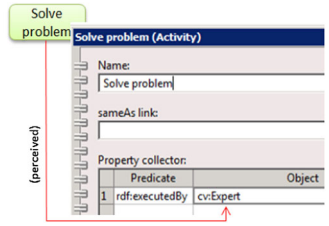
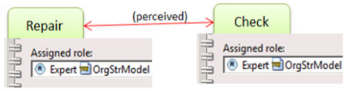
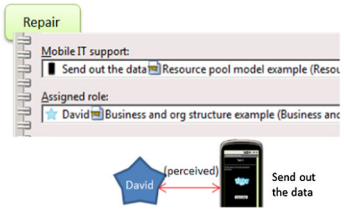
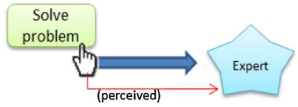
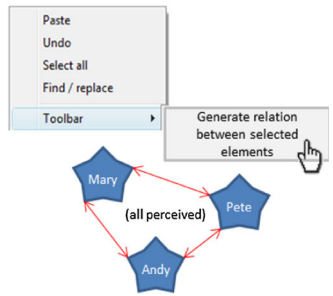
Variants	Subvariants	Mock-up examples	Explanations on the examples
1. Notationally induced relations	Relations stated explicitly in visual cues Relations implied by positioning Relations implied by notational similarity		(a) In a business process model, visual cues establish in the user's mind a relation between process activities and the responsible roles; (b) By grouping Andy and Mary inside the IT department rectangle, a relation is implied between the container (the department) and the contained (the persons, except for Pete). We also include in this category any relation that can be implied by positioning, for example: a *proximity* relation generated between elements that are close enough (under a certain threshold), an *alignment* relation generated between all elements aligned to a certain prescribed curve or axis, a *grouping* relation between all elements that occupy a certain area (or are part of the same model); (c) By using similar colour, the modeller implies a relation between Andy and Mary, while distinguishing Expert as a different object. The same could be applied to shape or size
2. Syntactically induced relations	Relations stated explicitly on the modelling canvas Relations stated explicitly outside the modelling canvas Relations implied by canvas relations (inferred from syntax)		(a) In a business process modelling language, a *flow* visual connector (arrow) is prescribed between activities, and a *responsible* visual connector (dashed) is prescribed from performers to activities. The metamodel enforces domain and range constraints. The relations are manually created using visual connectors that comply to the syntactical well-formedness of the language; (b) In a business process modelling language, a machine-readable reference can be created to an expert (from other models), using a property sheet outside the modelling canvas. The relation is again prescribed by the language, using domain and range constraints, but it has no representation on the modelling canvas; (c) In a business process modelling language, a direct visual connector is derivable between responsible roles that are consecutively linked to the same business process model. Hence, we have in this category visual connectors that are prescribed by the language syntax but inferred from other syntactical constructs

Table 3 continued

Variants	Subvariants	Mock-up examples	Explanations on the examples
3. Semantically induced relations	Relations stated explicitly with non-modelling means Relations implied by semantic similarity Relations implied by non-canvas relations (inferred from semantics)	(a)  (b)  (c) 	(a) Explicit relations can be written down with *non-modelling means*. In this example, an object is annotated with an RDF triple establishing a relation from the business process activity to an assigned expert (which might be available in another model; or, it can be just an abstract reusable resource). Other non-modelling means may include natural text annotations stating the existence of such relations in a more or less structured way (which influences their machine-readability, hence traceability); (b) In a business process modelling language, a relation is implied between activities which are already related to the same responsible role (*Expert*) by other means than well-formed connectors; (c) A business process modelling language allows for an activity to have assigned mobile apps and responsible roles (using a property sheet). By combining these assignments, a relation is implied between the responsible role and the mobile app of each activity
4. Functionally induced relations	Relations implied by navigational features Relations derived by any other modelling tool functionality	(a)  (b) 	In this category, we have explicit relations that are determined on tool level, rather than language level (a) A hyperlink enabling navigation between models or modelling objects implies a relation. Here we have a relation between the *Expert* role and the *Solve problem* activity; (b) In this catch-all category, we include any relation that is not induced by any of the previously indicated aspects (notation, language grammar, properties of the modelling objects). For example, a menu option could allow the generation of arbitrary relations between objects that have been previously selected, searched or filtered in some tool-specific way that did not consider the modelling language

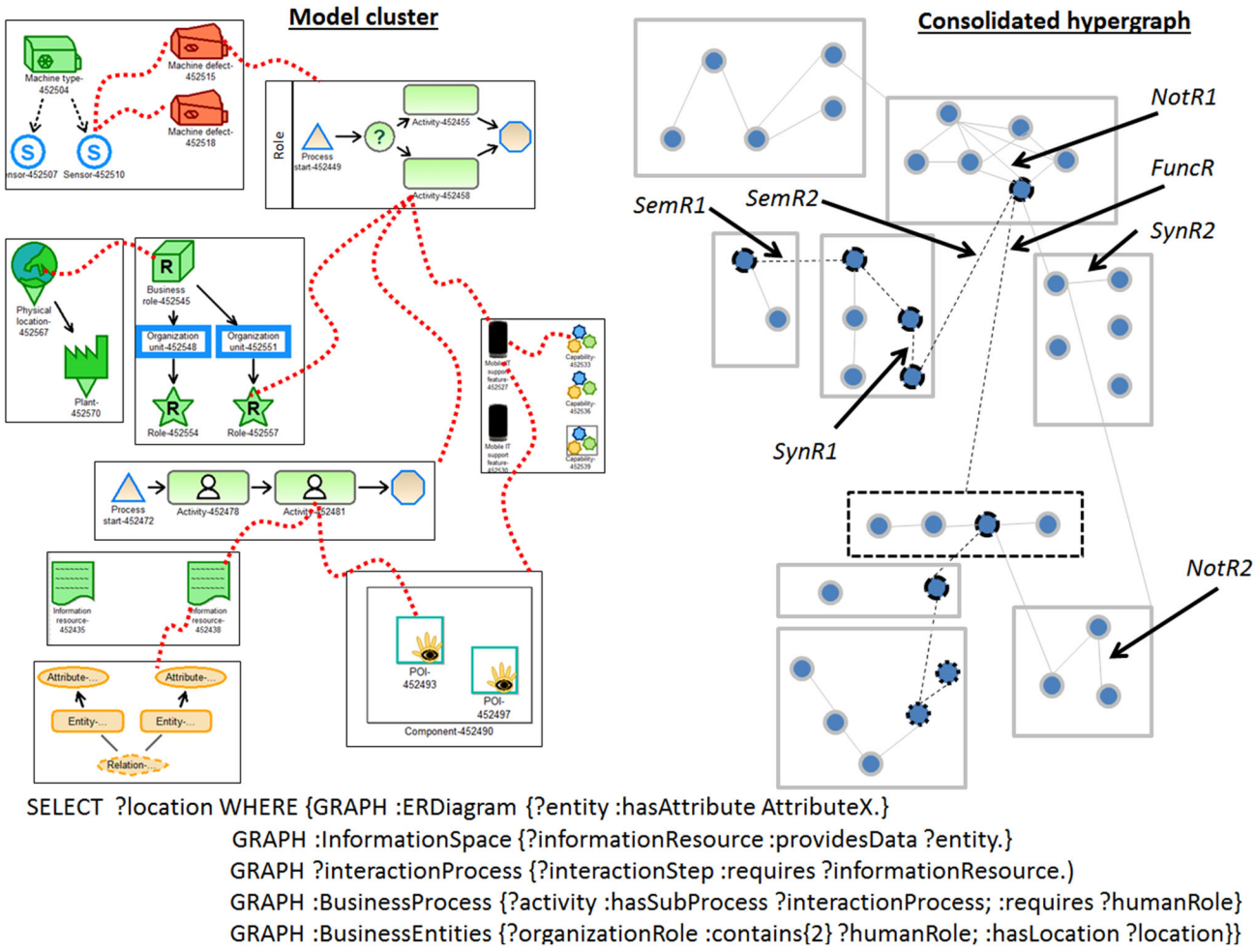


Fig. 12 Requirements tracing through SPARQL model queries

properties, and an RDF predicate if not; (d) each modelling attribute becomes an RDF predicate; (e) each inter-model link becomes an RDF object property bridging resources from different named graphs.

This enables the use of Linked Data technologies to query models or to manipulate the knowledge captured in models; therefore, it also enables the implementation of systems or services that are aware of the above-mentioned variants of modelling relations. Thus, the RDF exporter is a key enabler for semantic traceability across the different types of models. Figure 12 depicts traceable relations across a set of models covering the various aspects that have been presented in Sect. 4.2 (with the exception of the top model, which is a domain-specific "process motivator"—it describes types of machines and defects, and it links them to some designated maintenance process).

On the right side of the figure, the models are reflected in the consolidated hypergraph resulted from the RDF export. The dashed edges highlight an example of a SPARQL query that traverses the graph (its code also visible in the figure). The query traces the location of the business entity that requires a particular attribute (AttributeX) from the ER model (the model object names are not meant to be readable; however, familiarity with the already presented notations is needed to grasp the general structure and how it translates to the hypergraph).

We highlight here some relation variants that are captured (and highlighted) in the hypergraph, using concepts from the metamodel:

- *NotR*1 $\subseteq$ *ProcessSwimlane* $\times$ *ProcessElement*. It represents the visual containment relation between process swimlanes and their contents (in a business process model);
- *NotR*2 $\subseteq$ *Component* $\times$ *POI*. It represents the visual containment relation between a UI "aggregator" component and a POI element (see Table 2);
- *FuncR* $\subseteq$ *Activity* $\times$ *InteractionProcess*. It represents the hyperlinks that enable navigation from a process activity to its interaction process;

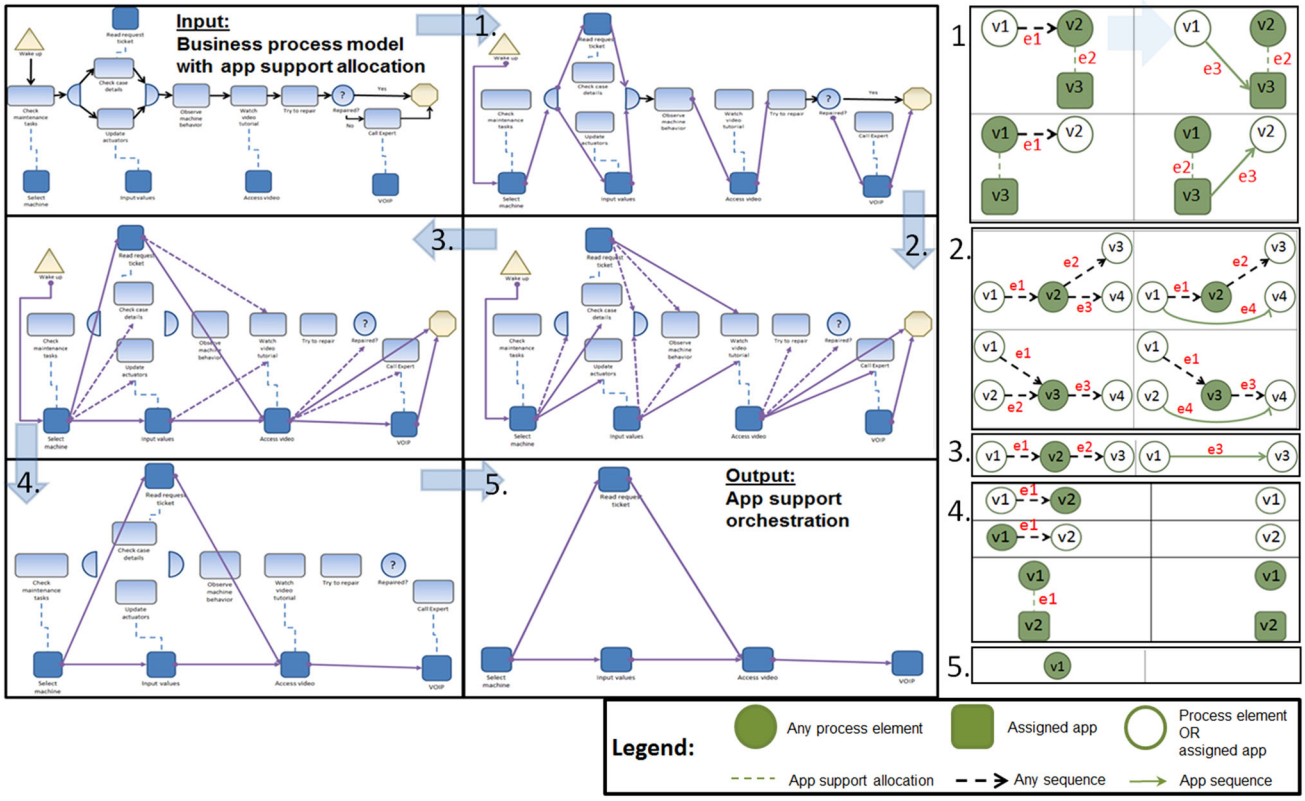


Fig. 13 Graph transformation rules for orchestration models

- *SemR*1 $\subseteq$ *BusinessEntity* $\times$ *Location*.
  It represents the location of a business entity, derived from the presence of an intermediate "access means" object;
- *SemR*2 $\subseteq$ *Activity* $\times$ *BusinessEntity*.
  It represents the relation between an activity and a responsible role, derived from the inclusion of the activity in a swimlane associated to that role;
- *SynR*1 $\subseteq$ *Organization* $\times$ *HumanRole*.
  It represents the visual connector between an organization and a subordinated role;
- *SynR*2 $\subseteq$ *App* $\times$ *Capability*.
  It represents a prescribed reference that can be established between apps and their capabilities.

It must be noted that queries traversing these relations can also retrieve data from the editable property sheets of each modelling object. These are typically domain-specific attributes (e.g. OS for an app) or attributes relevant for requirements management (e.g. build status, author, textual annotations, etc.)

6.2 The orchestrator

This was implemented as automated support for Step 9 of the modelling procedure, both for the full process and on a "per-role" basis (in order to filter only the app support required by a certain role).

Formally, this is based on a graph-rewriting mechanism with a set of rules applied in the sequence indicated by Fig. 13 (on the left side, the model element names are not meant to be readable, only the structural changes). It can be executed in various fashions, either outsourced from the modelling tool (through SPARQL-based transformations, through external graph-rewriting libraries), or algorithmically implemented directly in the modelling tool (current implementation uses the proprietary scripting language of ADOxx).

On the right side of Fig. 13, the transformation rules are depicted graphically in terms of node types and edge types, numbered in their execution order (each rule is applied throughout the entire model until the end node):

Step 1	Process elements having apps (requirements) assigned are bypassed through their app objects;
Step 2	Process splits and joins are bypassed, to the nearest following app objects;
Step 3	App objects get direct "sequence" relations between them, removing any intermediary process elements lacking requirements;
Steps 4–5	deal with the removal of bypassed nodes and of app assignment relations, so that only the orchestration model is left.

In the context of the ComVantage research project, the RDF serialization of the orchestration model is then read by an orchestration engine that deploys concrete chained apps according to the required business flow. Thus, business stakeholders can redesign business processes and redeploy the same suite of (adapted) apps according to process changes [67].

It is important to note that the orchestration can be generalized for any type of resource that is linked to process steps, as described in more detail in [11]:

Orchestration of roles will reflect the role switching performed during the process execution. *Orchestration of locations* will reflect the location switches necessary for a process, highlighting the need for mobility if locations are physical. If locations are digital (endpoints or organization domains), it will provide a justification for using Linked Data (as a facilitator of data federation at query time). *Orchestrations derived from the resource ownership* relations will reflect how asset ownership switches along the business process. If we talk about data ownership, the benefit of Linked Data for facilitating queries over multiple organizations is again highlighted. Therefore, the generalization of orchestration can be used to advocate the need for the ComVantage technological specificity (mobility and Linked Data). Arguments in favour of Linked Data may also include the general spread of the ER data model over multiple organizations.

6.3 Requirements tracing, consistency and gap analysis

Model queries play the role of *competency questions* and can be built on top of the semantic relations captured from/across models, with various possible goals:

- to trace requirements and their associated early designs up to the business context (including roles, activities, motivators that rely on them);
- to identify possible gaps, insufficiencies and conflicts in the modelled requirements.

Model queries can be performed on the Linked Data serialization of the models (as shown in Fig. 12), but also as dedicated functionality in the modelling tool. Figure 14 presents the results of a query (facilitated by a query builder component of ADOxx) from our proof-of-concept implementation.

Model queries are a useful tool for consistency checking. For example, the query results of Fig. 14 can be interpreted in several alternate ways: (a) that there is a gap in the requirements (hence the query works as a validator), (b) that there are business steps which do not require app support (hence the query works as a report generator), (c) that the requirements are not complete (hence the query works as a warning system); (d) that those activities have been inherited from the As-Is situation and they should be removed, being irrelevant in the envisioned To-Be (hence the query supports requirements analysis).

In order to distinguish between these situations, the property sets of most modelling objects are enriched with status attributes ("decided", "on hold", "under development", "implemented") and versioning attributes. These also become available to model queries, thus enhancing the semantic traceability with additional filters.

Consistency checking with error messages at "modelling time" is also possible, but with an impact on usability. For example, a message that warns the modeller every time he/she creates an activity without app allocation

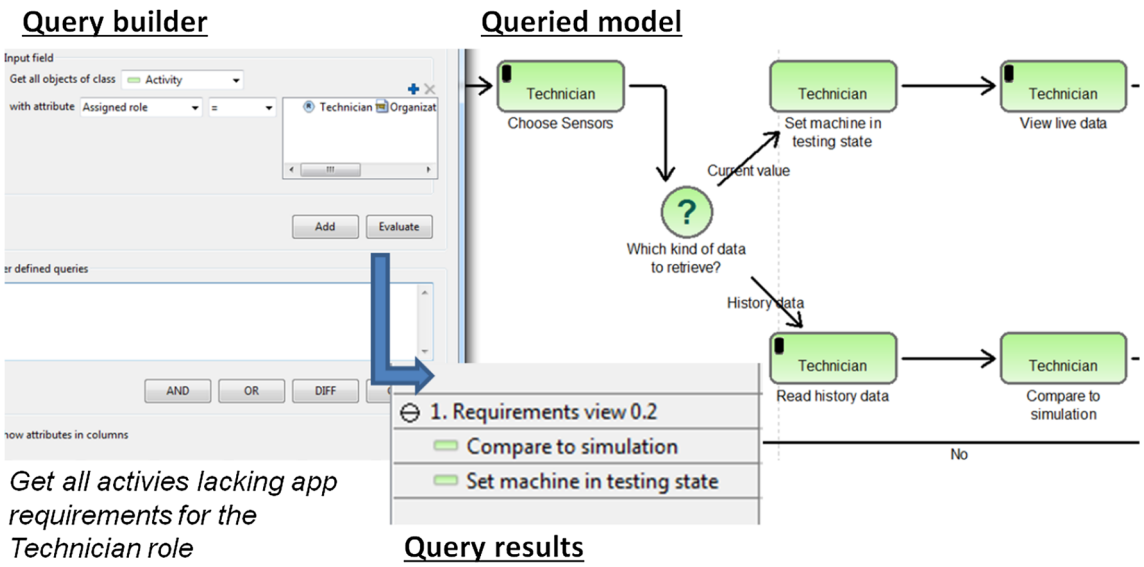


Fig. 14 Requirements tracing through model queries

tends to become increasingly annoying and does not consider the fact that requirements models evolve and may be partially designed for later refinement. Model queries acting as validators are significantly more beneficial.

Consistency checking in terms of relation domains, ranges and cardinalities is defined in the metamodel implementation itself (on the underlying metamodelling platform). The very basic actions of creating a model should be governed by the metamodel implementation, disabling the possibility of creating relations that are non-compliant with respect to their domain, range or cardinality.

Consistency by reasoning can be induced by production rules executed on the various relations (and relation variants introduced in Sect. 3.2) that are of interest for semantic traceability. Thus, the rules are employed to derive consistent relations rather than to check consistency for manually created ones.

For example, relative to the relations discussed in Fig. 12, we have the following rule-based derivations:

$$AT(x, y) \wedge Loc(y, z) \Rightarrow SemR2(x, y)$$

where *AT* and *Loc* are the metamodel relations *accessible_through* (from liable entities to access means) and *has_location* (from access means to physical locations) (see also their place in the metamodel—Fig. 11).

Or:

$$NotR1(x, y) \wedge Same(x, z) \Rightarrow SemR1(y, z)$$

where *NotR1* is the metamodel relation *aggregates* (from swimlane to process elements) and *Same* is an equivalence relation (similar to the ontological owl:sameAs) that our proof-of-concept tool allows to be manually created between any model objects (bypassing the prescribed metamodel).

Another example, one that derives syntactically induced relations is given by the orchestrator rules (from Fig. 13):

$$F(x, y) \wedge Req(x, a) \wedge Req(y, b) \Rightarrow F(a, b)$$

where *F* is the *followed by* relation (indicating the sequence in process models and in orchestration models), and *Req* is the *requires* relation (from process elements to their required resources).

6.4 The Interaction Stepper

This tool takes input from model serializations of the business process, its interaction processes and corresponding interaction element (outcomes of Steps 3 and 5 from the modelling procedure). As output, it guides the user through a step-by-step process-driven interaction (including its decisions), using the concrete notation mock-ups as app emulations.

As Fig. 15 indicates, for each process activity, and for each interaction step, the concrete notation of the assigned screen (component) is displayed together with the current activity attributes (e.g. involved roles). The current POI (determined by the link between interaction steps and interaction elements) is highlighted on the screen. Whenever a decision is met, the user is asked to select the preferred path.

This component supports the requirements validation loops between app modellers and the business stakeholder, as it facilitates the communication of envisioned app interactions in dynamic and intuitive mock-up views, for an early human-based evaluation. A secondary use may come in the training phase of a BYOD deployment strategy, where employees need to be accommodated with new or redesigned processes, particularly with the way in which the required apps will support those processes.

7 Empirical evaluation

7.1 Evaluation strategy

An evaluation of the proposed contribution may consider a wide range of criteria including *usability, usefulness* and *understandability*.

Usability is only briefly mentioned here, as it refers to the modelling tool rather than the designed artefact; therefore, it is largely dependent on the underlying metamodelling platform that was used for implementation. Moreover, the "fast prototype" status of the tool also means that usability refinements have been intentionally left for post-project productization, with only some explicitly requested productivity improvements being currently included (e.g. navigation across models via hyperlinks, the possibility of linking multiple elements at the same time, streamlining the RDF exporter to include functionality for uploading model serializations to a Linked Data server).

Usefulness is not an absolute quality, but one that must be related to requirements and change requests. The paper advocates the notion of "agile modelling method engineering" as an approach driven by *evolving requirements*—an alternative to the more rigid nature of standard modelling languages for which usefulness may be evaluated against global requirements of a domain (e.g. the ability to generate class definitions from class diagrams). The paper presents a relatively stable snapshot of the method evolution, without capturing longitudinal evaluations on intermediate iterations, or the design decisions that evolved with gradual emergence of domain knowledge/requirements. Further evolution is expected based on resources, feedback and requirements from follow-up projects.

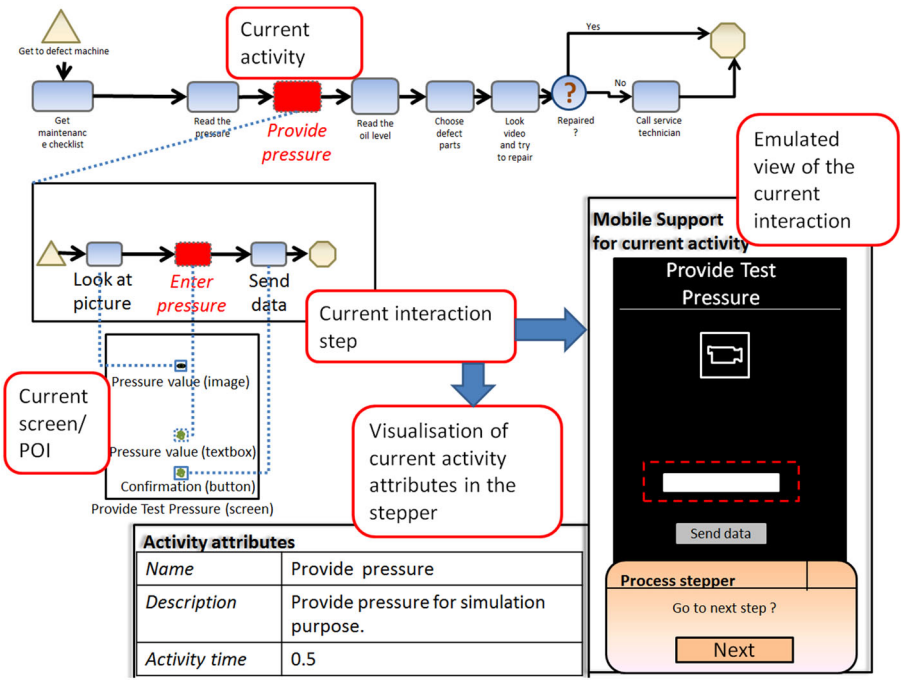


Fig. 15 Interaction stepper component

For the proposed modelling method, the requirements against which usefulness has been (iteratively) judged come from different sources:

- *Directly from project stakeholders*. In this category, we include both initial project requirements and iterative change requests;
- *Indirectly from run-time components* that need to consume model information. In this category, we include the app orchestration engine (for which the app orchestration model and some specific app description properties have been designed, together with the automated derivation of such models from multi-role business process models). In this respect, the semantics exposed by the model elements has been gradually aligned with the input expected by the execution engine;
- *Indirectly from the assumed research challenge of semantic traceability*, which inspired the RDF exporter functionality. This has also been used as a model serialization to satisfy the previous point (input for run-time components) and was validated against evolving project needs that dictated what properties and relations should be traceable across the models, following the core principles discussed in Sect. 6.1.

A comprehensive analysis in relation to the project requirements, as well as details on some usability improvements, is provided as a project report in [17] (deliverable 622). An overview on the "modelling requirements" and how they were reflected in the first iteration of the ComVantage method was provided in [12].

The following lists some improvements that are already incorporated in the work at hand, based on the project experience:

- The need to describe the business process in two distinct stages: the business view and the requirements view (addressed by Step 3 of the procedure);
- The POI taxonomy went through several versions in order to be easily assimilated by users and at the same time to provide structure for further processing (e.g. the interaction stepper functionality);
- The requirements gap checking functionality (addressed by model querying—Sect. 6.3).

Understandability is the focus of this empirical evaluation since it is a direct quality of the proposed work outside the project context and requirements.

The goal of understandability evaluation was to identify points in the modelling procedure where users unfamiliar with the project, but familiar with the key concept of a business process, may encounter obstacles in modelling tasks and model understanding.

For this purpose, an evaluation protocol was set up, involving subjects that had no familiarity with the project

and its modelling method. The protocol comprises several phases to be discussed in the following section. Additional details are also available in report [17]—deliverable D622 (however, they refer to the domain-specific parts of the modelling method).

7.2 Understandability evaluation: protocol and results

A set of 23 subjects was involved in the deployment of the protocol, all of them with an educational background in Business or Business Informatics, therefore with some existing familiarity with the notion of a business process (the key assumption on stakeholder skills) and in some cases with modelling tools. A first-contact survey helped to distinguish subjects who had previous experience with business process modelling (15 subjects) from those having no modelling experience (eight subjects). The results presented below also reflect the distinction.

7.2.1 Phase 1. Cognitive qualities carried by notations

The subjects were asked to guess, without prior documentation, which symbols have been defined for which concept names. A list of names and an extended list of notations (including a few standard ones) were provided, with the subjects attempting to match them. Experienced subjects were inclined to choose notations closely related to standards (the elements of the ER diagrams, most process elements) in the detriment of customized ones. The exercise also highlighted the value of domain-oriented notations for all subjects (e.g. the *App/feature*, the *App-supported activity*, the *POI* interpreted as UI control, the *Physical/Digital location*) compared to more generic ones (*Business role, Department*) or some considered too cryptic (*Access means*). After the correct answers have been revealed, subjects evaluated the overall visual intuitiveness at an average of 3.23 (on a 1–5 Likert scale), with experienced subjects at 3.08 and inexperienced ones at 3.5. This shows the preference for familiar standards in experienced subjects, while inexperienced ones rated higher those symbols that have been customized for visual cognition (showing less bias towards standards).

The next phases were based on grading the subjects' understanding of some given artefacts, expressed as *narrative descriptions of sample models* (phases 2 and 3) or as *models based on narrative scenarios* (phase 4). To avoid subjectivity, a single evaluator was involved. Concept-level granularity was applied, meaning that each example had a number of modelling concepts/relations (from the metamodel) of equal importance to be correctly described/represented. Three ratings were defined relative to some improvised "thresholds of understanding": 1. (misunderstanding or lack of understanding for more than 30 % of concepts present in the example), 2. (partial understanding, meaning that the semantics of some elements have been correctly described/represented), 3. (good understanding, meaning that most concepts—more than 70 %—have been correctly described/represented).

7.2.2 Phase 2. Description of sample models via story-telling, based on initial visual contact with sample models

Without previous training (other than the correct answers from Phase 1 used as a glossary), subjects had to describe in natural language the narrative that they can grasp from a given set of models with a complexity similar to the examples discussed throughout Sect. 4.2. Models of different types were grouped in five complex tasks (each typically covering two steps of the modelling procedure). The task-level counts for each grade (1–3) are displayed in Table 4. The last row indicates the percentage relative to the maximum possible rating, showing that on average subjects were able to describe about half of the information captured in models. The models pertaining to data requirements seem to be more challenging, and the business process models with assigned business entities the most intuitive (without any training).

An important aspect is the fact that each model communicates information on two levels: *visible* (whatever can be grasped by looking at a diagram) and *non-visible* (information that requires the inspection of editable properties, hyperlinks). Providing visual cues for every possible relation induces visual cluttering, whereas pushing more semantics outside the drawing canvas reduces visual understandability. Experienced users of standard notations (e.g. BPMN) expect to see all semantics reflected on notation level (thus having all relations *notationally induced*, according to Table 3). On the other hand, semantic traceability places focus on machine-readable relations which do not necessarily manifest on the canvas.

The overall conclusions of this phase impacted the development of modelling guidelines (to be used in training the users) and motivated the development of the modelling relation taxonomy described in Table 3.

7.2.3 Phase 3. Description of sample models after an introductory training

This is a reiteration of Phase 2 in a later session, after the subjects acquired some basic training (two hours of exercises and provision of the procedure documentation highlighting relations that can be created through other means than visual connectors on the canvas). This showed a general improvement in understandability (Table 5) with

Table 4 Rating counts for the ability to create narrative from models (before training)

		Business process and role modelling (Steps 1–2)	App requirements modelling (Steps 3–4)	Interaction modelling (Steps 5–6)	Data requirements modelling (Steps 7–8)	Orchestration modelling (Steps 3, 9)
Experienced modellers	1 count	6	7	10	9	7
	2 count	5	6	3	5	5
	3 count	4	2	2	1	3
Inexperienced modellers	1 count	5	3	6	6	5
	2 count	2	4	1	2	2
	3 count	1	1	1	0	1
Relative to maximum		58 %	57 %	48 %	46 %	55 %

an average of about 20 %—higher in the case of orchestration modelling and lower in the case of data requirements modelling which lags behind even after training (partly due to the high number of models involved, building up to a quite fragmented representation).

7.2.4 Phase 4. Creation of models based on user stories

This task was a reverse of the previous phase task—users had to create models based on some narrative, thus also acquiring a sense of procedure while advancing through the different types of models. This was strongly biased by tool usability, with subjects familiar with the user interface of ADOxx products (e.g. ADONIS CE [8]) performing better. Therefore, the focus was not on speed/time, but rather on task success and the identification of the modelling procedure points that raise obstacles for the task progress.

Table 6 uses the same grading scheme to evaluate the models, relatively close to the outcome of the previous phase (only slightly more difficult), and showing again weakness in data requirements, as well as some difficulty in interaction modelling (due to the *POI* taxonomy which was consequently simplified).

Based on a follow-up collection of feedback from the subjects, the difficulties with understanding the data requirements modelling approach is strongly influenced by (a) lack of familiarity with ER modelling and technological concepts pertaining to the Linked Data paradigm (*Query*, *Endpoint*) and (b) the need to work with many model types even for the smallest scenario. Therefore, a merging of the model types involved in data requirements modelling is considered for further evolution, as well as refined modelling guidelines to support users in expressing their data requirements on the level of detail they are familiar with, and avoiding a "take all or leave all" approach for the modelling procedure.

Other evaluation conclusions are synthesized as a SWOT analysis in the conclusive discussion Sect. 9.5.

8 Related works

This section briefly mentions related works, while a discussion against this baseline is given in the contribution summary (Sects. 9.1–9.3).

First of all, the viewpoints of [29, 48] (and their collected set of definitions) have been analysed with respect to the very nature of the "requirement" concept, which is questioned here by employing a modelling method as requirements elicitation means. The notion of "contextual requirements" has been investigated, using [1] and [20] as starting points, since the method aims to enable the tracing of requirements in relation to a business context. The Twin Peaks model [41] motivated the idea of an iterative requirements analysis approach that "bleeds into" early designs, adapted here for mobile app development.

Requirements engineering relies heavily on modelling languages [25], either UML-based (UML's use case diagrams [60], SysML [44]) or goal-oriented (i* [27], KAOS [18]). Each modelling language serves a particular approach, and efforts are being invested in transforming models between different approaches [35]. Serialization formats are available as enablers for further processing models: the more specific GRL [23] and ReqIF [42], or the more generic (UML-oriented) XMI [43]. The Eclipse community proposed a requirements structuring approach under the Open Requirements Management Framework [47], integrating concepts such as use case, process flow, artefact, actor in a high-level requirements description metamodel [49].

An investigation on the effectiveness of various negotiation channels [13] emphasized the role of computer-mediated communication tools in requirements elicitation. In contexts like those described in the study, our proposed models are content to be passed between business stakeholders and app designers, both in synchronous and asynchronous collaboration. Authors of [66] are also interested in bridging the gap between domain experts and software engineers by relying on mind maps to derive feature

Table 5 Rating counts for the ability to create narrative from models (after training)

		Business process and role modelling (Steps 1–2)	App requirements modelling (Steps 3–4)	Interaction modelling (Steps 5–6)	Data requirements modelling (Steps 7–8)	Orchestration modelling (Steps 3, 9)
Experienced modellers	1 count	1	2	2	3	1
	2 count	8	7	8	8	6
	3 count	6	6	5	4	8
Inexperienced modellers	1 count	1	1	2	3	1
	2 count	3	3	3	4	3
	3 count	4	4	3	1	4
Relative to maximum		78 %	77 %	72 %	65 %	81 %

models. The cited paper makes its proposals in the context of Software Production Lines [55], which conceptually supersedes the Industrial App Generation Framework proposed in the ComVantage research project [67], in the sense that they also promote model-driven code reusability (in our case this applies to the suggested app model portfolio).

Recent proposals of requirements modelling languages aim to compensate shortcomings of the traditional ones with superior semantic integration: a feature modelling language also covering behavioural capabilities as state machines is described in [52]; Ref. [6] proposes URML, a language that tries to capture in an integrated way both business rationale and product modularity. Such works converge towards the conclusion that requirements modelling needs to combine behavioural, intentional and structural aspects, and this is pinpointed as a semantic insufficiency of traditional, rather one-sided, approaches.

A language taking a more holistic approach, hence similar in spread to the work at hand, is RML [4, 50]. It provides designs for diagram types classified as People Models, Objective Models, System Models and Data Models. Some of them are mere predesigned tables, aiming to substitute the common use of Excel in gathering requirements with various types of matrices (e.g. use case centralizer, role/permission matrix, state matrix, decision tables, data dictionary, etc.). Others are diagrammatic in nature: the data flow diagram, the state diagram, decision trees, organizational charts and business process models. Our proposal also takes a holistic approach, but here the product under scrutiny is more specific: the mobile IT support to be developed and deployed, for example, in a BYOD environment. Our concept of *feature* is anchored in the interaction layer, as perceived by the end-user. It follows in the steps of the Cameleon framework [14] and its related serializations [59], which rely on the following abstraction layers: (a) tasks and concepts, (b) abstract UI, (c) concrete UI, (d) final UI.

A closely related piece of work (also inspired by the Cameleon framework) is [2], which gives a good summarized coverage of the interaction modelling literature (not to be replicated here). The authors align to the consensus that multiple abstraction levels are necessary, and define their own primitives mapped on the Cameleon layers. UML extensions for interaction modelling have also been proposed in the literature—see [40, 53]. Another approach to interaction modelling introduces the notion of *discourse models* [10] with the ambition of generating user interfaces.

With respect to the Linked Data requirements (given by the motivating project context), normally this technology is not something to be required explicitly by stakeholders, but rather a capability to fulfil particular business requirements. Most research work on Linked Data has been realized towards the goal of developing enabling tools or in promoting the growth of Linked Data Sets [33]. Shifting the focus to requirements is something to be handled in the business world, as business stakeholders must be made aware of benefits that are grounded in their business views. Companies providing semantic products are trying to cover this gap on a marketing level [51, 58]. Business motivations for Linked Data adoption are currently being refined by a W3C working group [63], and are described in the broader context of "Future Internet" by FInES, a project cluster and research community that provides an insightful roadmap for Future Internet Enterprise Systems [22]. All these sources converge towards recommending Linked Data as a data integration paradigm, subordinating all its other aspects, which inspired us to investigate what model-level signals can be relevant in a requirements analysis context.

9 Conclusive discussion

9.1 Contribution summary on requirements representation

The focus of the work at hand is on the engineering of a modelling method that enables the description of

Table 6 Rating counts for the ability to create models from narrative (after training)

		Business process and role modelling (Steps 1–2)	App requirements modelling (Steps 3–4)	Interaction modelling (Steps 5–6)	Data requirements modelling (Steps 7–8)	Orchestration modelling (Steps 3, 9)
Experienced modellers	1 count	2	1	2	5	0
	2 count	8	9	9	7	7
	3 count	5	5	4	3	8
Inexperienced modellers	1 count	1	1	1	4	1
	2 count	4	4	3	3	4
	3 count	3	3	3	1	3
Relative to maximum		74 %	75 %	70 %	59 %	81 %

requirements as a query-able knowledge structure which integrates business context with mobile interaction and data requirements, considering the relation variants that can be perceived on the level of diagrammatic modelling. The similarly holistic approach of RML [4] highlights the importance of business processes in requirements engineering, the need for extending traceability with inter-model links and the insufficiencies of UML in describing a business view with some domain specificity. The designers of RML have published an interesting evaluation for a comprehensive list of commercial tools [5] which confirms the core focus of the market on managing, rather than representing requirements. Representation is commonly handled through human-readable diagrams attached (as "artefacts") to entries from a use case/requirements catalogue.

RML aims to fill this gap, but it is merely a diagram language design which relies on hyperlinked templates (mostly Microsoft-centric, for Powerpoint and Excel), lacking in machine-readable semantics, an integrative metamodel and a dedicated tool with functionality built on top of the metamodel. The work at hand approaches this from the direction of the metamodelling scientific paradigm, also advocating the notion of "agile modelling method engineering" which can produce a modelling tool that supports inter-model linking with multiple purposes such as *usability* (navigation between related models) and *semantic traceability* (based on model queries, as an alternative to traceability approaches based on information retrieval techniques [56]). Our work introduces a novel taxonomy of modelling relations, which uses, as distinction criteria, the modeller perception on the existence of conceptual relations.

The ORMF initiative of the Eclipse community confirmed recurring concerns with requirements modelling and provided its own metamodels for this purpose [49]. However, the abstraction level was significantly higher than in the work at hand, as we try to retain business and technological specificity as first-class modelling citizens. The ORMF metamodels reflect the common perception that models are artefacts of textual requirements, while we are experimenting with the vision that models themselves should be seen as the main representations of requirements (with some textual annotations).

Therefore, the discussion can be furthered to the rather fundamental issue of "what is a requirement" and how this work deviates from common perception. The author of [28] states upfront that "there is no agreed definition of what counts as a requirement". Common definitions (as overviewed by [48]) use the term "statement", suggesting that requirements are "natural language items". Authors of [29] show that stakeholders only communicate *requirements representations*, and not the requirements themselves. To support this, we believe that modelling languages can be assimilated to logographic systems, having the ideogram (the notation) as the basic syntactical unit, hence models are statements formulated in such a language. Logographic statements composed in such "phrases" (inter-linked models) will semantically connect a requirement to its goal, context (the supported process and process motivators) and to its early design decisions (mock-up proposals) through different variants of modelling relations. The author of [48] also proposes his own definition, highlighting this semantic connectivity: "a *feature* of a *design* object that is necessary to achieve a *goal*".

Therefore, it should be apparent that the work at hand challenges and opens for debate several RE principles (mostly rooted in the waterfall development model):

- That the stakeholder must stay away from any design decisions. This often goes with the assumption (coming from an age when most projects were aiming to replace paper-based administration with IT systems) that stakeholders are technology-agnostics. Nowadays users are significantly more familiar with technology and business process management, and more IT projects aim to integrate or evolve legacy systems rather than to create systems from scratch. The stakeholder

Table 7 Mapping of the proposed approach on the Cameleon framework

Cameleon levels of abstraction	Cameleon artefacts	The proposed approach
Tasks and concepts	Task models (ConcurTaskTrees [62]) and domain models (UML class diagrams)	Describe business processes and involved roles in notations that are more familiar to the business stakeholder (than ConcurTaskTrees). Drill down business process steps to interaction steps using the same control flow language with extended semantics
		Describe domain concepts with simplified ER diagrams and highlight data ownership
Abstract UI	Inputs, Outputs, Facets	Collaboratively define functional elements of the app (POIs)
		Map interaction steps on structural elements and data requirements
Concrete UI	UI controls	Design wireframe mock-ups for requirements validation driven by interaction steps and their mappings
Final UI	Running code	We do not aim for code generation, but for model-driven code selection

familiarity with the business process management paradigm can be leveraged to refine the requirements granularity while retaining a sense of workflow on different levels;

- That software development phases should be strictly separated and sequenced. This was invalidated by the agile manifesto and model-driven paradigms that make development phases flow one into another gracefully (e.g. code generation from design to implementation). We try to stimulate a similar model-driven tight coupling between the phases of requirements analysis and design, a requirement emphasized by the Twin Peaks model;
- That the "why" (goal), the "what" (requirement) and the "how" (design decision) should be strictly separated. On a sufficiently small granularity level, the "what" tends to translate into the "how", and this is what the POI taxonomy aims to support. This is particularly noticeable for single-purpose apps—for example, the requirement of *getting* (*access to*) *the current sensor values* intuitively translates to a UI list displaying those values and to a *Sensor* data entity (maybe even some key attributes specified upfront by the stakeholder). It is then the choice of the designer to either build further on the stakeholder rudimentary models, or to propose innovative alternatives for covering the required POIs and data entities. This is the reason why the POI taxonomy is kept modality-independent and why the ER diagram should be seen as a concept map rather than a database design.

With respect to model serialization, compared to the existing requirements serialization languages based on XML, the work advocates an RDF serialization for two reasons: (a) to benefit from the graph-based traceability features of Linked Data paradigm; (b) due to an auxiliary research challenge (not emphasized in this paper) of bridging the worlds of conceptual modelling and Linked Data.

9.2 Contribution summary on mobile app requirements

The Cameleon framework [14] describes functionality in terms of interaction, on several abstraction layers, with the aim of enabling generative UI. The mapping of those layers on our approach is indicated in Table 7. These levels also inspired [2] to define primitives targeted to object-oriented code generation, with a richness of details beyond our scope (edit masks, queries, object mappings, etc.). Their conclusive section indicates a need for a requirements engineering approach that follows a similarly layered, interaction-oriented approach, and this is where our work is positioned. The work at hand favours specificity and context given by business processes, while execution-wise we are addressing model-driven app chaining [38] and code selection based on model information (the orchestration), rather than app code generation. Therefore, we use a specific notion of *context* compared to [1] (where context is a world state, possibly broken down by specific dimensions—task, social, personal, etc. as suggested by [32]) or [20] (where context is a human environment description including task, goal, organizational position). In our work, *context is given by the highest Cameleon layer* (tasks and concepts) *linked to auxiliary models from the high-level business view* (involved roles, locations, process motivators).

The *discourse modelling* approach of [10] shares with the Cameleon framework the ambitious goal of UI generation and the hierarchical approach to task representation, but it is rooted in communication theories. Discourse models occupy the same position as the hereby introduced *interaction processes.* Thus, we rely on the familiarity of

stakeholders with business process modelling rather than introducing a completely new language. In other words, interaction is intuitively approached as a process, rather than as a hierarchical task decomposition, and the POI taxonomy is introduced to describe "resources" required by such a process.

9.3 Contribution summary on Linked Data requirements

The related works on stimulating Linked Data adoption inspired the goal of signalling, with modelling means, opportunities for employing Linked Data. The data integration is technically achieved with query federation across multiple endpoints [64] or query orchestration (separated queries bound programmatically). The main contribution here is the proposal to employ the generalization of orchestration models (see final comments in Sect. 6.2) in order to justify the adoption of this technology in relation with business processes.

9.4 Degree of generality

Reusability beyond the scope where this work originates is impacted by two factors: *domain specificity* (the business domain for which the method has been designed) and *technological specificity* (the technology for which requirements must be modelled).

With respect to the first factor, the metamodel has been designed to include its *domain specificity* in a loosely coupled manner: it manifests in model types that can be linked to some elements of the hereby presented metamodel (e.g. as process motivators), but such links can be neglected and the requirements elicitation may start directly from business processes. By partitioning the metamodel in multiple inter-linked model types, a "take all or leave all" approach is avoided. Actually the metamodel discussed in this paper (Fig. 11) does not include the domain-specific parts at all. They are just hinted at as context of the work (in Step 1 of the procedure, Sect. 3) or as additional information that may be reached by the traceability queries (see top of Fig. 12). Concepts of other domains may extend the metamodel, and the RDF exporter will expose the newly added link and concepts to queries without any adaptation.

With respect to *technological specificity*, this limits reusability since it is embedded at deeper levels:

- In notation, e.g. symbols representing apps, app-supported activities, Linked Data queries, etc.;
- In semantics, through certain concepts and their prescribed editable properties, e.g. the concept of *app/feature* (with properties such as OS, device type, etc.); the pool of app capabilities; the subtyping of some concepts (*locations* can be SPARQL endpoints; *access means* can be SPARQL queries); the POI taxonomy was based on recurring patterns identified in app UIs and their interaction modalities; the interaction processes leverage the user perception that the app is a rich front-end providing units of functionality against loosely coupled data locations;
- In functionality: the requirement of deriving orchestration models on a per-role basis derives from the fact that each process participant uses a different device with a different deployed orchestration (as opposed to service orchestration where this is not inherently an issue); also, app descriptions include editable properties that are required by the run-time orchestrated app deployment engine.

Ultimately, any modelling approach must make a trade-off between *reusability* (typically addressed by standards that try to bring everyone on the same abstraction level), and *scenario-oriented customization* (typically addressed by agilely evolving modelling methods that retain specificity at various levels). It was an explicit goal for this work to lower generality level compared to more generic approaches (e.g. BPMN, UML activity diagrams). However, it must be noted that the specificity of ComVantage is not narrow, but rather multifaceted. Therefore, in the most general sense applicable here, we can state that *the modelling method facilitates the decomposition of a business model down to app and data requirements that can be passed towards the run-time side through an RDF serialization of models and their semantic links*. Since we are advocating an evolutionary approach to agile modelling methods, and the work presented here is a temporary fixed iteration, specificity may be further deepened based on what kind of information should become traceable in concrete applications.

9.5 SWOT evaluation of the proof of concept

A SWOT analysis has been performed on the current iteration of the modelling method based on the evolving implementation hosted by the Open Model Initiative Laboratory portal [46] for registered members, using ComVantage scenarios and relying on the RDF export capability to expose models to external components (for example the app orchestration framework [67]). All sample models presented in this work have been realized on this prototypical implementation. Modellers collaborate on the remotely accessible modelling tool, sharing and annotating models in a common model repository.

The conclusions of the SWOT analysis are as follows (weaknesses are discussed in more detail as basis for further work):

Strengths. The work at hand advocates a knowledge management approach for requirements elicitation, by creating a communication channel and a requirements representation space based on integrated conceptual models. Therefore, the proposed approach brings several benefits:

- It treats requirements representations as knowledge structures with a visual manifestation and semantic links that enable querying, analysis and traceability to the business environment or early design decisions;
- It exposes the semantic links and structure in a Linked Data (RDF) representation to leverage its relation-centric processing techniques;
- It leverages the wide adoption of business process management and the increasing stakeholder skills of describing business processes. This improves significantly the connection with the business view, the granularity of requirements and preserves a sense of flow that is lost in traditional elicitation approaches due to atomization of user stories and the representation of requirements as weakly structured item sets. Stakeholders are empowered by the modelling language to decompose their business view and to initiate rudimentary designs over the business-IT gap; therefore, requirements elicitation gets a tighter coupling with the design phase, reflecting the Twin Peaks vision;
- It advocates reuse of requirements models at later stages, to feed model-driven auxiliary components (e.g. for requirements analysis and validation, run-time app orchestration flexible deployment);
- It captures behavioural, structural and interaction aspects in a modelling method that can be further evolved in an agile fashion (in order to deepen the domain specificity and to lower the abstraction effort).

Weaknesses. There is a risk that the work is perceived as heavy-weight and excessively prescriptive (as a "take all or leave all" approach). It must be stressed that the modelling procedure should be read as guidelines—fragments can still be employed for disjointed purposes (business process modelling, sketching mobile app mock-ups or ER conceptual models, etc.) or for coarser granularity requirements. The method does not force the modeller to create all link/model types enabled by the metamodel, hence model completeness cannot be a strict goal here—requirements can be described with various levels of detail, as new insight is gained during requirements evolution. Model querying for missing links will guide the modellers through what they *can* do, rather than enforcing what they *must* do. Additionally, the procedure was designed as a granularity refinement process, inducing a gradual closeness between the stakeholder view and the developer view.

The empirical evaluation showed some understandability problems in the quite heavy part of data requirements modelling, which must be reconsidered in the further evolution of the method, possibly by merging some of the involved model types.

Regarding requirements management, the work at hand only briefly mentions properties that are relevant for managing requirements: authorship, responsibility, priority, difficulty, timestamps, build status, change tracking, etc. It has already been emphasized that we are currently interested in requirements representation rather than requirements management. Since the work takes a metamodelling approach, the modelling language is fully customizable and any attributes that are relevant from a management point of view can be added to all the presented concepts, including natural language descriptions to enhance model understanding. Model queries can aggregate reports from such attributes—however, the paper hints at this possibility without really pursuing its implications.

Regarding app design domain specificity, the POI taxonomy was designed to control granularity when describing app interaction (rather than to reflect a complete GUI control set). Specificity of the mobile domain can be improved by defining common UI patterns (status bar, document browser, etc.) as first-class modelling citizens. Such patterns can extend the capability pool towards the aim of accumulating a reusable vocabulary of UI patterns.

Regarding app orchestration, the currently employed graph transformation extracts an app usage precedence model acting as a rudimentary workflow. More prescriptive workflows can be derived by preserving the nature of the split nodes (XOR, AND) from the original business process. However, the orchestration deployment engine may also take input directly from the RDF serialization of the business process model if this level of detail must be prescribed in the app precedence flow.

Opportunities. Extended report generation mechanisms can be implemented on top of the RDF serialization, pushing the work towards the field of semantic information systems or even process-aware systems. This would rely on model queries for deriving aggregated requirements reports, which can be done both in the modelling tool and on the RDF serialization of models (via SPARQL queries).

The work lays some foundations for code reuse, under the assumption that app code is available and mapped on app models. Investigations on similarity between app models (hinted at in the procedure Step 4) can be furthered in a code reuse context scenario. Capability similarity can be extended with considerations on interaction process similarity (covered for business process in works such as [19]) to enable potential reuse of simple, single-purpose

apps or interaction patterns. This direction can also be furthered towards model-driven code generation, although significant implementation details must be added on modelling level.

The project context drives the research towards the app orchestration concept; however, new opportunities (and challenges) arise if the orchestration is seen as a facilitator for interaction process composition (if the developer chooses to implement a single complex app, instead of an orchestration of single-purpose apps).

Usability analysis evaluation can be performed with respect to the early mock-ups established in the requirements engineering phase, particularly with respect to interaction process simulation for quantitative usability metrics (e.g. number of interactions, screen distance to be covered in order to perform an interaction process).

Threats. As with any requirements elicitation effort, the collaborative steps of the modelling procedure may degenerate in fuzzy communication. Conceptual models, enhanced with the interaction stepper, are meant to minimize ambiguity and to channel communication using a minimal set of constructs. However, the semantics of these constructs and the modelling procedure as a whole must be assimilated by the modellers involved in elicitation with a certain overhead, a general problem for any modelling language. Natural language annotations can serve as annotations to all modelling objects, to reduce ambiguity and compensate the risk that the objects are misused during the language learning phase and a model refinement loop is recommended.

Model-based communication is heavier compared to natural language statement acquisition, but helps with flowing the requirements analysis phase into the design phase, while retaining a more granular formulation of the requirements rationale and context. Experience encouraged us to rely on the popularity of business process modelling and on the familiarity of business stakeholders with their legacy systems in order to reduce the overhead.

Finally, trained software engineers may prefer the universality of UML to describe not only apps, but also the business view, taking the full responsibility of structuring the business stakeholders "wishes". We, however, advocate the need for "agile modelling method engineering" which aligns to the same agile philosophy as modern software engineering: a metamodelling approach can tailor modelling tools to case specificity or change requests from specialized communities, thus providing shortcuts and lowering the abstraction effort when communicating through models.

9.6 Final conclusions and takeaway message

The work at hand employed a metamodelling framework to design modelling means representing process-centric app requirements that can be traced with semantic queries towards related context elements or early design proposals. We specifically target the BYOD environments, a natural next step of the cloud computing paradigm, shifting the "software as a service" deployment model towards a "service as an app" set-up. Enterprises have the necessary tools to store their data and knowledge, and to open them to employee-owned devices on a "need-to-know" basis. The proposed modelling approach can support decisions regarding what functionality and access means must be enabled for each role, process, or activity, and also to communicate this among all involved stakeholders and developers.

We see a modelling procedure as an integral part of a modelling method, rather than as the object of auxiliary consulting services. Instead, we place the focus of consulting on agile extensions for the language specificity.

With the rise of business process management practices and the ubiquity of mobile technology, stakeholders are more capable of stating what they need and when they need it (relative to their swimlane-based business processes). This is, however, merely an observation on our project experience—turning it into hypothesis should inspire interesting longitudinal studies.

The generalized takeaway message of the work at hand is summarized as follows:

(a) requirements representation should receive as much emphasis as requirements management;
(b) extensive and integrated conceptual modelling can be employed for this purpose, as it enables knowledge externalization over the business-IT gap (requirements elicitation becomes a knowledge acquisition effort);
(c) as models enable the software design phase to bleed into the coding phase during agile development, the requirements can also flow into the design phase with a proper inter-model integration;
(d) semantic traceability relies on the explicit representation or derivability of different variants of relations perceived by the modeller.

Acknowledgments The research leading to these results was funded by the European Community's Seventh Framework Programme under Grant Agreement No. FP7-284928 ComVantage.

References

1. Ali R, Dalpiaz F, Giorgini P (2010) A goal-based framework for contextual requirements modeling and analysis. Requir Eng 15:439–458. doi:10.1007/s00766-010-0110-z
2. Aquino N, Vanderdonckt J, Panach JI, Pastor O (2011) Conceptual modelling of interaction. In: Embley D, Thalheim B (eds) Handbook of conceptual modeling: theory, practice and research challenges. Springer, Berlin, pp 335–355
3. Apel S, Kästner C (2009) An overview of feature-oriented software development. J Object Technol 8(5):49–84. http://www.jot.fm/issues/issue_2009_07/column5.pdf
4. Beatty J, Chen A (2012) Visual models for software requirements. Microsoft Press, Redmond, Washington
5. Beatty J, Ferrari R, Vijayan B, Godugula S (2011) Seilevel's evaluations of requirements management tools. http://www.seilevel.com/download/9772/. Accessed Nov 2013
6. Berenbach B, Schneider F, Naughton H (2012) The use of a requirements modeling language for industrial applications. In: Proceedings of 20th IEEE International Conference on requirements engineering (RE 2012), IEEE, pp 285–290
7. Bizer C, Cyganiak R (2013) The TriG syntax specification. http://wifo5-03.informatik.uni-mannheim.de/bizer/trig/. Accessed Nov 2013
8. BOC-Group (2013) ADONIS community edition tool. http://www.adonis-community.com/. Accessed Nov 2013
9. BOC-Group (2013) ADOxx tool. http://www.adoxx.org/live/. Accessed Nov 2013
10. Bogdan C, Falb J, Kaindl H, Kavaldjian S, Popp R, Horacek H, Arnautovic E, Szep A (2008) Generating an abstract user interface from a discourse model inspired by human communication. In: 41st Hawaii international conference on systems sciences, IEEE, pp 36
11. Buchmann RA (2014) Conceptual modeling for mobile maintenance: the ComVantage case. In: 47th Hawaii international conference on systems sciences, IEEE, pp 3390–3399
12. Buchmann R, Karagiannis D, Visic N (2013) Requirements definition for domain specific modelling languages: the ComVantage case. In: Proceedings of BIR 2013, Springer, pp 19–33
13. Calefato F, Damian D, Lanubile F (2013) Computer-mediated communication to support distributed requirements elicitations and negotiations tasks. Empir Softw Eng 17(6):640–674
14. Calvary G, Coutaz J, Thevenin D, Limbourg Q, Bouillon L, Vanderdonckt J (2003) A unifying reference framework for multi-target user interfaces. Interact Comput 15(3):289–308
15. Carroll J, Stickler P (2013) The TriX syntax specification. http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html. Accessed Nov 2013
16. Cohn M (2004) User stories applied: for agile software development. Addison Wesley, Boston
17. ComVantage Research Project Consortium (2013) Project public deliverables. http://www.comvantage.eu/results-publications/public-deriverables/. Accessed in Nov 2013
18. Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1–2):3–50
19. Dijkman R, Dumas M, van Dongen B, Kaarik R, Mendling J (2011) Similarity of business process models: metrics and evaluation. Inf Syst 36(2):498–516
20. Fuentes-Fernandez R, Gomez-Sanz JJ, Pavon J (2010) Understanding the human context in requirements elicitation. Requir Eng 15:267–283. doi:10.1007/s00766-009-0087-7
21. FusionForge Tool (2013). https://fusionforge.org/. Accessed in Nov 2013
22. Future Internet Enterprise Systsems cluster (2013) The FInES research roadmap 2025. http://cordis.europa.eu/fp7/ict/enet/documents/fines-research-roadmap-v30_en.pdf. Accessed Nov 2013
23. Goal-oriented Requirements Language Resources (2013). http://www.cs.toronto.edu/km/GRL/. Accessed Nov 2013
24. Gordijn J, Akkermans H (2001) E3-value: design and evaluation of e-business models. IEEE Intell Syst 16(4):11–17
25. Greenspan SJ, Mylopoulos J, Borgida A (1982) Capturing more world knowledge in the requirements specification. In: Proceeding international conference on software engineering, Tokyo, 1982
26. IBM Rational Requirements Composer Tool (2013). http://www-03.ibm.com/software/products/en/rrc. Accessed in Nov 2013
27. I-star resources (2013). http://www.cs.toronto.edu/km/istar/. Accessed Nov 2013
28. Jackson M (2014) Topsy-turvy requirements. Requir Eng 19:107–111. doi:10.1007/s00766-013-0179-2
29. Kaindl H, Svetinovic D (2010) On confusion between requirements and their representations. Requir Eng 15:307–311. doi:10.1007/s00766-009-0095-7
30. Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) Feature-oriented domain analysis (FODA) feasibility study, Software Engineering Institute, technical report CMU/SEI-90-TR-021
31. Karagiannis D, Kühn H (2002) Metamodelling platforms. In: Proceedings of the 3rd International conference EC-Web 2002—DEXA 2002. LNCS 2455, Springer, pp 451–464
32. Krogstie J, Lyytinen K, Opdahl AL, Pernici B, Siau K, Smolander K (2004) Research areas and challenges for mobile information systems. Int J Mobile Commun 2(3):220–234
33. Linked Open Data 2—the EU Project page (2013). http://lod2.eu. Accessed Nov 2013
34. Loucks J, Medcalf R, Buckalew L, Faria F (2013) The financial impact of BYOD. http://www.cisco.com/web/about/ac79/docs/re/byod/BYOD-Economics_Econ_Analysis.pdf. Accessed Nov 2013
35. Microsoft Visio, the official website. http://visio.microsoft.com/en-us/pages/default.aspx. Accessed Nov 2013
36. Monteiro E, Araújo J, Amaral V, Goulão M, Patrício P (2013) Model-driven development for requirements engineering: the case of goal-oriented approaches. 8th International Conference on the Quality of information and communications technology (QUATIC 2012), IEEE CPS, pp 75–84
37. Moody D (2009) The physics of notations: towards a scientific basis for constructing visual notations in software engineering. IEEE Trans Softw Eng 35(5):756–777
38. Morgan J (2012) Guidelines for chaining iOS apps. http://usabilityetc.com/2012/05/guidelines-for-chaining-ios-apps/. Accessed Nov 2013
39. Mylopoulos J (1992) Conceptual modeling and Telos1. In: Loucopoulos P, Zicari R (eds) Conceptual modeling, databases, and case an integrated view of information systems development. Wiley, New York, pp 49–68
40. Nunes NJ, Cunha JF (2000) Wisdom: a software engineering method for small software development companies. IEEE Softw 17(5):113–119
41. Nuseibeh B (2001) Weaving together requirements and architecture. Computer 34(3):115–119
42. Object Management Group (2011) ReqIF documentation. http://www.omg.org/spec/ReqIF/1.0.1/. Accessed Nov 2013
43. Object Management Group (2011) XMI specification. http://www.omg.org/spec/XMI/. Accessed Nov 2013
44. Object Management Group (2012) SysML specification. http://www.omgsysml.org/. Accessed Nov 2013
45. Object Management Group (2013) UML resource page. http://www.uml.org/. Accessed Nov 2013

46. Open Model Initiative Laboratory (2013) ComVantage modelling prototype and resources. http://www.omilab.org/web/comvantage/home. Accessed Nov 2013
47. Open Requirements Management Framework (2008). http://www.eclipse.org/proposals/ormf/. Accessed Nov 2013
48. Ralph P (2013) The illusion of requirements in software development. Requir Eng 18:293–296. doi:10.1007/s00766-012-0161-4
49. Requirements model in ORMF (2008). http://wiki.eclipse.org/Requirements_Model. Accessed Nov 2013
50. Seilevel (2013) Requirements modeling language templates. http://www.seilevel.com/ba-resources/rml-requirements-visual-models/. Accessed Nov 2013
51. Sequeda J (2010) I believe linked data will enable new killer apps that are only possible thanks to linked data. http://www.semantic-web.at/news/juan-sequeda-x22-i-believe-linked-data-will-enable-new-killer-apps-that-are-only-possible. Accessed Nov 2013
52. Shaker P, Atlee JM, Shige W (2012) A feature-oriented requirements modelling language. In: Proceedings of 20th IEEE international conference on requirements engineering (RE 2012), IEEE, pp 151–160
53. da Silva PP, Paton NW (2003) User interface modeling in UMLi. IEEE Softw 20(4):62–69
54. Simon K (2010) SIPOC diagram. http://www.isixsigma.com/tools-templates/sipoc-copis/sipoc-diagram/. Accessed Nov 2013
55. Software production lines community (2013). http://www.softwareproductlines.com/. Accessed Nov 2013
56. Sundaran SK, Hayes JH, Dekhtyar A, Holbrook EA (2010) Assessing traceability of software engineering artifacts. Requir Eng 15:313–335. doi:10.1007/s00766-009-0096-6
57. Supply Chain Council (2013) The supply chain operations reference specification. http://supply-chain.org/scor. Accessed in Nov 2013
58. SWIRRL official website (2013). http://www.swirrl.com/publishmydata. Accessed Nov 2013
59. UsiXML—official page (2013). http://www.usixml.org/en/home.html?IDC=221. Accessed Nov 2013
60. Vidgen R (2003) Requirements analysis and UML: use cases and class diagrams. Comput Control Eng 14(2):12–15
61. W3C (2004) The RDF standard resources and specification. http://www.w3.org/RDF/. Accessed in Nov 2013
62. W3C (2012) Concur task trees submission page. http://www.w3.org/2012/02/ctt/. Accessed Nov 2013
63. W3C (2013) Linked data platform use cases and requirements. http://www.w3.org/TR/ldp-ucr/. Accessed Nov 2013
64. W3C (2013) SPARQL 1.1 federated query. http://www.w3.org/TR/sparql11-federated-query/. Accessed Nov 2013
65. W3C (2013) The SPARQL query language specification. http://www.w3.org/TR/sparql11-query/. Accessed in Nov 2013
66. Wanderley F, da Silveira D.S, Araújo J, Lencastre M (2013) Generating feature model from creative requirements using model driven design. In: Proceedings of the 16th Int. software product line conference, vol 2. ACM, New York, pp 18–25
67. Ziegler J, Graube M, Pfeffer J, Urbas L (2012) Beyond app-chaining—mobile app orchestration for efficient model driven software generation. Proceedings of the 17th IEEE international conference on emerging technologies and factory automation. IEEE, pp 1–8