

The Goals Approach: Enterprise Model-Driven Agile Human-Centered Software Engineering

Pedro Valente, Thiago Rocha Silva, Marco Winckler, Nuno Jardim Nunes

▶ **To cite this version:**

Pedro Valente, Thiago Rocha Silva, Marco Winckler, Nuno Jardim Nunes. The Goals Approach: Enterprise Model-Driven Agile Human-Centered Software Engineering. 6th International Conference on Human-Centred Software Engineering (HCSE 2016), Aug 2016, Stockholm, Sweden. pp.261-280, 10.1007/978-3-319-44902-9_17 . hal-01647713

HAL Id: hal-01647713

https://inria.hal.science/hal-01647713

Submitted on 24 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

The Goals Approach: Enterprise Model-driven Agile Human-Centered Software Engineering

Pedro Valente[1,2,3], Thiago Silva[1], Marco Winckler[1], Nuno Nunes[2]

[1] Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, Route de Narbonne, 118, 31400 Toulouse, France
[2] Madeira Interactive Technologies Institute (MITI), University of Madeira, Caminho da Penteada, 9020-105 Funchal, Portugal
[3] Software Applications Development Office, University of Madeira, Colégio dos Jesuítas, 9000-082 Funchal - Portugal
pvalente@uma.pt, rocha@irit.fr, winckler@irit.fr, njn@uma.pt

Abstract. Business Process Improvement (BPI) is a key issue in the development of the enterprise competitiveness. However, achieving a level of software development performance that matches enterprise BPI needs in terms of producing noticeable results in small amounts of time requires the existence of a comprehensive and also agile Software Development Process (SDP). Quite often, SDPs do not deliver software architectures that can be directly used for in-house development, as specifications are either too close to the user interface design or too close to business rules and application domain modeling, and produce architectures that do not cope with software development concerns. In this paper we present the *Goals Approach*, which structures business processes to extract requirements, and methodologically details them in order to specify the user interface, the business logic and the database structures for the architecture of a BPI. Our approach aims in-house software development in small and medium enterprises.

Keywords: Enterprise Engineering, Software Engineering, Human-Computer Interaction, Agile Software Development Process, Software Architecture.

1 Introduction

Software development within enterprises still lacks performance, and reports show that effectiveness is far from being achieved as software-project full-success rates in terms of time and budget are still as low as about 30% [1]. Furthermore, there is still a long way until software development is achieved in a patterned and predictable way regarding development effort, so it can be established as a consistent source of revenue following investment within enterprises [2].

Nevertheless, the advances of Software Engineering (SE) have taken us at least from a chaotic state of the practice [3], to a more inspiring situation where enhanced executive management support and increased user involvement in the Software Development Process (SDP) are appointed as factors for software project success [4].

adfa, p. 1, 2011.
© Springer-Verlag Berlin Heidelberg 2011

In our research, we investigated whether it would be possible to establish a direct relation between concepts valuable for enterprise management and the implementation of a supporting Information System. And by stating the hypothesis that, it is possible if a cross-consistent definition of concepts is established between the enterprise concepts that model its human interaction, and if this interaction specification is evidenced in the architecture of a software system.

In this paper we present the *Goals Approach*. The Approach was empirically developed following the application of different methods in order to maximize software development performance in a medium sized enterprise, filling the gaps left by the used methods in terms of business specification and software architecting. *Goals* targets tailored in-house development of Information Systems for Small and Medium Enterprises (SMEs), which is characterized by needs of agility concerning the supportive Software Development Process (SDP) as a way to allow the achievement of observable organizational changes in limited amounts of time [5]. *Goals* defines a SDP that applies a straightforward methodology that analyses the enterprise in a top-down process in order to produce an Enterprise Structure of valuable business concepts as requirements. The methodology continues by means of the detail of the Enterprise Structure components, in order to design and structure, also in a top-down process, the user interface, the business logic and the database (given an MVC architectural pattern [6]), and compose a final Software Architecture that can be used for software implementation management.

In short, our approach aims at establishing a cross-consistent bridge of enterprise and software concepts, and applies a methodology to derive them, which can be summarized in the following way (back-bone components are underlined): the human interaction is represented by means of Business Processes, User Tasks, User Intentions and User Interactions; the User Interface is represented by Interaction Spaces and Interaction Components; its Business Logic by means of Business Rules, User Interface and Database System Responsibilities, and the database by means of Data Entities and Fields.

The *Goals Approach* SDP is presented in Section 2. Its methodology is presented in Sections 3 (Analysis Phase) and 4 (Design Phase). The related work is presented in Section 5, conclusions are presented in Section 6, and future work in Section 7.

2 Software Development Process

Our approach Software Development Process (SDP) defines a Human-Centered Software Engineering (HCSE) methodology that integrates the Enterprise Engineering (EE) and Human-Computer Interaction (HCI) perspectives in the process of defining a Software Architecture for a given Business Process Improvement (BPI) problem.

The SDP defines an Analysis Phase that identifies Business Processes (BP, Step 1), User Tasks (UT, Step 2), Interactions Spaces (IS, Step 3), Business Rules (BR, Step 4) and Data Entities (DE, Step 5) in order to compose an enterprise model, the Enterprise Structure, by means of relating all the identified components of this Phase.

The Design Phase uses the Enterprise Structure in order to methodologically detail UTs using a Task Model (Step 6), an Interaction Model (in order to design the User Interface, Step 7), structure the Business Logic (Step 8) and the Database (Step 9), and elaborate a final Software Architecture (Step 10) of the Information System. Table 1 presents the SDP, including elements as inputs (I), output (O) or both (I/O) at each Step.

Table 1. *Goals* Software Development Process.[1]

	Analysis Phase					Design Phase				
Steps	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Business Inputs										
Enterprise Functional Description	I	I								
Business Regulations				I						
Business Concepts	I				I					
User Collaboration						I	I			
Enterprise Structure Components										
Business Process (BP)	O				I					I
User Task (UT)		O			I	I				I
Interaction Space (IS)			O		I	I	I			I
Business Rule (BR)				O	I			IO		I
Data Entity (DE)	O				IO				IO	I
User Behavior Specification										
User Intentions						O	I			
User Interactions							O			
Software Architecture Components										
Aggregation Space						O	I	I		I
Interaction Component						O	I	I		I
Interaction Object							O	I		I
User Interface System Responsibility						O	IO		IO	I
Database System Responsibility						O	IO		IO	I
Output Models										
Service Model	O	I								
Business Process Model		O	I							
UT – IS Relation			O	I						
UT – IS – BR Relation				O	I					
Enterprise Structure					O	I		I	I	I
Task Model						O	I			
Interaction Model							O	I	I	I
User Interface Design							O			
Business Logic Structure								O		I
Database Structure									O	I
Software Architecture										O

[1] I – Input. O – Output. IO – Input and Output.

The Software Development Process (SDP) presented in Table 1 presents the information used in each Step. The “Business Inputs” provide unstructured information that must be available in the enterprise domain in any format. The “Enterprise Structure Components”, “User Behavior Specification” and “Software Architecture Components” are elements that take part as inputs (I), outputs (O) or both (I/O) in the “Output Models” of each Step, and are used consecutively in the following steps, in a straight-lined process.

The application of the SDP presents a trade-off in terms of agility and traceability i.e. agility is constrained by the need to maintain traceability of all the elements as this is the primary foundation of the method. Traceability defines a structure between business and software concepts, that enables relating organizational changes in terms of changes in its supporting software system. Hence, the approach presupposes an initial effort for the documentation of a Software Architecture that later can be used to facilitate software development.

There are two distinct cases of changes. The ones that involve the Enterprise Structure, and the ones that are circumscribed to the Software Architecture. By the analysis of the SDP it is possible to identify both the Enterprise Structure (Step 5) and Interaction Model (Step 7) as core models of enterprise and of software respectively as both provide input for the final Steps (8, 9 and 10) of the SDP. Changes to the Interaction Model (Step 7) involve User Collaboration in order to specify interaction with the system, and involve a set of Software Architecture components that concern the support of a single User Task (UT). Oppositely, changes to the Enterprise Structure have a bigger impact in the Software Architecture, as this model also provides input for the Interaction Model (of Step 7) concerning a specific Business Process Improvement that involves Business Process and related UTs reorganization.

In any of the cases, the type of components (Enterprise Structure or Software Architecture) which are changed directly specifies the following Steps that need to be carried out. This is done means of the traceability of one Step components and Steps in which they are used as inputs when changed. In this way, by increasing the number of changed components the number of Steps that need to be carried out also increases, and consequently the software development effort also increases, providing a concrete perspective on the effort related to organizational and software changes.

In our approach, when the BPI does not imply the reorganization of the UTs of the BP, the method can be started from the Design Phase, which is the agile characteristic of the SDP. Directly relating our approach to the Agile Manifesto [7], it reduces the “need to follow a specific plan” other than the plan defined by traceability in Table 1. Defines “User Collaboration” (in Steps 6 and 7) for the specification of human interaction. And facilities the specification of a future architecture, with no need for further “comprehensive documentation”, avoiding the chaos that can be generated in software development when carried out without architectural-documentation support.

Following the Analysis and Design Phases, the process continues with the Implementation and Testing Phases (which detail is out of the scope of the this paper), and use the Software Architecture to guide software development, and the User Interface Design, Task Model and User Stories to guide the Information System test before deployment.

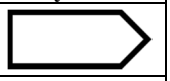
2.1 Foundations

The *Goals Approach* was developed by means of the continuous application of the *Wisdom* method [8] and its extension Process Use Cases Model [9] for the elicitation of requirements from business processes as Essential Use Cases (based on the *Activity Modeling* (AM) [10] method), in the process of architecting software for purposes of in-house tailored development in a medium-sized enterprise. The applicability of the architectural *Wisdom* method, and the relevancy of the representation of business process flows as sequences of Use Cases led to the definitive establishment of the combined software development method (initially named Goals Software Construction Process [11]) as it supported the team needs in terms of producing a programmable software architecture. The model enabled dialogue among stakeholders on BPI decisions, and allowed the identification of patterns of reusability concerning implementation.

The relation between business and software was further complemented by means of the inclusion of the concept of the DEMO method Action Rule [12] as the business-specific component of the Business Logic of the Software Architecture. This introduced a new separation of concerns which positively contributed to the organization of the remaining software-specific components. The Approach further benefited in terms of the theoretical validation of the patterned structure that relates enterprise and software concepts, as the *Goals* Enterprise Structure is compatible with the DEMO concepts of Transaction, Action Rule and Object Class. *Goals* adds to those concepts the notion of Interaction Space (IS) and the Goal of each Business Process (BP) that build-up a structure that provides the back-bone of the final Software Architecture.

This consolidated relation between enterprise and software concepts provides the core structure that allows the application of a methodological process that focus on user needs by means of the application of the BDD method [13]. The detail provided by BDD extends the application of the architectural *Wisdom* method in terms of physical interaction between the user and computer (clicks, keys, etc.), providing the base mechanism that specifies the User Interface, Business Logic and Database components with a level of detail usable for system programming.

Table 2. Enterprise Structure components definition, origin and symbol.

Component	Definition	Origin	Symbol
Business Process (BP)	*A network of UTs that lead to a Goal*	DEMO	
User Task (UT)	*A Complete Task within a BP*	AM	
Interaction Space (IS)	*The Space that supports a UT (with the same BRs and DEs)*	*Wisdom*	
Business Rule (BR)	*A Restriction over DEs Structural Relations*	DEMO	
Data Entity (DE)	*Persistent Information about a Business Concept*	*Wisdom*	

One particular view that matters concerning the methodology application, are the fundamental conceptual Enterprise Structure definitions on which it is based. The Enterprise Structure components, their definition, origin and symbol are presented in Table 2.

Goals defines as the top of the enterprise hierarchy, the Goal of the Business Process (BP). The BPs are composed by a series of User Tasks (UT), which once combined, lead the BP to the Goal. Those "Goals" are what names the approach. The human interaction between Actors that carry on, each his UT, happens in a given Interaction Space (IS) that makes available a series of Data Entities (DEs) that are subject to a number of Business Rules (BR) in order to be used by Actors. Each UT is considered complete when there is nothing that the Actor can do beyond his responsibility in order to further attain the Goal of the BP. This logic of the enterprise view provides the structure that is validated by means of the compatibility of concepts with the DEMO methodology, as every *Goals* component can be directly related or paternally derived from DEMO concepts. The difference between the two approaches is that DEMO does not consider the spaces where the human interaction happens, does not semantically structure the Business Process Goal, and cannot be directly related to the implementation parts of a Software Architecture, as DEMO defines separate ontologies for enterprise and software representation.

Figure 1 presents the relation between the main DEMO and *Goals* components, in which the DEMO concept of Business Process as an interrelated set of Transactions ("T1" and "T2" in the Figure) directly relates to the concept of BP of *Goals*. Furthermore, DEMO Transaction Acts ("rq", "pm", "st", "ac") related to *Goals* UTs in terms on consecutive Acts performed by the same Actor (e.g. sequence "T1 rq pm st ac" in "Pattern A", "T1 pm st" in A01 of "Pattern B" and "T1 pm T2 rq" in "Pattern C". And the Interaction Space identification is based on the space that is used by (between) Actors in order to support their interaction of any two UTs.

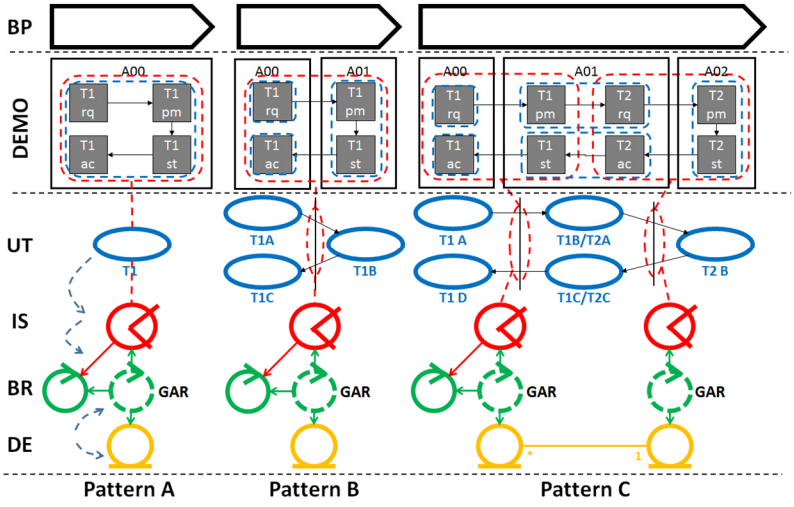


Fig. 1. Relation between DEMO and *Goals* main conceptual structures.

The mechanism that derives ISs from UTs and relates them to BRs and DEs serves as a bridge that relates business conceptual definitions (the BP and the UT) with the business-and-software-recognizable concepts of IS, BR and DE. This mechanism, that relies on the architectural *Wisdom* method for the identification of the IS, and the application of principle of merging consecutive UTs based on the Essential Use Case (EUC) definition application, serves as a door to the identification of business regulations (BRs) and business concepts (DEs) that must be available for the ongoing transactional process between two Actors, defining as a basic logic and structure for the enterprise functional description.

3 Analysis Phase

The Analysis Phase develops the Enterprise Structure, in which the Interaction Space (IS) concept is the mechanism that establishes the relation between the User Tasks (UT) of the Business Process (BP), and Business Rules (BR) that constraint existing Data Entities (DE). Each component is identified in a top-down methodological process in five Steps: Step 1 - Business Process Identification; Step 2 – User Task Identification; Step 3 – Interaction Space Identification; Step 4 – Business Rule Identification; and Step 5 – Data Entity Identification.

3.1 Step 1 - Business Process Identification

Goals defines a Business Process (BP) as "*A network of User Tasks that lead to a Goal*". The Goal is the objective, and also names the BP. It is expressed as a unique set of related enterprise business concepts (Data Entities, DE) that support its execution, and compose the enterprise domain model. The establishment of a relation between the BP and the set of DEs that the Information System will manage provides an increased awareness on the problem begin solved, and also increased communication capability between project stakeholders. Stakeholders in-depth their knowledge of the specific part of the enterprise that is being evolved. This facilitates the BPI development, and in practical terms results in faster and more productive project meetings, increasing the probability of developing projects in less time.

The relation between BPs and DEs is also useful in order to design the BP Model, which relates BPs, Actors and DEs, increasing the perception on how a BP uses and produces certain business concepts from a higher level of abstraction. We present the relation by means of the application of the Process Use Cases Model [9] adapted to the current *Goals* notation. The meta-model and an example are presente in Figure 2.

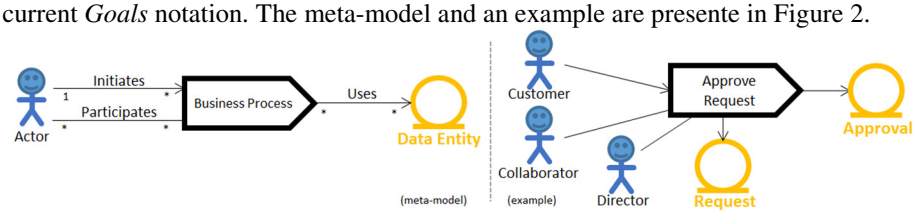


Fig. 2. Business Process Model meta-model, and BP Model example.

Figure 2 presents the meta-model of the BP Model, in which it can be read that only one actor can "Initiate" a BP, but an unlimited number of Actors can participate in it, and also, that an unlimited number of DEs can be used by a BP. It also presents an example where Actor "Customer" initiates the BP, Actors "Collaborator" and "Director" participate in it, and the DE "Request" is used and the DE "Approval" is produced.

3.2 Step 2 - User Task Identification

The User Task (UT) definition is derived from the concept of Essential Use Case (EUC) [10], which defines a Use Case as a "complete and meaningful task (carried out in relationship with a system)". This definition is adapted to the enterprise context based on the principle that the BP is a network of interrelated UTs, and that each UT is carried out by a single Actor, unless they are carrying out the same UT, performing cooperative work [14]. Since a BP always has a limited number of tasks, all UTs can be considered as meaningful, thus, we abandon the term "meaningful" and simply define a UT as a "*Complete Task within a BP*". We also apply the principle that an Actor (a User) never carries on two UTs consecutively and separately, which is a restriction that aims user performance and software development efficiency, in order to induce the reduction of the articulatory distance of the UT i.e. the user's effort [15], and suggest that the necessary tools should be provided using as little User Interface implementation space as possible. If two UTs are consecutive, then they can be merged in a single sequence of acts, expressed by a single UT, leading to is completion in the same way.

The relations between UTs are what designs a BP. The consecutive relation is the most common, as it supports the most usual BP flow. Yet, it is not sufficient to represent more complex services that must be available in different interaction points (also called as touchpoints) which usually have back-end support, and may be visited by the customer, but not necessarily in pre-defined order. This need for flexibility can be attained by the definition of conditional relations between UTs. Hence, we further define the conditional relation, meaning that the execution of a specific path of the BP is conditioned to the will of the responding Actor to carry on his task. This reflects the case when an enterprise suggests its customers the execution of a given action as a sequence of any other interaction, but will never be sure that they will follow the suggestion, and yet, continues to provide that customer the remaining service. The representation of services as a consecutive or conditioned sequence of UTs allows the representation of the service as a BP, and the possibility of well-defining a software architecture that paternally supports the service in a same way it supports the BP.

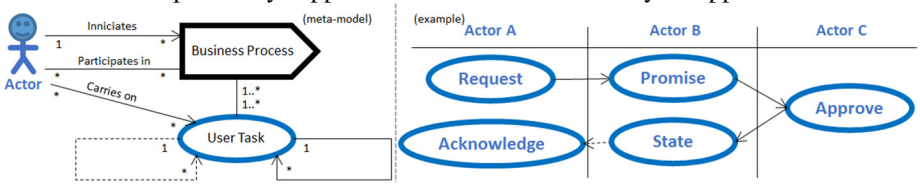


Fig. 3. User Task meta-model and example.

Figure 3 presents the meta-model of the UT, in which it can be read that: one Actor can carry on many UTs (and vice-versa); one BP can have one or more UTs (and vice-versa); and that one UT can consecutively or conditionally trigger one or more UTs. The example shows the initial UT being triggered by Actor "Customer" and consecutive UTs "Promise" and "Approve" being carried out by Actors "Collaborator" and "Director", and as the response tasks, "State" and "Acknowledge" being carried out by Actors "Collaborator" and "Customer" respectively. The relation between UT "State" and "Acknowledge" is conditioned to Actor "Customer" will to carry it on.

3.3 Step 3 - Interaction Space Identification

The Interaction Space (IS) definition is derived from *Wisdom* original concept of Interaction Space, as a space (a User Interface) where the "user interacts with functions, containers and information in order to carry on a task". We adapt this concept to the enterprise context by means of its generalization in order to consider the same purpose for the support of the UTs interaction in person, as in any of the cases, the same BRs and DEs also apply. *Goals* (re)defines the IS as "*The Space that supports a UT (with the same BRs and DEs)*". Hence, one IS supports the interaction between two users in person or remotely while each one carries on his own UT. Even if many UTs are carried out by many Actors in a cooperative way, the UTs will still be different, since at least one UT has initiated the other(s). If two Actors carry on the same UT remotely, then they are necessarily performing cooperative work [14].

The identification of ISs is derived from the interaction between the sequenced UTs of the BP, in order to support one Actor request and other Actor response, as in any case the same BRs and DEs apply.

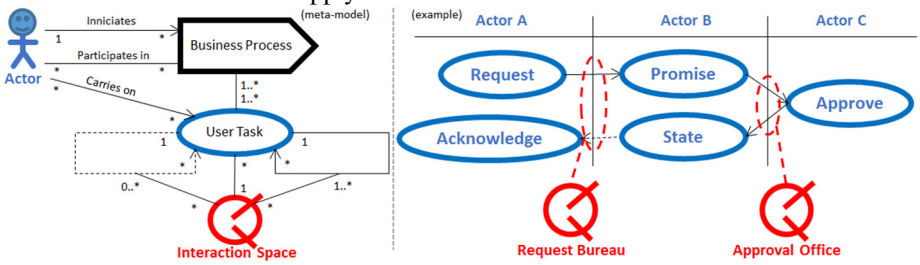


Fig. 4. Interaction Space meta-model and example.

Figure 4 presents the meta-model that specifies that an IS supports many UTs, having at least a consecutive relation and at most one conditional relation. The example shows the derivation of ISs that supports the interaction between Actors "Customer" and "Collaborator", and Actors "Collaborator" and "Director", by means of ISs "Request Bureau" and "Approval Office" respectively. This is based on the principle that the UTs that Actors operate in cooperation are subject to the same BRs and DEs. Hence, the "Request", "Promise", "State" and "Acknowledge" UTs that Actors "Customer" and "Collaborator" carry on cooperatively are supported by the IS "Request Bureau". The same happens with UTs "Promise", "State" and "Approve" and IS "Approval Request".

3.4 Step 4 - Business Rule Identification

The Business Rule (BR) definition is provided by DEMO notion of Action Rule, which defines a structure of decision (using pseudo-code) that applies restrictions to the identified Object Classes concerning the execution of business Transactions. These restrictions are paradigmatic relations (considering a semiotic association [16]) which are applied to the syntactic relations (also considering a semiotic definition) that exist between Data Entities (DEs), producing a new valuable business concept that cannot be expressed by the simpler relations between DEs. Hence, we define the BR as "*A Restriction over DEs Structural Relations*".

BRs represent regulations or explicitly defined requirements that should be elicited during the Analysis Phase in order to understand the constraints which the user is subject to when carrying on a UT. One important clarification is that BRs do not represent collaboration impositions between Actors, since these rules are already expressed by the BP design. BRs are the grounding foundation of the Information System Business Logic (given an MVC pattern), as they are the more business-specific programmed class concerning the structuring of this layer. The Business Logic will also be complemented with programmed parts that are responsible for the IS (User Interface) presentation and for the DEs (database) management, as will be presented in Step 8 – Business Logic Structuring.

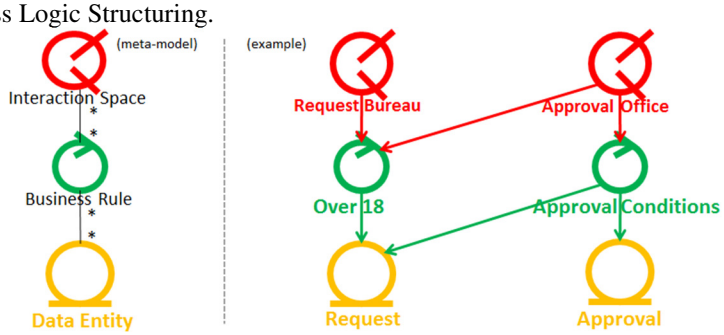


Fig. 5. Business Rules meta-model and example.

Figure 5 presents the meta-model concerning the relation between BRs, IS and DEs, in relations of many to many. The example shows that IS "Request Bureau" uses BR "Over 18", and that IS "Approval Office" uses BR "Approval Conditions". It also defines that BR "Over 18" uses DE "Request", and that BR "Approval Conditions" uses DEs "Request" and "Approval".

3.5 Step 5 - Data Entity Identification

The Data Entities (DE) definition is provided by *Wisdom* as a "class of perdurable information about a business concept". This means that persistency will be maintained by the Information System, and that it will enclose meaningful concepts which are recognized within the enterprise by those who have knowledge about it. The enclosed meanings (the concepts) can also be related between each other, allowing a

representation of reality by means of a computerized system which is made available for usage by means of a database application. These "meanings" are represented by Data Entities in *Goals*, and enclose attributes. In terms of common database objects, DEs are implemente by tables, and attributes are implemented by fields.

DEs are related between each other by means of the semiotic association of syntactic relations, which are expressed in *Goals* using an Unified Modeling Language (UML) [17] association, which also implies the definition of the multiplicity between the related DEs. The association multiplicity will typically be of one to many, or many to many. The definition of a specific multiplicity (e.g. 1 to 5) is uncommon, and should be expressed by a BR, as it is usually volatile (it will eventually change). The definition of relations of one to one is also uncommon, as in those cases the DEs meaning can usually be conciliated in a single DE.

As mentioned in Step 1 - Business Process Identification, the identification of DEs should be carried along the BP identification and the consequential Steps, so that the analyst at this stage already has a well-defined notion of the concepts involved in the BPI under analysis (and also how they relate between each other). In the current Step, the DEs only need to be identified and related to the BRs in order to compose the Enterprise Structure, the final artefact of the Analysis Phase, as depicted in Figure 6 with the DEs as a support of the Enterprise Structure.

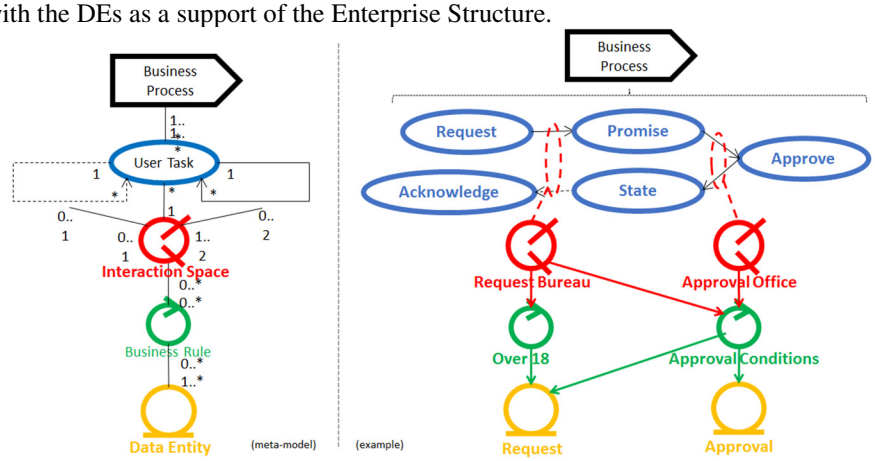


Fig. 6. Enterprise Structure meta-model and example.

The Enterprise Structure presented in Figure 6 is composed by every identified component until this moment and also by their relation to other components. It represents a relation which is representative of the enterprise in terms of a logic that relates BPs, UTs, ISs, BRs and DEs in terms of dependency and functional specification. It can be used in order to identify the implications of changing the enterprise in terms of its impact in the software structure, since, changing BPs, UTs or BRs, which is common in the business management domain, will inevitably change the underlying Information System to which the three lower levels layers (IS, BR and DE) are an inherent part.

The SDP continues with the elaboration of the Design Phase.

4 Design Phase

The Design Phase details the Enterprise Structure by means of the application of specific techniques that further specify and complement each component (Business Process (BP), User Task (UT), Interaction Space (IS), Business Rule (BR) or Data Entity (DE)) with new software specific components that structure the Software Architecture. The final Software Architecture is composed by the User Interface (View), Business Logic (Controller) and Database (Model) layers, which are related to the IS, BR and DE concepts respectively, given an MVC architectural pattern [6].

Each Software Architecture component is conceived in a top-down methodological process that details and completes the User Interaction (Step 6 - Task Model), the User Interface (Step 7 - Interaction Modeling), the Business Logic (Step 8 - Business Logic Structuring) and the DE layer (Step 9 - Database Structuring), and finishes with the composition and analysis of the Software Architecture (Step 10 - Software Architecture Composition).

4.1 Step 6 - Task Model

The Task Model details User Tasks (UTs) in order to obtain information to carry on the User Interface design, which happens in Step 7 - Interaction Modeling. The Task Model specifies the UT in terms of User Intentions (steps that the user takes to complete the task) and System Responsibilities (that provide the necessary information), following a traditional decomposition of an Essential Use Case (EUC) [10].

The decomposition of the UT in terms of User Intentions is carried out my means of the application of the Concur Task Trees (CTT) technique [18]. CTT defines the User Intentions as a hierarchical decomposition of what the user wishes to do in order to complete his task (the UT). This logic, is inherited from *Wisdom*, is maintained in *Goals*, and is represented using and UML Activity Diagram [17]. Each User Intention has an associated System Responsibility (SR) that provides the necessary information to an Interactive Component that supports User Interaction. The SR is a programmed class which is part of the Information System Business Logic, a layer which is composed in Step 8 – Business Logic Structuring. Interactive Components are spaces that provide the adequate implementation to allow data management, and are implemented by means of a User Interface programming language e.g. PHP.

The Task Model presents the flow of User Intentions that lead to the accomplishment of the UT. Each User Intention uses an Interaction Component by means of one or more User Interaction that in its turn also use System Responsibilities (SR) that supplies it with the necessary information. This relation is defined as the *Wisdom* architectural specification pattern i.e. the human-computer interaction happens in a User Interface part (the Interaction Component) and is supported by a programmed class (the SR). This type of SR is called as User Interface SRs. The last User Intentions of the Activity Diagram always lead to SRs that manages information, which in this case are called as Database SRs. If new Data Entities (DE) are identified by means of the Task Model elaboration, then they must also be represented in the DEs structure, which occurs in Step 9 - Database Structuring.

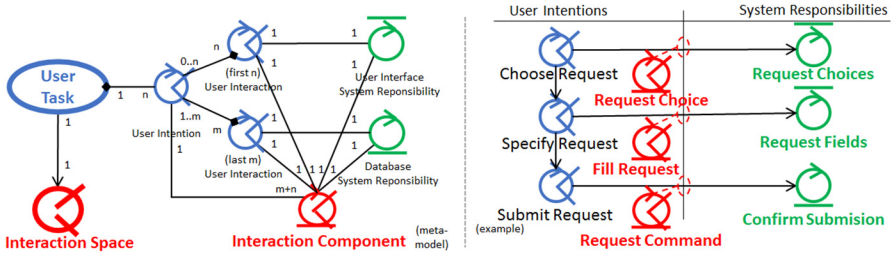


Fig. 7. Task Model meta-model and example.

Figure 7 presents the meta-model of the Task Model, where it can be read that a UT has up to *n* initial User Intentions, and up to *m* last User Intentions that use *m*+*n* Interaction Components (which compose the IS that supports the UT). Each Interaction Component supports one User Intention, and uses one User Interface System Responsibility (SR) or one Database SR. The example shows the decomposition of UT "Request", which has two initial User Intentions ("Choose Request" and "Fill Request") and one final ("Submit Request"). The first two relate to User Interface SRs "Request Choice" and "Fill Request", and the last relates to Database SR "Confirm Submission", meaning that the UT can be carried out by means of 3 interactions, which are supported by 3 System Responsibilities and 3 Interaction Components.

4.2 Step 7 - Interaction Modeling

The Interaction Modeling is carried out by means of the application of the Behavior Driven Development (BDD) method [13] that further specifies each User Intention as User Interactions, and also frames it in terms of used Interactions Spaces (ISs), specifying the navigation between the User Tasks (UT) of the Business Processes (BP). BDD is an agile software development method that describes the system behavior based on a User-Centered Design (UCD) perspective, producing pseudo-code for User Interface specification. BDD specifies User Stories that state that a system feature (a UT) which is used within a certain scenario (the IS), will result in specific system behavior which is expressed in the User Interface. The pseudo-code has the following syntax.

```
Given [State] When [Interaction] Then [System Behavior]
```

Where [State] represents the actual state of the system in the current scenario, the Aggregation Space [19], which is (are) the IS(s) where the UT occurs; [Interaction] is a flow of User Interactions that matches the User Intentions of the Task Model, specifying how the UT can be completed; and, [System Behavior] is the expected outcome that triggers User Interface and Database System Responsibilities. BDD interactions also specify the Data Entities (DEs) fields used in each User Interaction. This specification facilitates the mapping between Systems Responsibilities and DEs that occurs in Step 8 – Business Logic Structuring, and the completion of the Database specification that happens in Step 9 - Database Structuring.

BDD User Stories are represented by an Activity Diagram, and specify a User Intention that occurs before the Task Model in order to reference an IS, and details each User Intention using the pseudo-code which is presented in Table 3.

Table 3. Relation between BDD pseudo code syntax and Software Architecture components.

BDD pseudo-code	**Goals Component**
Given	(*provides Aggregation Space identification*)
Feature 'Feature'	User Task 'Feature'
Scenario 'Scenario'	User Intention 'Scenario'
Click, Choose, Set	User Intentions 'Click', 'Choose' or 'Set'
Display 'Page' or Go to 'Page'	User Interface System Responsibility 'Display Page' + Interaction Space 'Page'
Field	Data Entity Field
Then	(*last*) System Responsibilities

Figure 8 presents the User Stories User Interaction meta-model and an example that specifies the Task Model User Intentions using the pseudo-code presented in Table 3.

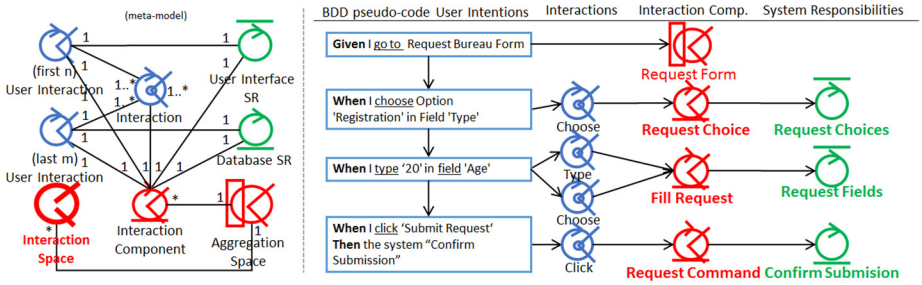


Fig. 8. User Interaction meta-model and example.

Now it is possible to design the User Interface by composing the generated components in each Interaction Component. Figure 9 shows a representation of the User Interface which identifies the Aggregation Space "Request Form", that uses the Interaction Space "Request Bureau", and the Interaction Components "Request Choice" that is composed of Field "Type", "Fill Request" which is composed of Field "Age", and the "Request Command" as the button "Submit Request", which trigger the User Interface SRs "Request Choices" and "Request Fields", and Database SR "Confirm Submission", respectively.

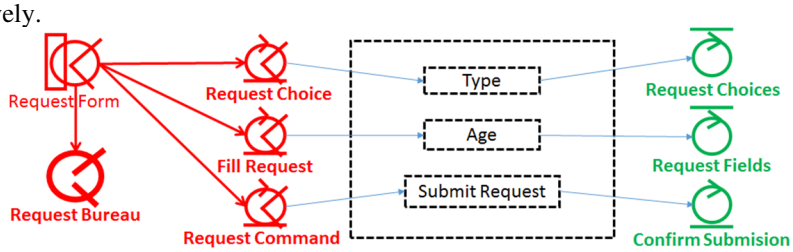


Fig. 9. User Interface Design example.

4.3 Step 8 - Business Logic Structuring

The Business Logic Structuring is carried out by defining the relations that each System Responsibility (SR) has to Data Entities (DEs), since the relation with the Interaction Spaces and Interaction Components is already established. The specification of each relation is dependent on the definition of to which DE the Fields identified in Step 7 - Interaction Modeling, belong to, which will also have an impact in the elaboration of Step 9 - Database Structuring.

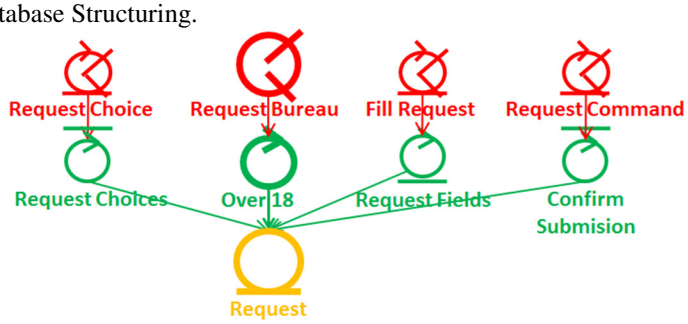


Fig. 10. Business Logic Structure example.

Figure 10 shows the manual mapping that was done between SRs and DEs. Business Rule "Over 18" is inherited from the Enterprise Architecture. User Interface SR "Request Choices" has been mapped to DE "Request", and it is assumed that Fields "Type" and "Age", belongs to DE "Request". By means of the analysis of the semantic of the Database SR "Confirm Submission" manages DE "Request".

4.4 Step 9 - Database Structuring

The Database Structuring is now possible since all the DEs are identified. Since two DEs (a and B) have been identified, and DE "Request" provides information for a given Field, it is possible to assume that DE "Request" only related to a single record in DE "Approval", yet, on the contrary, any record in DE "Approval" can be related to many records in DE "Request". Figure 11 presents the Database Structure.

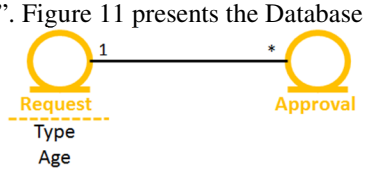


Fig. 11. Database Structure example.

4.5 Step 10 - Software Architecture Composition

The Software Architecture is the model that relates all the previously identified components in a single structure. It can be used to specify implementation responsibilities, and priority (within a software development team).

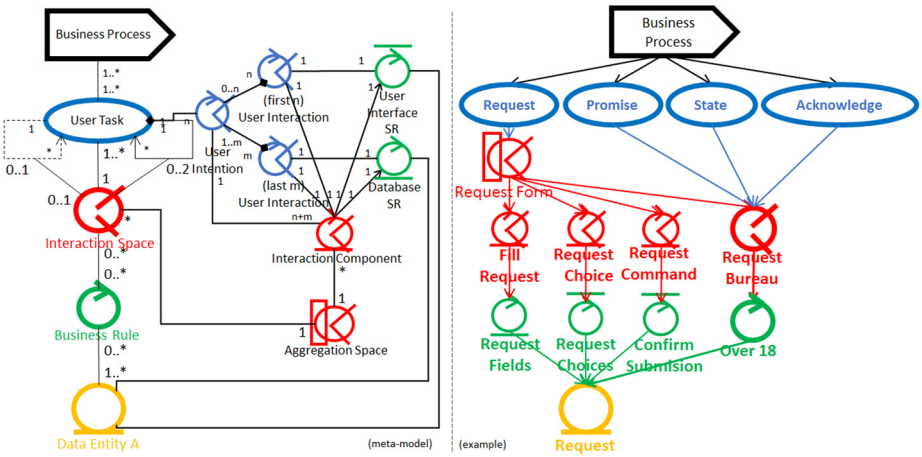


Fig. 12. Software Architecture meta-model and example.

The meta-model of the approach is presented in Figure 12, where it is possible to identify that the relation between BP and DE is now supported by means of the Software Architecture structure.

The implementation priority applied to the Software Architecture example, would be: DE "A" (since it will be used in) Business Rule "Over 18"; Database SR "Request Choices"; User Interface SR "Request Fields"; and only then User Interface SR "Confirm Submission". Interaction Components "Request Choice", "Fill Request" and "Request Command" can follow any order, and once developed, the IS "Request Bureau" and the Aggregation Space "Request Form" can be implemented and tested. For purposes of implementation, each IS will need a specific template per Actor in order to define its perspective according to "his" Task Model. The SRs will usually imply the definition of complex algorithms, and the Database will usually need the development of interfacing objects that facilitate data retrieval and update.

The architecture provides the advantage that the separation of implementation concerns is already defined at this stage, reducing significantly management efforts.

5 Related Work

Our approach can be compared to ArchiMate [20] and BPMN [21] in the perspective that it provides an enterprise and software structuring language. It is different in the perspective that it applies a methodology that derives software implementation specifications from business business models, in a business Model-Driven Architecture (MDA) process.

Concerning existing MDA approaches, and regarding the specific HCI perspective, the closest solutions are methods that design the user interface based on user task and domain models, as Sukaviriya's [22], Sousa's [23] and Cedar [24]. Our approach is different as it complementarily conceives the business logic layer based on enterprise business rules and coordination structures.

The I-Star framework [25] is a requirements engineering method that has a similar approach to *Goals* concerning the user perspective, as it considers the complete task (a UT) as a Goal that is decomposed in Tasks (which are User Intentions), that can be further decomposed in the same way that *Goals* further identifies User Interaction. The main difference between the approaches is that *Goals* further defines the supporting Information System.

There are more holistic MDA approaches to software architecting, like the Living Models [26], the Formal Design Analysis Framework [27], Zikra's [28], which also structure the business logic based on business process models, that yet, do not design the user interface.

Considering the enterprise-driven development, the Generic Software Development Process (GSDP) [29] is based on DEMO models, from which it derives the business rules, data structure, and business process design. And uses this information to conceive an enterprise operating system, which however, does not apply a structured user interface conception [30]. The Inter-enterprise Service Engineering (ISE) [31] uses BPMN business process models to automate the design of the user interface in detail, but, however, does not structure the remaining parts of the Information System.

6 Conclusions

Our approach inherently aims at facilitating requirements elicitation, focuses on user needs, and simplifies traceability between business requirements and software implementation, which matches project management needs and user involvement in the SDP, in what we believe to be the more important contribution of our work.

The validation of results, at this moment is mostly empirical, yet since the method as been developed by means of its application for over a decade, the techniques applied, including the notation, have been thoroughly revised. Five projects elaborated using this method were previously statistically analyzed for purposes of software effort estimation, from which it was possible to derive enhanced patterns of effort, which gives some guarantees about the stability of the development process [32].

Despite the existence of 10 Steps, the choice to maintain compatibility of concepts with Enterprise Engineering, Human-Computer Interaction and Software Engineering concepts, facilitates the understanding of the methodology by the specific domain professional experts. Yet, understanding it as a whole is more difficult as it crosses the enterprise and software domains, and for that reason it needs further application in order to be possible to inspect its usage in terms of effectiveness.

The introduced concept of Interaction Space (IS), as a framework of support for enterprise business-driven cooperative work is an extension of the traditional HCI interaction space that aims the simplification of the conception of the user interface. This simplification results in the specification of less implementation components and more manageable software architectures for a single BPI, resulting in more feasible and probably more successful software projects. This forecasts small BPI as a good strategy, since based on The Standish Group reports, projects under 1 M$ (one million dollars) cost are believed to be up to 10 times more successful than 10 M$ projects [1].

A controllable set of architectural components will usually be implemented with great efficiency (concerning work-hours) by programmers with knowledge of the domain. These circumstances induce iterative enterprise and information system development, matching the continuous software development proclaimed by the Agile Manifesto [7].

7 Future Work

Future work mostly concerns the continuation of the development of the approach concerning cooperative work, more specifically: a social perspective for the patterned conception of the user interface in terms of information visualization and tool execution permissions; a contextual perspective that facilitates user interface design decisions in terms of usability objectives; the User Interface design and prototype procedure specification; the elaboration of a business process model that supports the specification of cooperative work beyond the 2-actor swinlanes; the development of a Platform Specific Model for software generation; and the application of the approach by other software development teams as a strategy to further validate the presented techniques.

Ackownledgments

First author would like the acknowledge the scientific and finantial support from the afilliated institutions.

References

1. The Standish Group. *Chaos Report*. (2014)
2. Valente, P., Aveiro, D., Nunes, N., 2015. *Improving Software Design decisions towards enhanced Return of Investment*. In Proceedings ICEIS 2015. pp. 388-394.
3. Morgenshtern, O., Raz, T., Dvir, D. *Factors affecting duration and effort estimation errors in software development projects*. IST, V. 49, pp. 827–837. (2007)
4. The Standish Group. *Chaos Report*. (2013)
5. Gerogiannis, V., Kakarontzas, G., Anthopoulos, L., Bibi, S., Stamelos, I.; *The SPRINT-SMEs approach for software process improvement in small-medium sized software development enterprises*. In: Proceedings of ARCHIMEDES III. (2013)
6. Zukowski, J.: *The Model-View-Controller Architecture*. In: John Zukowski's Definitive Guide to Swing for Java 2. ISBN: 978-1430252511. (1999)
7. *Agile Manifesto*. Retrieved 02-05-2016: http://agilemanifesto.org/iso/en/
8. Nunes, N.: *Object Modeling for User-Centered Development and User Interface De-sign: The Wisdom Approach*. Phd Thesis. Universidade da Madeira. (2001)
9. Valente, P., Sampaio, P.: *Process Use Cases: Use Cases Identification*. Proceedings of ICEIS 2007. Vol. Information Systems Analysis and Specification, pp. 301-307. (2007)
10. Constantine, L.: *Human Activity Modeling - Toward a Pragmatic Integration of Activity Theory and Usage-Centered Design*. Springer. (2009)
11. Valente, P. Goals *Software Construction Process: Goal-Oriented Software Development*. VDM Verlag Dr. Müller. ISBN: 978-3639212426. (2009)

12. Dietz, J.: *Enterprise Ontology - Theory and Methodology*. Springer-Verlag Berlin Heidelberg. ISBN: 978-3540331490. (2006)
13. Chelimsky, D., Astels, D., Helmkamp, B., North, D., Dennis, Z., Hellesoy, A.: *The Rspec Book*. ISBN: 1934356379. (2010)
14. Grudin, J.: *Computer-supported cooperative work: History and focus*. IEEE Computer, Vol. 27(5), pp. 19–26.. (1994)
15. Winckler, M., Freitas, C., Palanque, P., Cava, R., Barboni, E.: *Usability aspects of the inside-in approach for ancillary search tasks on the web*. IFIP TC13 Conference on Human-Computer Interaction 2015 (INTERACT), pp. 207-226. (2015).
16. Damjanovic, V., Gasevic, D., Devedzic, V.: *Semiotics for Ontologies and Knowledge Representation*. Proceedings of Wissens Management, pp. 571–574. (2005)
17. Booch, G., Jacobson, I., Rumbaugh, J.: *The Unified Modeling Language Users Guide*. Addison-Wesley. (1998)
18. Paternò, F.: *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK. (1999)
19. Costa, D., Nóbrega, L., Nunes, N.: *An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes*. LNCS. Proc. of the 5th International Workshop, TAMODIA 2006, pp. 137-152. (2007)
20. Archimate Foundation.: *Archimate Made Practical*. (2008).
21. Völzer, H.: *An overview of BPMN 2.0 and its potential use*. In: BPMN 2010. Lecture Notes in Business Information Processing. V. 67. pp. 14–15. Springer. (2010)
22. Sukaviriya, N., Sinha, V., Ramachandra, T., Mani, S., Stolze, M.: *User-Centered Design and Business Process Modeling: Cross Road in Rapid Prototyping Tools*. In LNCS. Proceedings of INTERACT 2007. pp. 165-178. (2007)
23. Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: *User interface derivation from business processes: a model-driven approach for organizational engineering*. In: Proceedings of 2008 ACM SAC. pp. 553-560. (2008)
24. Akiki, P.: *Engineering Adaptive Model-Driven User Interfaces*. The Open University. PhD Thesis. (2014)
25. Aguilar, J., Zaldívar, A., Tripp, C., Misra, S., Sánchez, S., Martínez, M., García, O.: *A Solution Proposal for Complex Web Application Modeling with the I-Star Framework*. Proc. Int. Workshop on Software Engineering Process and Applications, pp. 135-145. (2014)
26. Breu, R., Agreiter, B., Farwick, M., Felderer, M., Hafner, M., Innerhofer–Oberperfler, F.: *Living Models – Ten Principles for Change–Driven Software Engineering*. International Journal of Software and Informatics. (2010)
27. Dai, L., Cooper, K.: *Using FDAF to bridge the gap between enterprise and software architectures for security*. Science of Computer Programming, V. 66, pp. 87–102.(2007)
28. Zikra, I.: *Integration of Enterprise Modeling and Model Driven Development*. Stockholm University. PhD Thesis. (2014)
29. Kervel, S., Dietz, J, Hintzen, J., Meeuwen, T., Zijlstra, B.: *Enterprise Ontology driven Software Engineering*. Proceedings of ICSoft 2012. (2012)
30. Hintzen, J., Kervel, S., Meeuwen, T., Vermolen, J., Zijlstra, B.: *A professional case management system in production, modeled and implemented using DEMO*. Proceedings of 16th IEEE Conference on Business Informatics. (2014)
31. Dividino, R., Bicer, V., Voigt, K., Cardoso, J.: *Integrating business process and user interface models using a model-driven approach*. Proc. of ISCIS 2009, pp. 492–497. (2009)
32. Alves, R., Valente, P., Nunes, N.: *Improving software effort estimation with human-centric models: a comparison of UCP and iUCP accuracy*. In Proc. of EICS '13, pp. 287-296. (2013)